

# **ESCUELA POLITÉCNICA NACIONAL**

## **ESCUELA DE FORMACIÓN DE TECNÓLOGOS**

**IMPLEMENTACIÓN DE PROTOTIPO DE UNA ALARMA  
RESIDENCIAL QUE PERMITA SU CONTROL A TRAVÉS DE  
*INTERNET***

**IMPLEMENTACIÓN DE PROTOTIPO DE UNA ALARMA  
RESIDENCIAL QUE PERMITA SU CONTROL A TRAVÉS DE  
*TELEGRAM Y DE UNA APLICACIÓN ANDROID***

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO  
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR  
EN REDES Y TELECOMUNICACIONES**

**KERLEY SANTIAGO GÓMEZ TAPIA**

**kerley.gomez@epn.edu.ec**

**DIRECTOR: LEANDRO ANTONIO PAZMIÑO ORTIZ**

**leandro.pazmino@epn.edu.ec**

**DMQ, agosto 2023**

## **CERTIFICACIONES**

Yo, KERLEY SANTIAGO GÓMEZ TAPIA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

**KERLEY SANTIAGO GÓMEZ TAPIA**

**kerley.gomez@epn.edu.ec**

**kerleygomez@outlook.com**

Certifico que el presente trabajo de integración curricular fue desarrollado por KERLEY SANTIAGO GÓMEZ TAPIA, bajo mi supervisión.

**LEANDRO ANTONIO PAZMIÑO ORTIZ**

**DIRECTOR**

**leandro.pazmino@epn.edu.ec**

## **DECLARACIÓN DE AUTORÍA**

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

**KERLEY SANTIAGO GÓMEZ TAPIA**

**kerley.gomez@epn.edu.ec**

**kerleygomez@outlook.com**

## **DEDICATORIA**

A mis padres, Rosa y Victor, quienes han creído en mí siempre, y me brindaron la oportunidad de finalizar mis estudios a pesar de las adversidades. Ustedes son la fuerza que necesité para nunca rendirme.

A mi hermano, Victor Hugo, mis sobrinos James, Alessandro y Deni por el apoyo que me otorgaron desde el principio. Y a mis familiares y amigos que me animaron de manera desinteresada para llegar hasta aquí.

Kerley.



## **AGRADECIMIENTO**

A la prestigiosa Escuela Politécnica Nacional por brindarme la oportunidad de forjarme en sus aulas. A la Escuela de Formación de Tecnólogos por todo el conocimiento impartido y que me ha servido en mi desarrollo personal y profesional.

A cada uno de los docentes que, con su valioso conocimiento y vasta experiencia, han desempeñado un papel fundamental en mi crecimiento y desarrollo académico. En especial agradecer a los ingenieros Fanny Flores, Gabriela Cevallos, Fernando Becerra, Andrés Reyes y Leandro Pazmiño. Sus esfuerzos y dedicación han sido una fuente constante de inspiración para mí.

A mi director de tesis Ing. Leandro Pazmiño quien gracias a su conocimiento impartido y su constante guía me permitió desarrollar este proyecto de manera correcta.

A mi amigo Jamie Reascos quien fue parte de todo este camino y me sirvió de ejemplo de que nunca hay que rendirse.

Kerley.

# ÍNDICE DE CONTENIDOS

CERTIFICACIONES .....	I
DECLARACIÓN DE AUTORÍA .....	II
DEDICATORIA .....	III
AGRADECIMIENTO .....	IV
ÍNDICE DE CONTENIDOS .....	V
RESUMEN.....	VIII
<i>ABSTRACT</i> .....	IX
1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO.....	1
1.1 Objetivo general .....	2
1.2 Objetivos específicos.....	2
1.3 Alcance.....	2
1.4 Marco Teórico.....	2
Seguridad residencial .....	2
Sistemas de alarmas .....	3
Redes Inalámbrica de Área Local .....	3
<i>ESP – WROOM – 32</i> .....	3
<i>Sensor PIR HC-SR501</i> .....	4
Sensor de contacto magnético.....	4
<i>Arduino IoT Cloud</i> .....	5
<i>Telegram</i> .....	5
2 METODOLOGÍA.....	5
3 RESULTADOS .....	7
3.1 Identificación de los requerimientos para el diseño del prototipo .....	7
3.2 Selección del <i>hardware</i> y <i>software</i> acorde a los requerimientos establecidos	8
Selección de <i>hardware</i> .....	8
Selección del <i>Software</i> .....	13
3.3 Diseño del prototipo de la alarma .....	15

Selección de plataforma .....	15
Esquema general del proyecto .....	18
Desarrollo del código en <i>ESP32</i> .....	19
Creación de la aplicación .....	44
Creación del <i>bot</i> de <i>Telegram</i> .....	50
Alimentación .....	51
3.4 Implementación del prototipo .....	52
Desarrollo de la placa de circuito impreso.....	52
Elaboración de la caja contenedora del circuito .....	54
Conexión de los sensores.....	55
Elaboración de la maqueta residencial .....	56
Montado del sistema sobre la maqueta .....	57
Ejecución del <i>bot</i> de <i>Telegram</i> .....	60
Instalación y acceso a la aplicación en <i>Android</i> .....	61
3.5 Realización de pruebas de funcionamiento del prototipo .....	62
Inicio del sistema de alarma.....	62
Cambio de clave de encendido o apagado .....	63
Revisión de sensores.....	64
Encendido y apagado de la alarma.....	68
Detección de movimiento mediante el <i>PIR</i> .....	75
Detección mediante el sensor magnético de la ventana .....	77
Detección mediante el sensor magnético de la puerta .....	79
Funcionamiento de la Alarma sin <i>Internet</i> .....	84
Reconexión del sistema a <i>Internet</i> .....	86
Funciones adicionales .....	87
Costo del prototipo.....	91
4 CONCLUSIONES .....	91
5 RECOMENDACIONES.....	93
6 REFERENCIAS BIBLIOGRÁFICAS.....	94

7	ANEXOS.....	97
	ANEXO I: Certificado de Originalidad.....	i
	ANEXO II: Enlaces.....	ii
	ANEXO III: Códigos Fuente.....	iii

## RESUMEN

El presente proyecto presenta la implementación de un prototipo de alarma residencial que permite su control desde *Internet* a través de una aplicación desarrollada en *Arduino IoT Cloud*, así como también por medio del *bot* Alarma\_casa\_bot de *Telegram*. El proyecto notifica a el usuario del *bot* de la intrusión detectada por medio de un sensor de movimiento, y dos sensores magnéticos. La aplicación alerta el cambio de estado de los *widgets*. En conjunto con la parte física del prototipo, el sistema alerta al dueño de la residencia mediante un sonido constante, el usuario permitido podrá apagar o encender la alarma por medio de la aplicación, el *bot*, o el teclado físico de la caja de la alarma.

En la primera sección se establece los objetivos a cumplir en conjunto con el componente desarrollado en el proyecto. Se detallan los principales conceptos necesario para la implementación del sistema.

La segunda sección se detalla la metodología seguida con la finalidad de cumplir con los objetivos planteados. La metodología permite establecer la secuencia seguida en el desarrollo del proyecto.

En la tercera sección se muestran los resultados obtenidos en el cumplimiento de cada objetivo. Se ha detallado los requerimientos del proyecto, la justificación en la selección de *hardware* y *software*, el diseño del código junto con la aplicación en *Arduino IoT Cloud* y el *bot* Alarma\_casa\_bot en *Telegram*, la implementación total del sistema sobre una maqueta que emula una residencia y finalmente las pruebas que comprueban el funcionamiento del prototipo.

Por último, se establecen las conclusiones y recomendaciones obtenidas durante el desarrollo del proyecto, así como las fuentes de investigación que sirvieron de sustento teórico y finalmente los anexos con el video de pruebas y el código final del sistema.

**PALABRAS CLAVE:** Alarma, *ESP32*, sensores, *Wi-fi*, *Arduino IoT Cloud*, *bot*, *Telegram*.

## **ABSTRACT**

*This project presents the implementation of a residential alarm prototype that allows its control from the Internet through an application developed in Arduino IoT Cloud, as well as through the Telegram bot Alarma\_casa\_bot. The project notifies the bot user of the intrusion detected by means of a motion sensor, and two magnetic sensors. The application alerts the widget status change. In conjunction with the physical part of the prototype, the system alerts the owner of the residence using a constant sound, the user will be able to turn off or turn on the alarm with the application, the bot, or the physical keyboard of the alarm box.*

*The first section establishes the objectives to be met in conjunction with the component developed in the project. The main concepts necessary for the implementation of the system are detailed.*

*The second section details the methodology followed to meet the proposed objectives. The methodology allows establishing the sequence followed in the development of the project.*

*The third section shows the results obtained in the fulfillment of each objective. The project requirements, the justification in the selection of hardware and software, the design of the code together with the application in Arduino IoT Cloud and the Alarma\_casa\_bot bot in Telegram, the full implementation of the system on a mockup that emulates a residence and finally the tests that verify the operation of the prototype have been detailed.*

*Finally, the conclusions and recommendations obtained during the development of the project are established, as well as the research sources that served as theoretical support and finally the annexes with the video of tests and the final code of the system.*

**KEY WORDS:** Alarm, ESP32, sensors, Wi-fi, Arduino IoT Cloud, bot, Telegram.

# 1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

La inseguridad se ha convertido en uno de los temas más debatidos hoy en día en el Ecuador. Hasta agosto del 2022 se reportaron 5496 robos a residencias, algo que preocupa a la población ecuatoriana y ha obligado a la implementación de medidas de seguridad como alarmas domiciliarias [1].

El presente proyecto ha desarrollado un prototipo de alarma residencial que permite ser controlado por medio de un *bot* en *Telegram* y de una aplicación en la plataforma de *Arduino IoT Cloud*. El prototipo ofrece una alternativa a los clásicos sistemas de seguridad, combinando lo ya existente con plataformas de *Internet of Things (IoT)*. El prototipo de alarmar permite detectar la presencia de un intruso mediante la implementación de sensores. Se ha utilizado un detector pasivo infrarrojo (*PIR*), que permite detectar el movimiento de una persona no autorizada, activando un *Light emitting diode (LED)* durante la detección, esta acción será notificada a él *bot* de *Telegram*, como también mostrar el estado de alerta en la aplicación de *Arduino IoT Cloud*.

La detección de intrusos en puertas y ventanas se ha garantizado mediante la utilización de sensores magnéticos, que notificarán a él *bot* de *Telegram*, así como alertar con el cambio de estado en la aplicación de *Arduino IoT Cloud* de la intrusión de una persona no autorizada. La activación de cualquier sensor mencionado alertará en el domicilio mediante la emisión de un sonido constante por parte de un *buzzer* y mensajes mostrados en una pantalla *Liquid Crystal Display (LCD)*, lo que permitirá alertar al usuario del prototipo de manera física, así como también por medio de *Telegram* y la aplicación.

También, se implementó un teclado matricial con la pantalla *LCD* y un *LED* para que el usuario pueda activar, desactivar y cambiar la clave principal de la alarma, de manera física, evitando que el prototipo solo funcione con conexión a *Internet*. El prototipo se ha desarrollado sobre el microcontrolador *ESP32*, con conectividad *Wi-fi* a *Internet* para la comunicación con las plataformas. Para un funcionamiento adecuado se ha desarrollado una placa *PCB* donde se soldaron los elementos. El *ESP32*, el *PCB*, la pantalla *LCD*, el teclado, los *LEDs*, el *buzzer* y los sensores mediante cableado, se colocaron dentro de una caja principal que funcionará como panel.

Todo el sistema de alarma se ha montado sobre una maqueta que emula una residencia clásica del Ecuador, al ser un prototipo, su control por parte del *bot* y la aplicación será

de uso exclusivo para el usuario que cuenta con las credenciales utilizadas durante el desarrollo, así como la clave para el manejo por medio del teclado físico. La alimentación del sistema será por medio de un cargador *Huawei* de 5 (V) y 1 (A), enchufado al puerto *micro-USB* del *ESP32*.

## **1.1 Objetivo general**

Implementar un prototipo de una alarma residencial que permita su control a través de *Telegram* y de una aplicación *Android*.

## **1.2 Objetivos específicos**

- Identificar los requerimientos para el diseño del prototipo.
- Seleccionar el *hardware* y *software* acorde a los requerimientos establecidos.
- Diseñar el prototipo de la alarma.
- Implementar el prototipo en una maqueta.
- Realizar pruebas de funcionamiento del prototipo.

## **1.3 Alcance**

Por medio del presente proyecto se busca implementar una alarma que permita alertar al dueño de la vivienda del ingreso de una persona no autorizada. Adicionalmente, tendrá la capacidad de realizar las siguientes acciones:

- Controlar la alarma por medio de una aplicación a través de *Internet*.
- Controlar la alarma por medio de *Telegram*.

## **1.4 Marco Teórico**

### **Seguridad residencial**

La seguridad residencial es un sistema que permita garantizar la vigilancia y una capacidad de respuesta adecuada ante un hecho que altere la integridad de una persona o de un inmueble, ante actos de vandalismo. Gracias a la integración con la tecnología se han logrado crear varias herramientas que permiten aportar con seguridad en los hogares, en donde destacan los sistemas de alarmas [2].



## **Sistemas de alarmas**

Un sistema de alarma es la unión de varios dispositivos que alertan a las personas sobre un suceso que amenaza su seguridad o la de un bien inmueble. Entre los distintos tipos de alarmas existentes se encuentran las antirrobo, las de incendio y las especiales, que tienen varias finalidades. Las alarmas no pueden evitar que se produzcan sucesos, pero sirven para tomar medidas disuasorias y de advertencia si estos ocurren. La probabilidad de ocurrencia de cualquier evento generador de pérdidas puede reducirse mediante la implementación de un sistema de alarma [7].

Gracias al avance del *IoT (Internet of Things)* los sistemas de seguridad han transformado su forma de monitorizar y alertar a sus usuarios. La gran red de *Internet* ha permitido el despliegue de sistemas de seguridad que recogen información de forma remota, realizan monitorización y crean alertas de intrusión [8]. Por eso el proyecto se centra en implementar una alarma que permita su control por *Internet*.

## **Redes Inalámbrica de Área Local**

Las redes de área local inalámbricas (*WLAN*) permiten que un gran número de dispositivos se comuniquen mediante ondas electromagnéticas. Gracias a este principio, es uno de los medios de transmisión más utilizados hoy. El uso de bandas *Industrial, Scientific and Medical (IMS)*, sin licencia, permite un gran despliegue de servicios, especialmente en los hogares, a través del estándar 802.11 o *Wireless Fidelity (Wi-fi)* [3]. Es por ello que el presente proyecto utilizará en su desarrollo el estándar 802.11b/n/g, en la frecuencia de 2.4 (GHz), para la comunicación del microcontrolador.

## **ESP – WROOM – 32**

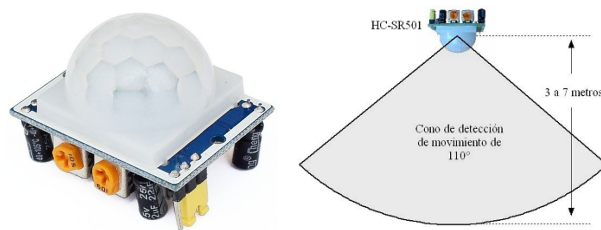
El *ESP32*, ver Figura 1.1, es un chip con conectividad inalámbrica gracias a que cuenta con *Wi-fi*, *Bluetooth* y la compatibilidad con la frecuencia de 2.4 (GHz). Actualmente, gracias a su diseño robusto, su bajo consumo de batería y su capacidad para soportar varios lenguajes de programación, es uno de los microcontroladores más utilizados para el desarrollo de proyectos *IoT*. Entre sus características más destacadas se encuentran: Memoria *flash* de 4 (MB), tensión de funcionamiento de 3.3 (V) a 5 (V), 34 pines digitales y soporte para módulos I2C [4][5]. Gracias a estas características es el microcontrolador con el que se desarrollará el presente proyecto.



**Figura 1.1 ESP32 [4]**

### **Sensor PIR HC-SR501**

El detector pasivo infrarrojo (*PIR*), permite detectar el movimiento realizado por una ser humano o animal, este tipo de sensor es uno de los más utilizados en sistemas de seguridad. Su funcionamiento se basa en la detección de la radiación electromagnética infrarroja debido al aumento de temperatura en los seres vivos. Su rango ajustable permite tener un alcance de detección de 3 a 7 metros y con un cono de movimiento de 90 a 110°[6][9]. Como se muestra en la Figura 1.2.



**Figura 1.2 PIR [9]**

El sensor *PIR* será el encargado de la detección de movimiento en el desarrollo del proyecto.

### **Sensor de contacto magnético**

El sensor de contacto magnético, que se visualiza en la Figura 1.3, cuenta con un sensor y un imán, al juntarse las piezas actuará como cerrado, y al separarse como abierto, de esta manera se lo utiliza en puertas o ventanas para la detección de intrusión en la seguridad [10], su facilidad al implementar lo convierte en uno de los más utilizados en proyectos de seguridad.



**Figura 1.3 Sensor magnético [10]**

## **Arduino IoT Cloud**

Es una de las plataformas más utilizadas para implementar proyectos *IoT*. Su compatibilidad con los microcontroladores de *Arduino* la convierten en una opción muy utilizada. Además, de ofrecer muchas de sus principales características de manera gratuita es muy amigable con el usuario y muy fácil de utilizar. Sus *widgets* y paneles permiten que los proyectos *IoT* se puedan monitorear de manera efectiva. También, Su servicio de nube ofrece seguridad a la información con la que se manejan sus servicios [11]. Esta plataforma permitirá desarrollar de manera correcta el presente proyecto.

## **Telegram**

Es una de las aplicaciones más utilizadas en el mundo, donde su principal servicio es de mensajería con alta seguridad y velocidad. Cuenta con más 700 millones de usuarios colocándola dentro de las 10 aplicaciones más descargadas. Entre sus servicios destaca la posibilidad de crear *bots* que facilitan llevar a cabo tareas automatizadas como buscar información, interactuar con usuarios y controlar placas de desarrollo como las de *Arduino* [12].

La amplia información para crear proyectos combinados con placas de desarrollo la convierten en una plataforma óptima para llevar a cabo el presente proyecto.

## **2 METODOLOGÍA**

El proyecto se desarrolló cumpliendo cinco objetivos claves, que permitieron implementar una alarma residencial que permite su control a través de *Internet*. La metodología que se ha empleado se puede visualizar en la Figura 2.1.



**Figura 2.1** Metodología empleada

Para iniciar, mediante una investigación detallada, se identificaron los requerimientos necesarios para el diseño del prototipo, para garantizar el cumplimiento del alcance establecido, y con los que se establecieron las principales características que debe cumplir el prototipo de alarma.

Una vez identificados los requerimientos del prototipo de alarma, se hizo un análisis para determinar las características del *hardware* y *software* necesario para detectar el ingreso de una persona no autorizada a un área restringida. En la sección de selección de *hardware* se evaluará las características de varios microcontroladores para justificar la elección del módulo *ESP32*, igual que los demás componentes físicos, como sensores, elegidos según los requerimientos establecidos para el prototipo.

En la sección de selección de *software* se generaron las características de las plataformas existentes y que permiten desarrollar códigos abiertos, los cuales son compatibles con los microcontroladores. La elección de la plataforma se realizará mediante la comparación de las características principales, donde se justificará la elección de *Web Editor* como la plataforma para el desarrollo del código.

A continuación, se realizó el diseño del prototipo mediante una investigación para determinar qué plataformas se emplearán para el desarrollo de la aplicación y para la creación del *bot* de *Telegram*. La elección de la plataforma para la creación de la aplicación en *Android* se realizará mediante una comparación de las distintas existentes en el mercado, justificando la elección de *Arduino IoT Cloud*, aplicación que permitirá el control de la alarma juntamente con el *bot BotFather* de *Telegram*.

Además, se crearon los códigos de los programas que deberán ser programados en el microcontrolador y finalmente la creación de la aplicación y del *bot*, también se detalló el esquema del prototipo, se establecieron las conexiones realizadas y se creó las instrucciones en base al código para permitir el control de la alarma, su encendido, apagado y monitoreo por parte de la aplicación y de *Telegram*, así como la utilización del teclado físico.

Finalmente, una vez seleccionado el *hardware*, *software* y creados los códigos para el prototipo, se realizó la implementación del sistema, uniendo las partes desarrolladas y montando en una maqueta creada para evidenciar el funcionamiento del prototipo del sistema. Completada la implementación del prototipo, se realizarán pruebas de funcionamiento que permitirán validar el correcto desempeño del prototipo. Los resultados de las pruebas se usaron para subsanar errores si existen y así se garantizará que el prototipo cumpla con todos los requerimientos.

### 3 RESULTADOS

En la presente sección se detallan las cinco fases del desarrollo del prototipo; inicialmente se identifica los requerimientos del prototipo. Luego, se define el *hardware* y *software* acorde a los requerimientos del prototipo; seguido, se realiza el diseño e implementación del prototipo. Finalmente, se llevan a cabo las pruebas necesarias del funcionamiento del prototipo.

#### 3.1 Identificación de los requerimientos para el diseño del prototipo

El prototipo requiere una conexión *Wi-fi* ya que se mantendrá fijamente en un sitio y accederá a una conexión a *Internet* de manera inalámbrica, lo que permitirá la comunicación con las plataformas. Para esto, se utilizará la tecnología 802.11b/g/n en la frecuencia no licenciada de 2.4 (GHz). La alimentación de energía del prototipo se realizará mediante una conexión cableada al tendido eléctrico del domicilio donde se implementará.

El prototipo debe permitir la detección de movimiento, por lo que se empleara un sensor *PIR* conectado al microcontrolador, también se utilizarán dos sensores magnéticos colocados en una ventana y una puerta, la alerta de intruso será mediante un *buzzer*. Al existir movimiento, el sensor *PIR* se acompañará de un *LED* rojo que se encenderá o apagará según la detección. También, se colocará un *LED* que permita saber si el prototipo de alarma está encendido o apagado.

Para controlar el prototipo por *Internet*, se requerirá el desarrollo de una aplicación que pueda ser utilizada en un dispositivo *Android*. Esta aplicación permitirá monitorear el estado del prototipo de alarma, así como activarlo o desactivarlo. El desarrollo del proyecto requerirá contar con las credenciales de acceso a la aplicación, ya que al ser un prototipo se diseñará para uso exclusivo de un usuario.

Además, se creará un *bot* de *Telegram* que permita controlar la alarma y recibir las notificaciones enviadas cuando se detecte movimiento con el *PIR* o un intruso acceda a través de los sensores magnéticos de la puerta y ventana. El control del *bot* será exclusivo de un solo usuario, con la finalidad de ofrecer seguridad en el sistema, para lo cual será necesario contar con el ID del usuario de *Telegram* desde el inicio del desarrollo del proyecto.

Se ha optado que la recepción de las notificaciones sea a través de *Telegram*, ya que el equipo del usuario debe contar con la aplicación de *Android* y *Telegram*, evitando así posibles conflictos. Tanto la interfaz de la aplicación como la de *Telegram* serán diseñadas de manera amigable para facilitar su uso por parte del usuario.

Ya que el prototipo se basa en utilizar *Internet*, para comunicarse con las plataformas, se ha optado que el control de la alarma se pueda hacer con un teclado que permita activar y desactivar la alarma, con el ingreso de una clave, si no existe conexión a *Internet*. El sistema tendrá una clave por defecto, que podrá cambiar el usuario con el teclado físico.

Todos los elementos físicos que componen el *Hardware* mantendrán la comunicación con las aplicaciones mediante el desarrollo del código en un *Software* compatible con el microcontrolador. El circuito final, junto con los elementos serán montados en una caja adecuada, que proteja los elementos, y se verificará su funcionamiento mediante una maqueta.

## **3.2 Selección del *hardware* y *software* acorde a los requerimientos establecidos**

### **Selección de *hardware***

En el mercado existen placas de desarrollo y sensores que permiten crear proyectos electrónicos, la elección requiere de un análisis de sus características técnicas y evaluar cuáles cumplen con los requerimientos planteados para este proyecto.

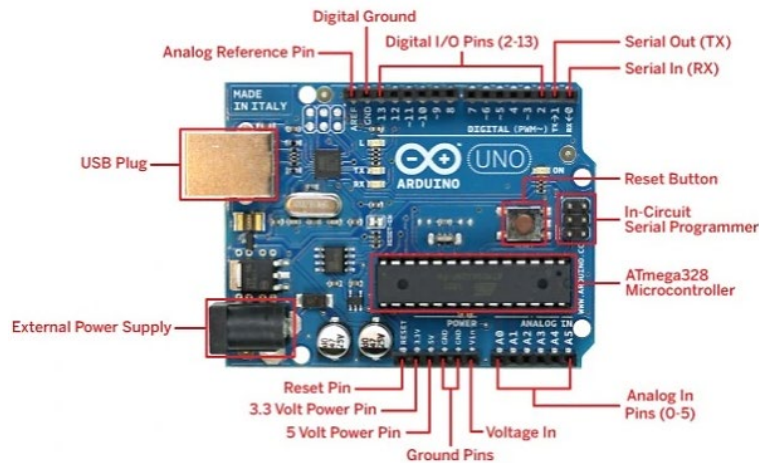
### **Selección de microcontrolador**

Existe una amplia variedad de microcontroladores empleados en el desarrollo de proyectos *IoT*, los cuales logran satisfacer los requerimientos del proyecto. Un análisis adecuado muestra que la mayoría microcontroladores comparten características tanto en rendimiento, como en operación. La selección del microcontrolador requirió una evaluación comparativa de las siguientes placas de desarrollo: *ESP32*, *NodeMCU ESP8266* y *Arduino UNO*.

El *ESP32* es una placa de desarrollo basada en un microcontrolador ampliamente utilizada y popular para aplicaciones *IoT*. Es fabricado por *Espressif Systems* y es conocido por sus potentes características y capacidades. Tiene importantes características para el desarrollo de varias aplicaciones, en las que destacan: conectividad *Wi-fi* y *Bluetooth*, 34 pines digitales, compatibilidad con varios códigos libres que permiten programar una amplia gama de tareas, entre los más importantes



Finalmente, el *Arduino Uno*, que se muestra en la Figura 3.3, es una placa de microcontrolador de código abierto basada en el microcontrolador *ATmega328P*. Es una de las placas *Arduino* más populares y se usa para varios proyectos, tiene 16 pines de entrada y salida digitales y 6 pines analógicos, inferior al número de pines del *ESP32*, no tiene un módulo de conexión inalámbrico, por lo que requiere implementar *hardware* adicional [15].



**Figura 3.3** *Arduino Uno* [15]

El microcontrolador seleccionado que cumple con los requerimientos del proyecto es el *ESP32*, el cual es ideal para establecer la comunicación *Wi-fi*, su gran cantidad de pines permite conectar los sensores para la detección de intrusos, conectar el teclado y la pantalla *LCD* para la operación manual de la alarma. Su memoria permite una ejecución correcta de la programación en cada acción y establecer la comunicación con las aplicaciones para el control de la alarma. En la Tabla 3.1 se observa la comparativa de las características de los 3 microcontroladores, mediante la cual se tomó la decisión para la elección del *ESP32* para el desarrollo del proyecto.

**Tabla 3.1** Comparación de microcontroladores [13][14][15]

Característica	ESP32	ESP8266	Arduino uno
<b>CPU</b>	Xtensa LX6 de doble núcleo	Xtensa LX8 de un solo núcleo	Atmega328P
<b>Velocidad de reloj</b>	Hasta 240 (MHz)	Hasta 160 (MHz)	16 (MHz)
<b>Memoria flash</b>	528 (KB), 1 (MB), 2 (MB), 4 (MB), 8 (MB), 16 (MB)	16 (MB)	32 (KB)
<b>Pines GPIO</b>	34	10	14
<b>Canales ADC</b>	dieciséis	1	6
<b>Canales DAC</b>	2	0	0

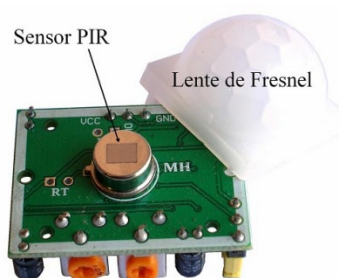


Característica	ESP32	ESP8266	Arduino uno
<b>Wi-fi</b>	802.11b/g/n/ac	802.11 b/g/n	N / A
<b>Bluetooth</b>	Bluetooth 4.2, Bluetooth de bajo consumo (BLE) 5.0	Bluetooth 2.1 + EDR	N / A
<b>Voltaje de funcionamiento</b>	3.3 (V) a 5 (V)	3 a 3.6 (V)	5 (V)
<b>Precio</b>	\$10-20	\$5-10	\$20-30

### Selección del sensor de movimiento

Para la detección del movimiento se tomó la decisión de utilizar el *PIR* HC-SR501, que se muestra en la Figura 3.4, el cual mediante la utilización de un lente *fresnel* permite enfocar las señales infrarrojas que son emitidas por la temperatura de los seres vivos. Para su operación requiere de la alimentación con un voltaje de 4.2 (V) a 5 (V) y consumo de corriente máximo de 65 (mA) [9].

Su funcionamiento simple permite detectar el movimiento y colocarlo en estado alto, al no existir presencia pasa a estado bajo. Mediante dos potenciómetros permite ajustar el sensor, el potenciómetro de la parte izquierda permite ajustar el tiempo de detección que tendrá el *PIR*, por otro lado, el potenciómetro de la parte derecha permite calibrar la distancia de detección [9]. El sensor descrito se adapta a los requerimientos del proyecto y permitirá la detección de personas no autorizadas.



**Figura 3.4** *PIR* HC-SR501 [9]

### Selección del sensor magnético

Para la detección de apertura de puertas o ventanas se optó por seleccionar el sensor de contacto magnético el mismo que cuenta con dos partes, un sensor y un imán, como se visualiza en la Figura 3.5. Las dos piezas permiten actuar al sensor como un interruptor con estado de encendido o estado apagado. El momento que las dos piezas se separen se colocara en estado apagado, la acción contraria lo coloca en estado encendido. La distancia de operación del sensor se encuentra en el rango de 15 a 25 (mm) de separación del sensor con el imán, para su operación se requiere de 3 (V) a 24 (V) y

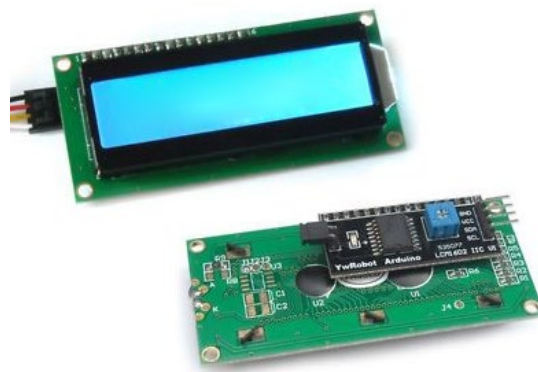
una corriente máxima de 100 (mA) [10]. Este sensor se adapta a las necesidades del presente proyecto y se complementa de manera correcta con el *PIR*.



**Figura 3.5** Sensor magnético [10]

### **Selección de la pantalla**

Para permitir observar la digitación de la clave, al momento de activar o desactivar la alarma, se eligió la pantalla *LCD I2C*. El *I2C* es una interfaz serial que facilita el control del *LCD*, sin la necesidad de utilizar varios cables. El *LCD I2C*, que se puede observar en la Figura 3.6, cuenta con una resolución 16x4 pixeles y permite mediante un potenciómetro ajustar el contraste para la visualización del texto de mejor manera. El panel *LCD* utiliza tecnología de cristal líquido para producir caracteres y símbolos con alto contraste y claridad. Opera con un voltaje de 5 (V) y corriente de 2 (mA) [16]. Mediante el *LCD* se visualizará las acciones que realice la alarma, en especial observar el ingreso de la clave como se especifica en los requerimientos del proyecto.



**Figura 3.6** *LCD I2C* [16]

### **Selección del teclado**

El teclado matricial o *keypad*, seleccionado para la manipulación de la alarma, es un *keypad* 4x4 como se muestra en la Figura 3.7. Este teclado es compatible con el microcontrolador *ESP32*, su funcionamiento requiere de un voltaje de 3.3 (V) a 5 (V) y corriente de 30 (mA), cuenta con 16 botones entre números y letras. Su forma compacta,

interfaz amigable y fácil de usar, permite su utilización en varios proyectos electrónicos [18]. Mediante el *keypad* se cumplirá con el ingreso de la clave para la activación, desactivación y cambio de clave de la alarma como se expuso en los requerimientos del proyecto.



**Figura 3.7 Keypad 4x4 [18]**

### **Selección del Software**

Para determinar el software con el cual se creará el código, que permitirá que los sensores elegidos se ejecuten de manera correcta, mediante el microcontrolador *ESP32*, se realizó una evaluación de las características de *Arduino IDE* y del *Web Editor* de *Arduino IoT Cloud*.

El *Web Editor*, que viene integrado en la plataforma de *Arduino IoT Cloud*, es una de las opciones más utilizadas para desarrollar proyectos en las placas de *Arduino*. En la Figura 3.8 se visualiza la interfaz de *Web Editor*. Este software no requiere de instalación de algún elemento en el ordenador, gracias a que *Arduino IoT Cloud* cuenta con su propia nube de almacenamiento. *Web Editor* permite que las líneas de código queden almacenadas en su plataforma y facilita la implementación de cualquier tipo de biblioteca, siempre que esta esté disponible y sea compatible. [11].

Cuando se trabaja con *Arduino IoT Cloud*, *Web Editor* integra la librería *thingProperties.h* que trae las variables enlazadas con la plataforma de *Arduino IoT Cloud*, así como las credenciales de las placas de desarrollo y de las redes *Wi-fi*, facilitando de esta manera crear proyectos *IoT*. Una gran ventaja de *Web Editor* es que permite acceder a cualquier proyecto creado sin tener que cerrar el anterior. También, los códigos se pueden compilar, depurar y subir a la placa de desarrollo desde el mismo *Web Editor* mediante una conexión USB o de manera *Online* [11].

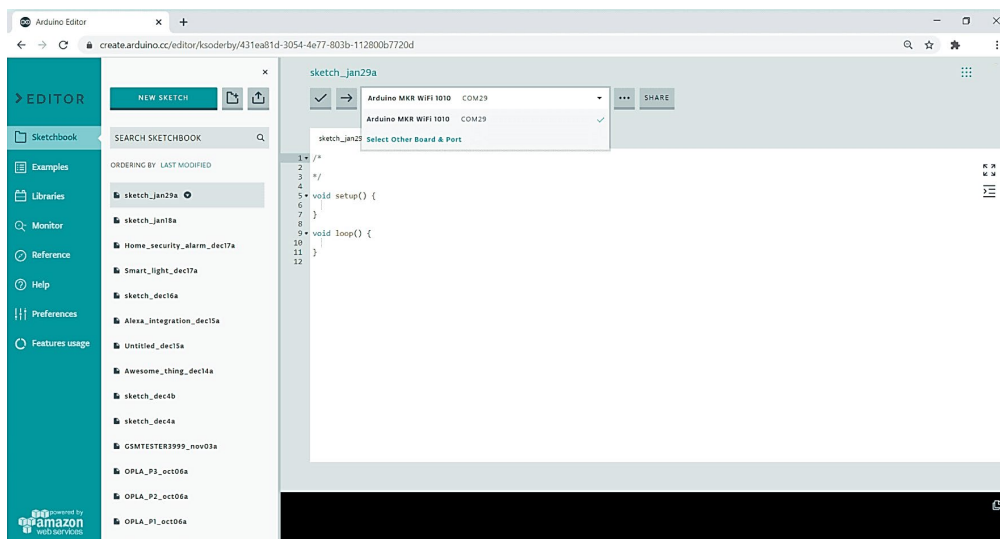


Figura 3.8 Web Editor [11]

Por otro lado, el entorno de desarrollo integrado (*IDE*) de *Arduino* es un *software* que cuenta con varias herramientas para programar en distintos lenguajes de programación compatibles con placas de desarrollo. Su interfaz permite realizar la compilación de las líneas de código creadas, al igual que depurar y subir a la placa de desarrollo mediante conexión USB, como muestra la Figura 3.9. Para poder utilizar *Arduino IDE* es necesario descargar e instalar el *software* desde la *web* oficial de *Arduino*, lo que es una desventaja en comparación a *Web Editor*. También, para su funcionamiento es necesario contar con el modelo exacto de la placa de desarrollo, lo que puede convertirse en un problema al utilizar placas genéricas [20].

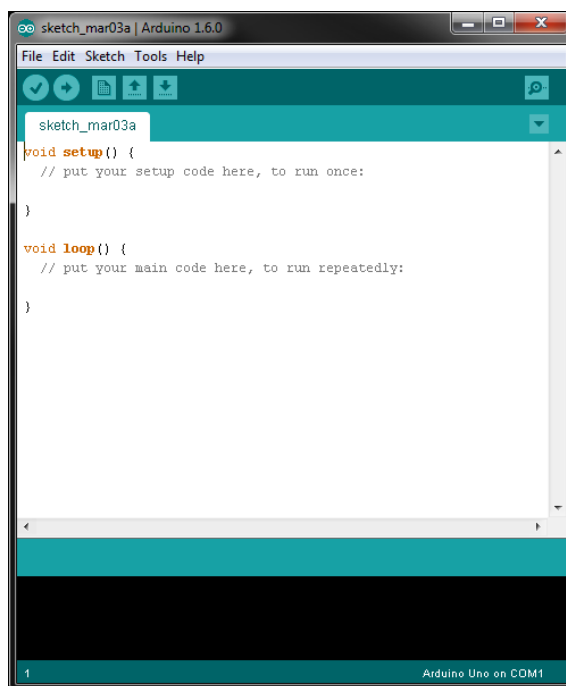


Figura 3.9 Arduino IDE [20]

El proyecto será desarrollado mediante el *software Web Editor* de *Arduino IoT Cloud*, ya que al analizar las características más importantes es la que permite desarrollar de mejor manera las líneas de código, para enlazar todos los elementos físicos que componen el proyecto, finalmente, *Web Editor* cumple con lo necesario para desarrollar los códigos que enlacen los componentes físicos con el *software* a utilizar, como se establece en los requerimientos del proyecto.

### 3.3 Diseño del prototipo de la alarma

#### Selección de plataforma

La elección de la plataforma para el desarrollo de la aplicación se realizó mediante una investigación de las características más importantes que se adapten a los requerimientos del proyecto.

#### Selección de la plataforma para la aplicación

Para la implementación de la aplicación que permita controlar y monitorear la alarma mediante *Internet* se evaluaron dos de las plataformas más comerciales: *Arduino IoT Cloud* y *Blynk*.

*Arduino IoT Cloud* es una plataforma que permite implementar proyectos *IoT* gracias a su compatibilidad con placas de desarrollo, como *ESP32*. Su interfaz amigable, como muestra la Figura 3.10, permite a los usuarios desarrollar los proyectos de manera fácil e intuitiva.

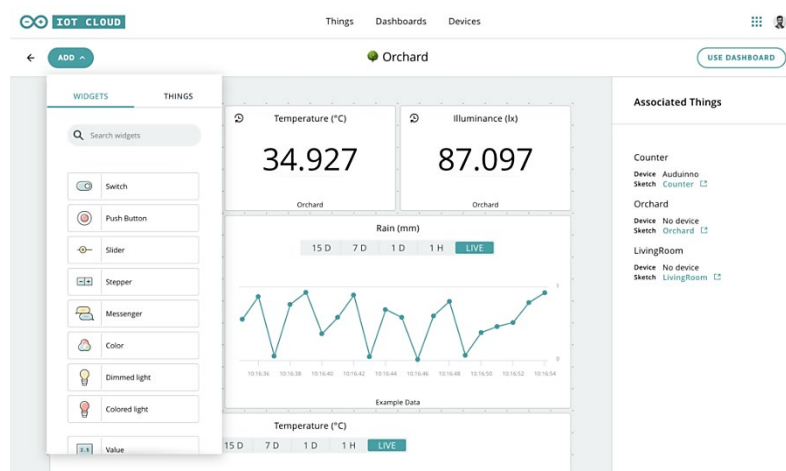


Figura 3.10 *Arduino IoT Cloud* [11]

Los principales servicios a los que se puede acceder en *Arduino IoT Cloud* son monitoreo de sensores en tiempo real, muy útil en sistemas de seguridad. La sincronización de las variables permite establecer la comunicación de manera correcta entre la plataforma y las placas de desarrollo, los tableros de *Arduino IoT Cloud* ofrecen

versatilidad acorde a los requerimientos del usuario. La cantidad de *widgets* con los que cuenta, en su versión gratuita, permite que la interacción a la hora de ejecutar algún proyecto sea de manera sencilla [11].

También, *Arduino IoT Cloud* cuenta en su plataforma su propio editor *web* en donde se puede llevar a cabo la programación mediante código abierto, además la integración con las bibliotecas para cada dispositivo se simplifica gracias a su editor. Todo tablero que se crea en su plataforma se verá reflejada en su aplicación disponible en la *Play store* de dispositivos *Android*, lo que permite usar sus servicios desde un terminal móvil [11].

Por otro lado, *Blynk* es una plataforma *IoT* que permite crear proyectos en distintas placas de desarrollo. Su interfaz amigable, como se observa en la Figura 3.11, permite con el simple hecho de seleccionar un *widget* y enlazarlo a una variable, en la placa de desarrollo, empezar a ejecutar acciones. A diferencia de *Arduino IoT Cloud*, *Blynk* no cuenta con su editor para programar, es necesario que la programación se desarrolle en el *software* adecuado a la placa de desarrollo. Su plataforma *web* no se refleja en su aplicación móvil, por lo que hay que volver a crear la interfaz. Muchos de sus *widgets* solo están disponibles en su versión *premium* lo que limita en grandes proyectos [17].



Figura 3.11 Blynk [17]

A partir de la información expuesta se ha optado por seleccionar a *Arduino IoT Cloud* como la plataforma para desarrollar la aplicación para el control y monitoreo de la alarma. La amplia información existente permite ser más llevadero el desarrollo del proyecto, a diferencia de *Blynk*, *Arduino IoT Cloud*, sí permite que su aplicación móvil cuente con la misma interfaz existente en la plataforma *web*. Todas las características expuestas permiten cumplir los requerimientos del proyecto.

### Selección de la plataforma para el *bot* de *Telegram*

Como parte de los objetivos del proyecto la alarma debe ser controlada a través de *Telegram*. *Telegram* es una de las aplicaciones de mensajería más utilizadas a nivel mundial, su amplia gama de servicios la convierte en una plataforma adecuada para controlar la alarma de manera remota a través de *Internet*. Entre sus muchos servicios destaca la creación de *bots*. Un *bot* es una aplicación que funciona internamente en *Telegram* y no requiere de algún tipo de instalación adicional [12][19].

El *bot* de *Telegram* permite mediante instrucciones en base a comandos ejecutar acciones, ya sea de manera interna en *Telegram* o mediante otro dispositivo, como por ejemplo el control de una alarma. Los *bots* permiten acceder a varios servicios como: recibir notificaciones de otros dispositivos, integrarse con otras plataformas, permitir el control de aplicaciones o dispositivos. Además de existir *bots* ya creados que permiten acceder a servicios de música, películas, libros, entre otros más, *Telegram* permite crear un *bot* de manera personalizada acorde a los requerimientos del usuario [19].

Para crear el *bot* se compararon tres opciones. *BotFather*, que se observa en la Figura 3.12, es un *bot* integrado en *Telegram* y no requiere de conocimiento en programación para automatizar instrucciones, su amplia documentación permite acceder a guías para desarrollar *bots* que permitan controlar aplicaciones de manera sencilla con instrucciones simples [21].



**Figura 3.12** *BotFather* [21]

Por otro lado, *Manybot* es otro *bot* integrado en *Telegram* para poder crear *bots* que se integran con distintos proyectos. Finalmente, *AradBot* es una opción externa a *Telegram* que permite desarrollar *bots*, su limitante es que su desarrollo se hace de manera externa [22]. Si bien *Manybot* y *AradBot* son buenas opciones para crear *bots*, la falta de información de guías para crear proyectos en *ESP32* limita su utilización en el



desarrollo del presente proyecto. Por tal razón, se ha seleccionado a *BotFather* para desarrollar el *bot* en *Telegram* que permitirá el control y monitoreo de la alarma.

### Esquema general del proyecto

El esquema general del funcionamiento del prototipo de alarma, que se visualiza en la Figura 3.13, presenta de forma general la operación de la alarma. El prototipo de alarma se desarrolló sobre el microcontrolador *ESP32* que actuará como el elemento central de todo el sistema de alarma. La placa mantendrá conexión vía *Wi-fi* para establecer comunicación por *Internet* con las plataformas de *Arduino IoT Cloud* y *Telegram*, las cuales permitirán el monitoreo y control de la alarma en tiempo real, siempre y cuando exista conexión a *Internet*.

Finalmente, el microcontrolador estará montado sobre una caja para su protección, en donde se establecerá las respectivas conexiones con el sensor de movimiento, los dos sensores magnéticos, el teclado matricial 4x4, el *LCD I2C 16x2*, el *buzzer*, los dos *LED* y con la fuente de alimentación. La notificación de detección de algún intruso por parte de los tres sensores será recibida por el *bot* de *Telegram*, el cual recibirá que sensor fue el afectado, además la alerta será emitida mediante un sonido por el *buzzer*.

El usuario podrá realizar algún tipo de acción como encendido o apagado, revisión de sensores desde *Arduino IoT Cloud* y *Telegram*. Mediante el teclado matricial junto con la pantalla *LCD* podrá activar, desactivar y cambiar la clave de la alarma de manera manual. De esta manera se garantiza la operación del prototipo de alarma con y sin conexión a *Internet*.

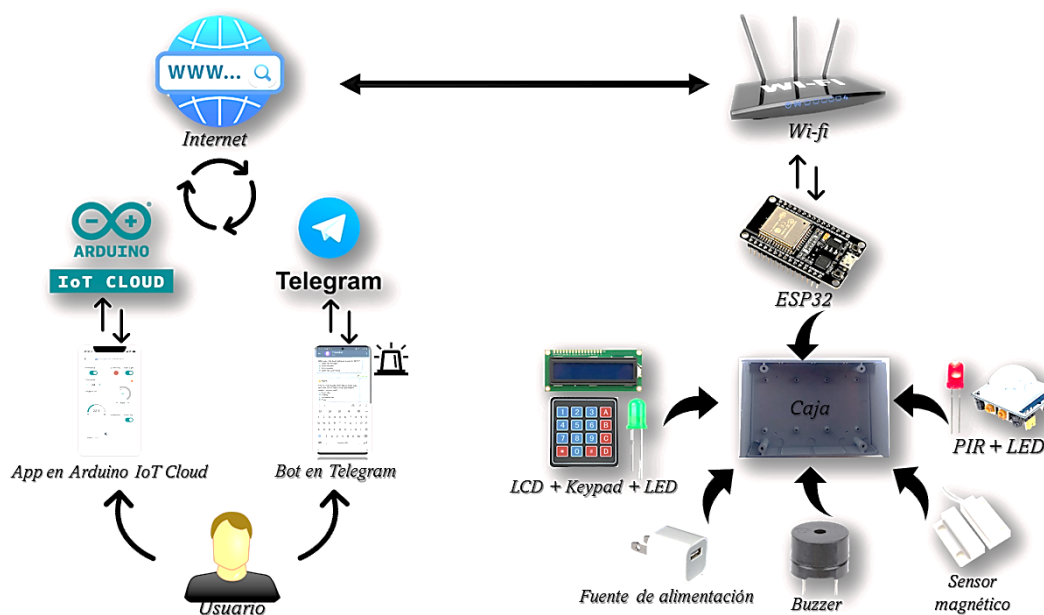
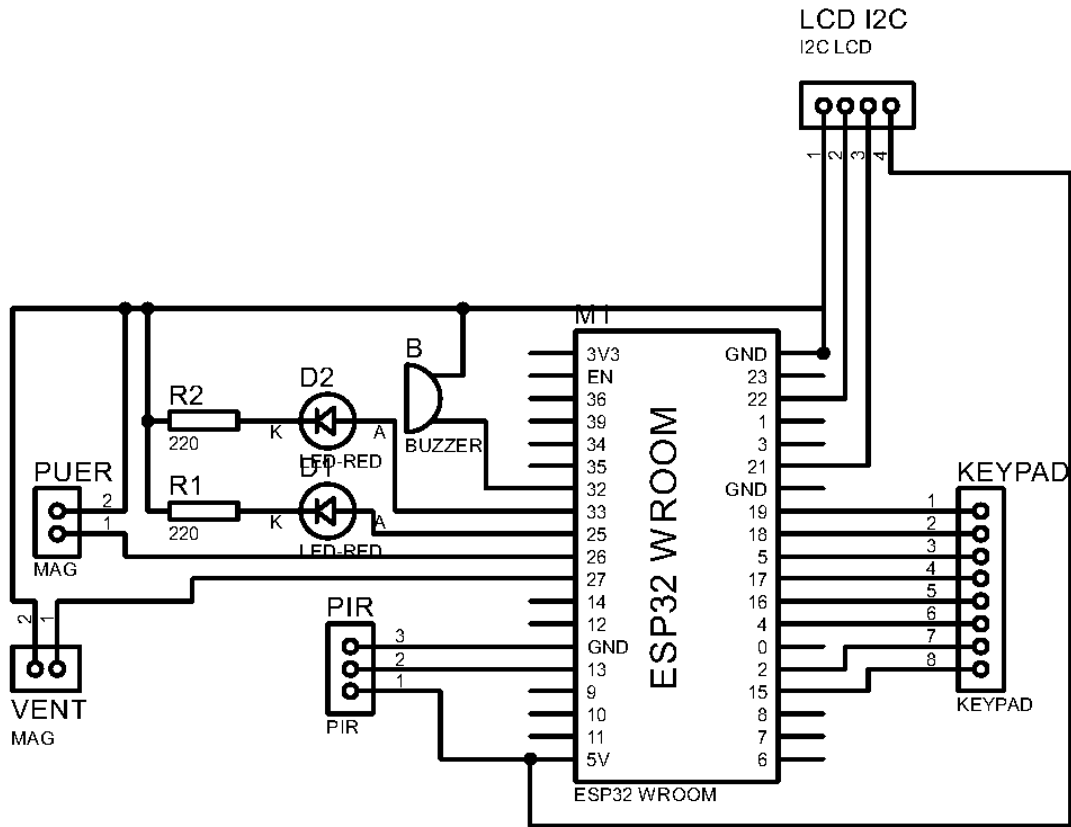


Figura 3.13 Esquema general del funcionamiento del prototipo de alarma



## Desarrollo del código en ESP32

El desarrollo de las líneas de código se realizó considerando el circuito eléctrico creado en *Proteus*, que se muestra en la Figura 3.14, en donde se visualiza las conexiones de cada elemento en la placa *ESP32*.



**Figura 3.14** Circuito del prototipo de alarma

Como se muestra en el circuito el *PIR* cuenta con 3 pines, un pin necesario para la alimentación a 5 (V), un segundo pin con conexión a *ground* (*GND*) y un pin para la entrada a la placa *ESP32*. Los dos sensores magnéticos para implementar en una ventana y puerta requieren una conexión a un pin de entrada en la placa *ESP32* y un pin *GND*. El módulo *LCD I2C* cuenta con cuatro pines, un pin con conexión a *GND* y otro para la alimentación a 5 (V), el *LCD* cuenta con dos pines específicos *Serial Clock* (*SCL*) y *Serial Data* (*SDA*), los cuales deben ser conectados en sus pines respectivos en la placa *ESP32*, en este caso pin 21 y 22 respectivamente.

El teclado matricial 4x4 cuenta con 8 pines para su funcionamiento, al ser de entrada se utilizó los pines respectivos. El *buzzer* requiere de dos pines, uno para salida y el *GND*, finalmente los dos *LED* que servirán para mostrar si la alarma está encendida o apagada, *LED* verde, y para mostrar la actividad del *PIR*, *LED* rojo, requieren de un pin

para salida en el *ESP32* y uno para *GND*. Cada *LED* cuenta con una resistencia para su protección. Para determinar el valor de las resistencias se requirió de los valores de alimentación y corriente para el funcionamiento de cada *LED*.

El *LED* rojo requiere de un voltaje de 2 (V) y corriente de 20 (mA), mediante la ley de ohm se puede obtener el valor de la resistencia [24], conociendo que cada pin del *ESP32* proporciona 3.3 (V). Mediante la Ecuación 3.1 se calculó la resistencia del *LED* rojo.

$$R = \frac{V_{cc}}{I}$$

**Ecuación 3.1** Ley de Ohm [23]

donde:

$V_{cc}$ : (V) Voltaje

R: ( $\Omega$ ) Resistencia

I: (A) Corriente

Por lo tanto:

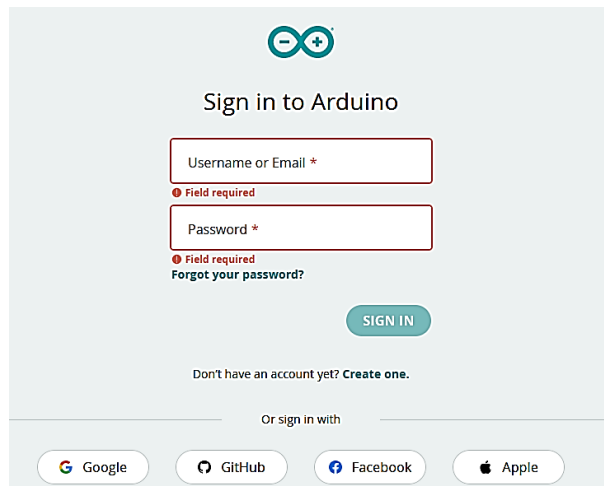
$$R_{\text{rojo}} = \frac{3.3 \text{ (V)} - 2 \text{ (V)}}{20 \text{ (mA)}} = 65 \text{ } (\Omega)$$

Para el *LED* verde se requiere de 2.4 (V) y corriente de 20 (mA) [24], con lo cual mediante la ley de Ohm se obtuvo:

$$R_{\text{verde}} = \frac{3.3 \text{ (V)} - 2.4 \text{ (V)}}{20 \text{ (mA)}} = 45 \text{ } (\Omega)$$

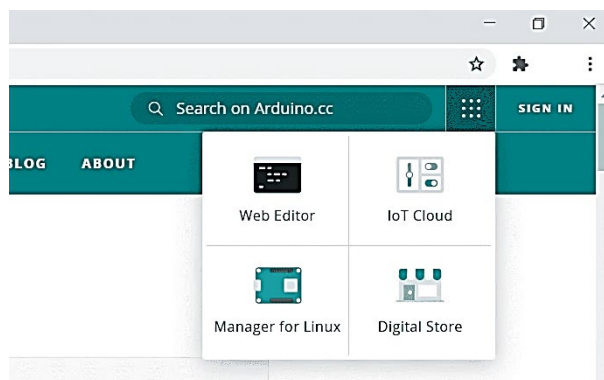
Los valores obtenidos son teóricos, y deben adaptarse a los existentes en el mercado.

Con la información anterior se realizó la creación de los códigos en *Web Editor*, dado que la plataforma para la aplicación es *Arduino IoT Cloud*. Para poder utilizar *Web Editor* se debe crear una cuenta en la página de *Arduino Cloud*. En la Figura 3.15 se muestra la pantalla de inicio, para lo cual se debe contar con un correo, o a su vez se puede iniciar con una cuenta de *Facebook* o *Google*.



**Figura 3.15** Cuenta *Arduino IoT Cloud* [11]

Una vez dentro de la plataforma, se debe seleccionar el apartado *Web Editor*, como se muestra en la Figura 3.16.



**Figura 3.16** *Web Editor* [11]

Como se mencionó en un punto anterior, una de las facilidades que tiene *Web Editor* es que cuenta con la librería *thingProperties.h* donde están las variables que se utilizan en la aplicación creada en *Arduino IoT Cloud*. Para este proyecto las variables enlazadas con los *widgets* son las que se muestran en la Figura 3.17.

```
int mov;
bool onoff;
bool pir;
bool puerta;
bool ventana;
```

**Figura 3.17** Variables *Arduino IoT Cloud*

La variable de tipo entero **mov**, se encarga de recibir un 1 si la alarma detecta algún intruso o un 0 si la alarma se mantiene sin novedad, siempre y cuando la alarma este activada, para lo cual se utiliza la variable de tipo booleana **onoff** enlazada con un *switch* virtual de la aplicación en *Arduino IoT Cloud*. Para los estados de los sensores se

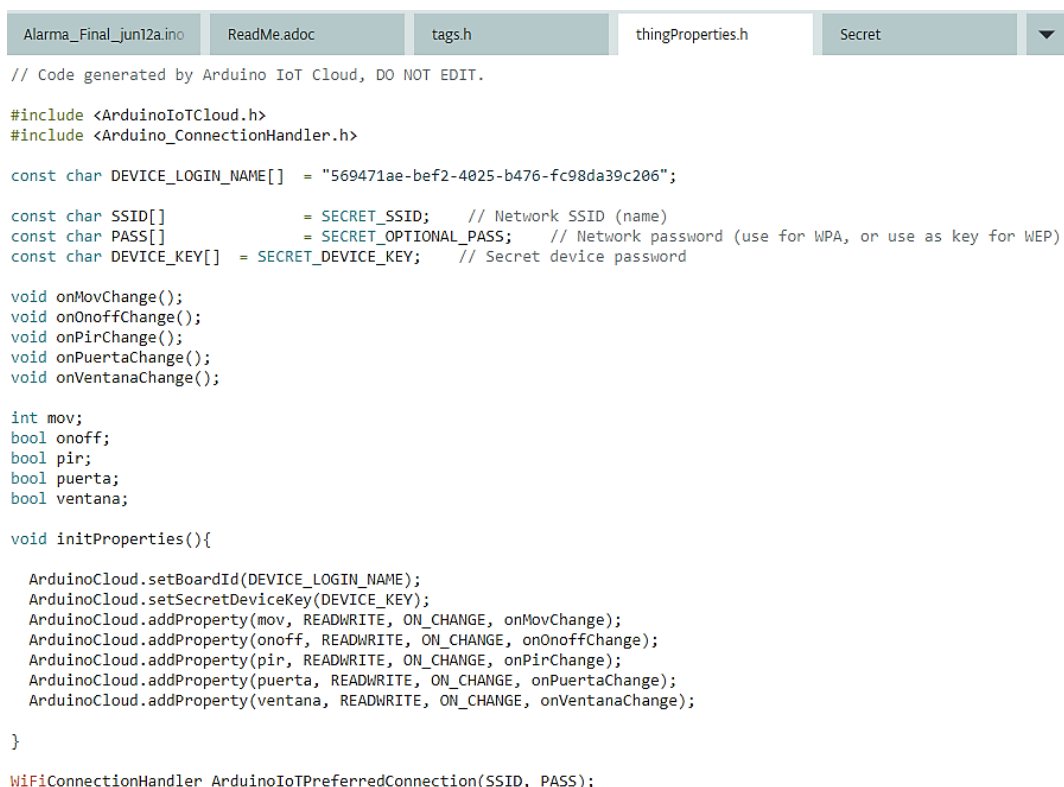
utilizaron las variables booleanas **pir** para el sensor de movimiento, **puerta** para el sensor magnético de la puerta y **ventana** para el sensor magnético de la ventana.

Las librerías utilizadas para el desarrollo del código se las muestra en la Figura 3.18, cada una con su respectiva descripción acorde a su funcionamiento, en donde se observa la librería *thingProperties.h* la cual como se mencionó cuenta en su interior las variables y credenciales para la conectividad con la plataforma *Arduino IoT Cloud*. También, se debe destacar la librería *UniversalTelegramBot.h* la cual permite establecer la comunicación con la plataforma de *Telegram* y con el respectivo *bot* creado.

```
#include <Arduino_ConnectionHandler.h>
#include <ArduinoIoTCloud.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h> //Libreria para Telegram
#include "thingProperties.h" //Libreria con las variables de arduino Iot Cloud
#include <Wire.h> //Libreria para el LCD
#include <LiquidCrystal_I2C.h> //Libreria par el LCD
#include <EEPROM.h> //Libreria para guardar en la memoria EEPROM
```

**Figura 3.18** Librerías

Como se ha mencionado en puntos anteriores, la librería *thingProperties.h* es muy importante para el desarrollo y ejecución del presente proyecto, en la Figura 3.19 se muestra las variables y credenciales que permiten la comunicación con *Arduino IoT Cloud* y que están dentro de la librería mencionada.



```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[] = "569471ae-bef2-4025-b476-fc98da39c206";

const char SSID[] = SECRET_SSID; // Network SSID (name)
const char PASS[] = SECRET_OPTIONAL_PASS; // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[] = SECRET_DEVICE_KEY; // Secret device password

void onMovChange();
void onOnoffChange();
void onPirChange();
void onPuertaChange();
void onVentanaChange();

int mov;
bool onoff;
bool pir;
bool puerta;
bool ventana;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(mov, READWRITE, ON_CHANGE, onMovChange);
  ArduinoCloud.addProperty(onoff, READWRITE, ON_CHANGE, onOnoffChange);
  ArduinoCloud.addProperty(pir, READWRITE, ON_CHANGE, onPirChange);
  ArduinoCloud.addProperty(puerta, READWRITE, ON_CHANGE, onPuertaChange);
  ArduinoCloud.addProperty(ventana, READWRITE, ON_CHANGE, onVentanaChange);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

**Figura 3.19** *thingProperties.h*

Las variables globales que se utilizan en la ejecución de la alarma se muestran en la Figura 3.20. Cada variable cuenta con su respectiva descripción acorde a su utilización, se puede observar en especial la variable **BOT\_TOKEN** para almacenar el *token* otorgado por el *bot* de *Telegram*. También, la variable **ID\_Chat** que cuenta con el ID personal del usuario que utilizara el *bot*.

```
#define BOT_TOKEN "6328090633:AAF1jX_nDHcuZ9hNhtPmDT17kFzb-URMshQ" //Variable que almacena el token del bot de Telgram
#define ID_Chat "1285841367" //Variable que almacena el Id personal de Telegram
const unsigned long tiempo = 1000;
WiFiClientSecure secured_client;
UniversalTelegramBot bot(BOT_TOKEN, secured_client);

unsigned long tiempoAnterior;
int estadoLed12 = 0;
String chat_id; //Variable para almacenar el ID del usuario que envia instrucciones mediante el bot de Telegram

bool alarmaActivada = false; //Variable para almacenar el estado de la alarma activado o desactivado
bool mensajeEnviado = false; //Variable para almacenar el estado de los mensajes enviados a telegram
bool cambioDesdeTelegram = false; //Variable para almacenar el estado activado o desactivado enviado del mensaje a telegram
int inicio = 1;
int ledPin = 25; //Variable para el led del sensor PIR, pin 25
int ledOn = 33; //Variable para el led de encendido o apagado, pin 33
int ven = 26; //Variable para el sensor magnetico de la ventana, pin 26
int pue = 27; //Variable para el sensor magnetico de la Puerta, pin 27
#define MOTION_SENSOR_PIN 13 // Variable para el funcionamiento del sensor PIR, pin 13
#define BUZZER_PIN 32 // Variable para el funcionamiento del buzzer, pin 32
int motionStateCurrent = LOW; // Variable para almacenar el estado actual del PIR
int motionStatePrevious = LOW; // Variable para almacenar el estado previo del PIR
String ac; //Variable para almacenar el estado en texto del sensor PIR
String pu; //Variable para almacenar el estado en texto del sensor magnetico de la puerta
String vn; //Variable para almacenar el estado en texto del sensor magnetico de la ventana
bool vtana; //Variable para almacenar el estado actual del sensor PIR
bool prta; //Variable para almacenar el estado actual del sensor magnetico
bool sla; //Variable para almacenar el estado actual del sensor magnetico
bool cloudprevio = false; //Variable para almacenar el estado previo del switch en Arduino IoT Cloud
bool cloudactual; //Variable para almacenar el estado actual del switch en Arduino IoT Cloud
//----Variables para el encendido del LCD
bool encender = false;
unsigned long t = 0;
int t_ms_encendido = 7000;
//-----
bool conexionprevia = false; //Variable para almacenar el estado previo de la conexion con Arduino IoT Cloud
bool conexionactual; //Variable para almacenar el estado actual de la conexion con Arduino IoT Cloud

#define frecuencia_inicio 220 // define la frecuencia inicial del sonido
#define frecuencia_final 880 // define la frecuencia final del sonido
#define tiempo_ejecute 30 // define el tiempo en segundos para que el código se ejecute
bool z;
char clave_anterior[7]; //Variable para almacenar una clave para el keypad por defetco para la primera utilizacion
char nueva_clave[7]; //Variable para almacenar la clave de activacion y desactivacion de la alarma
char ingresaPass[7]; //variable para almacenar la clave ingresada por el usuario mediante le Keypad
int indice = 0;
int direccionIndicador = 100; // Dirección en la EEPROM para almacenar el indicador de primera ejecución
int direccionAlarmaActivada = 200; // Dirección en la EEPROM para almacenar el estado de alarmaActivada
```

**Figura 3.20** Variables

Se debe mencionar que debido a una incompatibilidad con la librería *keypad.h*, la y la librería de *Arduino IoT Cloud*, el funcionamiento del *keypad* se realizó mediante la detección de pulsación, para lo cual se utilizaron las variables que se muestran en la Figura 3.21.

```
LiquidCrystal_I2C lcd(0x27, 16, 2);

const uint8_t ROWS = 4;
const uint8_t COLS = 4;

char keys[ROWS][COLS] = {
  { '1', '2', '3', 'A' },
  { '4', '5', '6', 'B' },
  { '7', '8', '9', 'C' },
  { '*', '0', '#', 'D' }
};

uint8_t colPins[COLS] = { 16, 4, 2, 15 };
uint8_t rowPins[ROWS] = { 19, 18, 5, 17 };
```

**Figura 3.21** Variables de funcionamiento del *keypad*

Una vez declaradas las librerías y variables necesarias para el funcionamiento del prototipo, se inicia con la creación de las líneas de código para el funcionamiento de la alarma, iniciando con la función principal **void setup()**, que se puede observar en la Figura 3.22.

La función **void setup()** se inicia una sola vez por cada encendido que tenga el sistema en general, por tal motivo es fundamental ya que en esta función se inician las variables de los pines a ser utilizados por cada sensor y demás elementos como el *keypad* y *LCD*. También, se inicia con la lectura de los valores almacenados en la memoria *EEPROM* del *ESP32*, las cuales son: **alarmaActivada**, variable para obtener el último estado activado o desactivado de la alarma; **nueva\_clave**, variable para obtener el valor de la última clave con la se activa o desactiva la alarma de manera manual mediante el *keypad*. Se puede observar el llamado de la función **connectToWifi()**, función que permite establecer la conexión a *Wi-fi*, la comunicación con el *bot* de *Telegram* y con la plataforma *Arduino IoT Cloud*, como se muestra en la Figura 3.23.

```
void setup() {
  Serial.begin(115200);
  lcd.init();
  lcd.print("...INICIADO...");
  EncenderLCD();
  EEPROM.begin(512);
  bool primeraEjecucion = EEPROM.read(direccionIndicador) != 'X';
  if (primeraEjecucion) {
    // Establecer valores predeterminados
    strcpy(clave_anterior, "ABCD45");
    strcpy(nueva_clave, "ABCD45");
    alarmaActivada = false; // Valor predeterminado cuando no se ha guardado en la EEPROM
    // Guardar valores predeterminados en la EEPROM
    for (int i = 0; i < 7; i++) {
      EEPROM.write(i, clave_anterior[i]);
      EEPROM.write(i + 7, nueva_clave[i]);
    }
    // Guardar indicador de primera ejecución en la EEPROM
    EEPROM.write(direccionIndicador, 'X');
    EEPROM.commit();
  } else {
    // Recuperar valores de la EEPROM, la clave
    for (int i = 0; i < 7; i++) {
      clave_anterior[i] = EEPROM.read(i);
      nueva_clave[i] = EEPROM.read(i + 7);
    }
    alarmaActivada = EEPROM.read(direccionAlarmaActivada) == '1'; // Cargar estado de alarmaActivada desde la EEPROM
  }
  EEPROM.end();
  //-----Variables para los sensores, leds y buzzer -----
  pinMode(pue, INPUT_PULLUP);
  pinMode(ven, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  pinMode(ledOn, OUTPUT);
  pinMode(MOTION_SENSOR_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  connectToWifi();
}
```

**Figura 3.22** Función *void setup*

```

//----- Función para iniciar la conexión a la red y plataformas
void connectToWiFi() {

    Serial.print("Conectando a la red ");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD); // Conexión a wifi
    secured_client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
    Serial.print("\nConectado a la red Wi-Fi. Dirección IP: ");
    initProperties(); // Inicia las propiedades guardadas en thingProperties.h de arduino cloud

    ArduinoCloud.begin(ArduinoIoTPreferredConnection); // Inicia la conexión entre arduino iot cloud y el esp32

    setDebugMessageLevel(2);
    ArduinoCloud.printDebugInfo();
}

```

**Figura 3.23** Función para conexión con las plataformas

Una vez establecido las variables y establecer la comunicación con los elementos a ser utilizados en la alarma, se procedió a crear la función **void loop()**, en esta función es donde propiamente se establecerá el programa de la alarma y el orden en que se ejecutaran las acciones. Todo lo generado dentro de esta función se ejecutará en un ciclo infinito [25].

En primer lugar, se establecerá la actualización de los parámetros y comunicación con la plataforma de *Arduino IoT Cloud*, mediante la función **ArduinoCloud.update()**. También, se verifica si existe comunicación con la plataforma mediante una comparación del estado previo de la conexión con el estado actual de la conexión, esto permite notificar al *bot* de *Telegram* del inicio de la alarma, siempre y cuando exista una primera conexión, significando esto que hay conexión a *Internet*, como se muestra en la Figura 3.24.

```

void loop()
{
    ArduinoCloud.update();
    // ----- Verifico si la conexión se perdió y se reestableció para notificar a telegram -
    conexionactual = ArduinoCloud.connected();
    if(conexionactual && !conexionprevia){
        bot.sendMessage(ID_Chat, "ALARMA INICIADA!!!, escribe Ayuda para ver las opciones", "");
        conexionprevia = true;
    }
    if(!conexionactual){
        conexionprevia=false;
    }
}

```

**Figura 3.24** Void loop inicio

En segundo lugar, se realizará la lectura del *keypad*, si existiese alguna tecla presionada. Debido a la incompatibilidad de la librería que se mencionó en puntos anteriores, se utiliza la función **readKeypad()** para leer las pulsaciones que existieran en el *keypad*, mediante un escaneo, como se observa en el código de la Figura 3.25.

```

//----Funcion para leer el teclado -----
char readKeypad() {
  // Realizar el escaneo del teclado matricial
  for (int col = 0; col < COLS; col++) {
    pinMode(colPins[col], OUTPUT);
    digitalWrite(colPins[col], LOW);

    for (int row = 0; row < ROWS; row++) {
      pinMode(rowPins[row], INPUT_PULLUP);
    }
    delayMicroseconds(10);
    for (int row = 0; row < ROWS; row++) {
      if (!digitalRead(rowPins[row])) {
        delay(50); // Debounce
        // Determinar el carácter correspondiente a la tecla presionada
        char key = keys[row][col];
        // Esperar a que se libere la tecla
        while (!digitalRead(rowPins[row])) {
          delay(10);
        }
        // Restaurar los pines del teclado a sus configuraciones originales
        for (int row = 0; row < ROWS; row++) {
          pinMode(rowPins[row], OUTPUT);
          digitalWrite(rowPins[row], LOW);
        }
        for (int col = 0; col < COLS; col++) {
          pinMode(colPins[col], INPUT_PULLUP);
        }
        return key;
      }
    }
  }
  // Restaurar los pines del teclado a sus configuraciones originales
  for (int row = 0; row < ROWS; row++) {
    pinMode(rowPins[row], OUTPUT);
    digitalWrite(rowPins[row], LOW);
  }
  pinMode(colPins[col], INPUT_PULLUP);
}
return '\0'; // Si no se presionó ninguna tecla, devolver '\0'
}

```

**Figura 3.25** Función *readKeypad*

La lectura del *keypad* se almacena en la variable tipo *char* llamada **key**. Si durante la ejecución del ciclo se presionó alguna tecla se realiza una evaluación de la variable *key* y se ejecuta la acción acorde a la tecla que se ha presionado, como muestra la Figura 3.26. Si la tecla presionada corresponde al carácter numeral (#), el sistema solicita el ingreso de la clave, esto con la finalidad de ejecutar la acción de encendido o apagado de la alarma. Por defecto se ha limitado la longitud de la clave a 6 caracteres entre números o letras. Una vez el usuario finaliza el ingreso de la clave se ejecuta de manera inmediata la evaluación de la clave ingresada y almacenada en la variable **ingresePass**.



```

//----- Comprueba si esta precionado una tecla en el keypad -----
char key = readKeypad();
if (key == '#') { // Si preciono # Solicito que ingrese la clave para poder Activar o Desativar ----
  indice = 0;
  EncenderLCD();
  lcd.clear();
  lcd.print("CLAVE POR FAVOR");
  while (true) {
    char key = readKeypad();
    if (key) {
      lcd.setCursor(indice, 1);
      lcd.print('*');
      ingresaPass[indice] = key;
      indice++;
    }
    if (indice >= 6) {
      ingresaPass[indice] = '\0';
      break;
    }
  }
} else if (key == '*') { // Si la tecla es * Se ingresa para el cambio de clave ----
  EncenderLCD();
  indice = 0;
  cambiar_clave();
} else if (key == 'D' || key == 'd'){ // Si la tecla es D se ilumina el LCD para ver el mensaje ----
  EncenderLCD();
}
}

```

**Figura 3.26** Acciones del Sistema con *Keypad*

La evaluación de la clave se realiza siempre y cuando se ha presionado 6 valores como clave. Mediante un *if* se evalúa si el contador llamado **índice** es igual o mayor a 6, si esto es correcto se procede a comparar la clave ingresada con la clave que se encuentra guardada en la memoria del *ESP32*, que se obtuvo al inicio del sistema. Si al comparar las variables **IngresePass** con **nueva\_clave** y el resultado es 0, significa que son iguales, se procede a imprimir el mensaje “CLAVE CORRECTA” en el *LCD*.

Acto seguido se ejecuta un nuevo *if* en el que se evalúa el estado último que ha tenido la alarma y que se encuentra almacenado en la variable **alarmaActivada**. Si el valor es verdadero significa que la alarma está encendida, por tal razón se procede a apagar la alarma. También, se procede a asignar un valor falso a la variable **alarmaActivada**, se evalúa nuevamente el estado de conexión con *Arduino Cloud*, si existe conexión se envía el mensaje Alarma Apagada al *bot* de *Telegram* y se coloca al *switch* en la *app* de *Arduino IoT Cloud* en estado *LOW*, es decir apagado. A demás, se realiza las impresiones de mensajes respectivos en el *LCD*, esta misma acción ocurre si el valor de la variable **alarmaActivada** es falsa, lo que significa que está apagada, ejecutando acciones para el encendido, en este punto se coloca la variable **z** con estado *true*, ya que al encender la alarma existirá un tiempo de 30 segundos antes de que los sensores se activen y esto se notificará mediante la emisión de pitidos por parte del *buzzer*.

Por otro lado, si al evaluar la clave se obtiene un valor diferente de 0, se da por entendido que las claves no son iguales y se procede a imprimir los mensajes en el *LCD* de “CLAVE INCORRECTA”, “REINTENTE”. Finalmente se imprime el estado actual de la

alarma y se reinicia el contador **índice** a 0, todo este proceso se puede visualizar en la Figura 3.27.

```

if (índice >= 6) { // Si los valores ingresados son iguales o mayores a 6 se evalúa la clave
if (strcmp(nueva_clave, ingresaPass) == 0) { // Si es igual a la clave guardada se acepta
  lcd.clear();
  lcd.print("CLAVE CORRECTA");
  if (alarmaActivada) { //Si el estado actual de la alarma es verdadero, lo cambio a apagado
    alarmaActivada = false; //Coloca el valor de la variable en falso para una acción posterior
    if (Arduinocloud.connected()) {
      bot.sendMessage(ID_Chat, "Alarma APAGADA", ""); //Envía un mensaje al bot de telegram, si existe conexión con Arduino Cloud
      onoff = LOW; //Cambia el estado del switch a apagado, en la aplicación en Arduino IoT Cloud
    }
    cambioDesdeTelegram = true;
    estadoLed12 = 0; //Se asigna el valor de cero a la variable estadoLed12, esto para uso posterior para verificar el estado de la alarma
    lcd.clear(); //Se limpia la pantalla del LCD
    lcd.setCursor(0, 0); //Se coloca el cursor en la posición 0,0 en el LCD
    lcd.print("ALARMA APAGADA"); //Se imprime el mensaje de ALARMA APAGADA en el LCD
  } else { //Si el estado es apagado lo cambio a encendido
    alarmaActivada = true;
    z = true; //Coloco en verdadero la variable para evaluar el llamado a la función del buzzer para iniciar la alarma
    if (Arduinocloud.connected()) {
      onoff = HIGH;
      bot.sendMessage(ID_Chat, "Alarma ENCENDIDA", "");
    }
    cambioDesdeTelegram = true; //Valor utilizado para verificar que se notificó a telegram
    estadoLed12 = 1;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ALARMA ENCENDIDA");
  }
} else { //Si la clave ingresada no es igual a la almacenada
  lcd.clear();
  lcd.print("CLAVE INCORRECTA"); //Se imprime en el LCD el mensaje CLAVE INCORRECTA
  delay(500);
  lcd.clear();
  lcd.print("REINTENTE");
  if (alarmaActivada) { // Si el estado de la alarma es verdadero ejecuto acciones de alarma encendida
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ALARMA ENCENDIDA");
  } else { // Si es falso ejecuto acciones de alarma apagada
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ALARMA APAGADA");
  }
}
}
índice = 0; //Reinicia a 0 el valor de la variable utilizada como contador a la hora de ingresar la clave.
}

```

**Figura 3.27** Evaluación de la Clave

Continuando con las acciones relacionadas al *keypad*, si la tecla ingresada es asterisco (\*), significa que el usuario está solicitando el cambio de la clave, para lo cual se llama a la función **cambiar\_clave()**. En esta función se imprime de inicio un mensaje en el LCD de "INGR CLAVE ACTUAL", solicitando al usuario que ingrese la clave actual con la que cuenta. Nuevamente se utiliza la función **readKeypad()** para almacenar la clave en la variable *key*. Mediante un *While* se almacena los 6 caracteres en la variable **clave\_ingresada**. Acto seguido se compara la clave almacenada en **clave\_ingresada** con la clave almacenada en la memoria que se encuentra en la variable **clave\_anterior**, obtenida al cargar el sistema en su inicio.

Si el resultado de la comparación es un 0, significando ser iguales, se procede a solicitar al usuario una nueva clave, mediante un *While* se almacena los valores presionados del teclado, mediante la función **readKeypad()**, en la variable **nueva\_clave**. Seguidamente mediante la función *strcpy* se copia el contenido de la variable **nueva\_clave** en la variable **clave\_anterior**. Acto seguido se almacena en la memoria *EEPROM* las variables **nueva\_clave** y **clave\_anterior**, se envía un mensaje al *bot* de Telegram de

Se cambió la clave, Envía *Pass* para revisar. Todo el proceso anterior se visualiza en la Figura 3.28.

```

void cambiar_clave() {
  lcd.clear();
  lcd.backlight();
  lcd.print("INGR CLAVE ACTUAL");          //Despliego en el LCD el mensaje

  char clave_ingresada[7];
  int indice_clave = 0;

  while (true) {
    char key = readKeypad();

    if (key) {
      lcd.setCursor(indice_clave, 1);
      lcd.print('*');                      //Despliego un * en cada pulsacion
      clave_ingresada[indice_clave] = key; //Almaceno los digitos ingresados 6 veces
      indice_clave++;
    }

    if (indice_clave >= 6) {
      clave_ingresada[indice_clave] = '\0';
      break;
    }
  }

  if (strcmp(clave_ingresada, clave_anterior) == 0) { //Comparo la clave guardada con la clave ingresada
    lcd.clear();
    lcd.print("INGR NUEVA CLAVE");

    while (true) {
      char key = readKeypad();

      if (key) {
        lcd.setCursor(indice, 1);
        lcd.print(key);
        nueva_clave[indice] = key;
        indice++;
      }

      if (indice >= 6) {
        nueva_clave[indice] = '\0';
        break;
      }
    }

    strcpy(clave_anterior, nueva_clave); //Copeo la nueva clave en la clave guardada

    EEPROM.begin(512);                    //Inicio la memoria para guardar la nueva clave
    for (int i = 0; i < 7; i++) {
      EEPROM.write(i, clave_anterior[i]);
      EEPROM.write(i + 7, nueva_clave[i]);
    }
    EEPROM.commit();
    EEPROM.end();

    delay(500);
    lcd.clear();
    lcd.print("CLAVE CAMBIADA");
    if (ArduinoCloud.connected()) {
      bot.sendMessage(ID_Chat, "Se cambio la clave, Envía Pass para revisar", "");
    }
    delay(500);
    if (alarmaActivada) {                  // Si el estado de la alarma es verdadero ejecuto acciones de alaram On
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("ALARMA ENCENDIDA");
    } else {                               // Si es falso ejecuto acciones de alarma apagada
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("ALARMA APAGADA");
    }
    EncenderLCD();
    delay(500);
  } else {                                 // Si las la clave ingresada no es igual a la clave guardada se despliega calve incorrecta
    delay(500);
    lcd.clear();
    lcd.print("CLAVE INCORRECTA");
    delay(500);
  }
  indice = 0;
}
}

```

**Figura 3.28** Función Cambiar Clave

Como última acción respecto al *keypad*, si se ha presionado la letra “D” se llama a la función **EncenderLCD()**, la misma que activa la iluminación del *LCD* y se utiliza para poder observar el último mensaje que se imprimió en el *LCD*, la función se puede observar en la Figura 3.29.

```
//----- Funcion para encender el LCD -----
void EncenderLCD() {
    t = millis();
    encender = true;
    lcd.backlight();
}
```

**Figura 3.29** Función para encender la iluminación del *LCD*

Continuando con la ejecución del *loop* se ejecuta una evaluación de la variable *encender*, la cual obtiene su valor de la función *EncenderLCD*, el objetivo es apagar la iluminación desde de un periodo de tiempo, como se observa en el código de la Figura 3.30. También se observa el llamado a **ArduinoCloud.update()**, con la finalidad de actualizar la información a la *app* en *Arduino IoT Cloud*.

```
if (encender) {
    if (millis() - t > t_ms_encendido) {
        encender = false;
        lcd.noBacklight();
    }
}
ArduinoCloud.update();
```

**Figura 3.30** Apagar iluminación de *LCD*

En tercer lugar, como acción general en el bucle, se comprueba si ha existido algún mensaje o instrucción recibido desde el *bot* de *Telegram* hacia el *ESP32*. Este bloque de código verifica si ha pasado cierto tiempo desde la última vez que se ejecutó. Si ha pasado el tiempo especificado, se obtienen los mensajes nuevos recibidos por el *bot* de *Telegram*. Luego, se procesan los mensajes uno por uno mediante el llamado a la función **mensajesNuevos()** y se actualiza el tiempo anterior para contar el tiempo hasta la próxima ejecución del bloque de código, como se observa en las líneas de código de la Figura 3.31.

```
// -----Verificar mensajes de Telegram Bot API-----
if (millis() - tiempoAnterior > tiempo) // Verifica si ha pasado el tiempo especificado desde la última ejec
{
    int numerosMensajes = bot.getUpdates(bot.last_message_received + 1); // Obtiene el número de mensajes nu
    while (numerosMensajes) // Inicia un bucle while para procesar todos los mensajes nuevos
    {
        Serial.println("Comando recibido");
        mensajesNuevos(numerosMensajes); // Llama a una función llamada 'mensajesNuevos' y le pasa el número de
        numerosMensajes = bot.getUpdates(bot.last_message_received + 1); // Obtiene el número de mensajes nuevos
    }
    tiempoAnterior = millis(); // Actualiza el valor de 'tiempoAnterior' con el tiempo actual en milisegundos
}
```

**Figura 3.31** Comprobar mensajes nuevos desde *bot* de *Telegram*

La función **mensajesNuevos()** es la encargada de procesar los mensajes recibidos desde *Telegram* y ejecutar la acción correspondiente siguiendo las líneas de código que se muestran en la Figura 3.32. La función recibe la cantidad de mensajes con lo que realiza la ejecución de la instrucción *for*. Dentro de la instrucción se obtiene el ID del usuario que ha enviado el mensaje el *bot* de *Telegram* y se lo almacena en la variable **chat\_id**, también en la variable **text** se almacena el mensaje recibido desde el *bot*. Mediante la instrucción *if* se compara si el ID del usuario que ha enviado el mensaje es igual al ID del usuario que está permitido, y que está almacenado en la variable **ID\_Chat**, la cual se carga al iniciar la placa *ESP32*. Si los dos ID son iguales se procede a evaluar y ejecutar la acción correspondiente al mensaje recibido.

Como primera comparativa se verifica si el contenido de la variable **text** es igual a "Alarmaon", si es así se cambia el estado del *switch* en *Arduino IoT Cloud* a *HIGH*, se envía un mensaje al *bot* con la frase "Alarma ENCENDIDA", en el *LCD* se imprime la frase "ALARMA ENCENDIA", se asigna el valor 1 a la variable **estadoLed12**, se asigna el estado *true* a las variables **alarmaActivada**, la cual es la encargada más adelante de llamar a las funciones para el encendido o apagado de la alarma y se coloca como *true* a la variable **z**. El caso contrario ocurre cuando el mensaje recibido es "Alarmaoff", ejecutando acciones para el apagado de la alarma.

Por otro lado, si el mensaje recibido corresponde a "Estado", se evalúa el valor que se ha almacenado a la variable **estadoLed12**. Si el valor de la variable es 1, correspondiente a que la alarma esta activada, se envía como respuesta un mensaje al *bot* con la frase "Alarma ENCENDIDA" acompañado del estado de cada sensor almacenado en las variables **ac** para el *PIR*, **vn** para el sensor magnético de la ventana y **pu** para el sensor magnético de la puerta, estas variables obtienen el estado al evaluar el estado del sensor, lo cual se realiza más adelante. La misma acción ocurre si el valor de la variable **estadoLed12** es 0, se envía el mensaje "Alarma APAGADA" acompañada del estado de cada sensor.

Si el mensaje recibido es "Ayuda" se responde al *bot* con todas las instrucciones que acepta el sistema y que ejecuta cada acción, se debe mencionar que en este mensaje no se especifica el mensaje *Pass*, el cual permite recibir como respuesta la clave que tiene la alarma para el activado o apagado de la alarma, esto como medida de seguridad. Esta instrucción solo la conocerá el usuario. Finalmente, si el mensaje recibido no corresponde a ninguno de los detallados, se notifica como respuesta la frase "Escribe Ayuda para ver las opciones".

```

//----- Funcion para ejecutar los mensajes recibidos de telegram -----
void mensajesNuevos(int numerosMensajes)
{
  for (int i = 0; i < numerosMensajes; i++)
  {
    String chat_id = bot.messages[i].chat_id; // Se obtiene el ID del usuario que ha escrito al bot
    String text = bot.messages[i].text; // Se obtiene el mensaje que se ha enviadp
    if (chat_id == ID.Chat) { // Comprueba si el ID del mensaje es igual al ID del telegram del usuario
      if (text == "Alarmaon") // Si el mensaje es igual Alarmaon, ejecuto acciones de Encendido
      {
        estadoLed12 = 1;
        onoff = HIGH; // Cambio el estado switch del arduino cloud
        bot.sendMessage(chat_id, "Alarma ENCENDIDA", ""); // Envio notificacion del encendido al telegram
        cambioDesdeTelegram = true;
        alarmaActivada = true;
        z = true; //Coloco en verdadero la variable para evaluar el llamado a la funcion del buzzer para iniciar la alarma
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("ALARMA ENCENDIDA"); // Imprime el mensaje de encendido en el LCD
        EncenderLCD(); // Llmo la funcion para encender el LCD
      }
      else if (text == "Alarmaoff") // Si el mensaje es Alarmaoff ejecuto acciones de Apagado
      {
        estadoLed12 = 0; // Guardo el valor para ejecutar el estado de la alarma en telegram
        onoff = LOW;
        bot.sendMessage(chat_id, "Alarma APAGADA", "");
        cambioDesdeTelegram = true;
        alarmaActivada = false;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("ALARMA APAGADA");
        EncenderLCD();
      }
      else if (text == "Estado") // Si el mensaje es Estado
      {
        if (estadoLed12) //si el estado de la alarma es verdadero o encendido
        {
          String ayuda = "Alarma ENCENDIDA, " ".\n\n"; // Envio la notificacion con el estado de la alarma y de los sensores
          ayuda += "Sala " + ac + ".\n";
          ayuda += "Ventana " + vn + ".\n";
          ayuda += "Puerta " + pu + ".\n";
          bot.sendMessage(chat_id, ayuda, "");
        }
        else //Si el estado es falso o apagado
        {
          String ayuda = "Alarma APAGADA, " ".\n\n";
          ayuda += "Sala " + ac + ".\n";
          ayuda += "Ventana " + vn + ".\n";
          ayuda += "Puerta " + pu + ".\n";
          bot.sendMessage(chat_id, ayuda, "");
        }
      }
      else if (text == "Ayuda") //Si el mensaje es Ayuda desde telegram, responde con los comandos aceptados en el bot
      {
        String ayuda = "Bienvenido al sistema de Alarma con Esp32, " ".\n";
        ayuda += "Estas son tus opciones.\n\n";
        ayuda += "Alarmaon: Para encender la Alarma \n";
        ayuda += "Alarmaoff: Para apagar la Alarma \n";
        ayuda += "Estado : Devuelve el estado actual de la Alarma\n";
        ayuda += "Ayuda: Imprime este menú \n";
        ayuda += "Recuerda el sistema distingue entre mayuculas y minusculas \n";
        bot.sendMessage(chat_id, ayuda, "");
      }
      else if (text == "Pass"){
        bot.sendMessage(chat_id,nueva_clave,"");
      }
      else {
        bot.sendMessage(chat_id, "Escribe Ayuda para ver las opciones", ""); // Si se ingresa un comando erroneo se envia una respuesta
      }
    }
  }
}
}
}

```

**Figura 3.32** Función mensajes nuevos

Finalizado las acciones correspondientes a *Telegram*, el sistema continuo con la siguiente acción, la cual corresponde a las instrucciones desde la aplicación *de Arduino IoT Cloud* que se especifica en la Figura 3.33. En la variable **cloudactual** se almacena el estado actual de la variable **onoff**, vinculada con el *switch* de la aplicación. Con esta variable se evalúa si hubo un cambio de estado en el *switch* de la aplicación, para evitar conflictos cuando el sistema evalúa el estado del *switch* y repita acciones ya realizadas. En otra variable llamada **cloudprevio** se almacenará el estado último que tendrá el *switch*, iniciando esta con *false* al arrancar por primera vez el sistema.

Mediante la instrucción *if* se compara si los dos estados de las variables son diferentes, si es el caso, significa que el estado del *switch* cambio y se procede a ejecutar las acciones correspondientes. La variable **onoff** contiene el estado que se envía mediante la manipulación del *switch* virtual de la aplicación. Si la variable tiene valor *true* significa que el *switch* está en estado de activado, encendiendo la alarma, en este estado se evalúa si el *switch* está como activado y si la variable que almacena él envió de mensajes de *Telegram* **mensajeEnviado** es falso, si se cumple se procede a verificar en otro *if* si desde el *bot* no se ejecutó ya una acción similar de encendido, de esta forma se evita que se repita un envío doble de mensajes al *bot*.

Si lo anterior se cumple se otorga el valor de la variable **estadoLed12** en 1, se envía un mensaje a *Telegram* con la frase “Alarma ENCENDIDA”, si es la primera vez que se envía el mensaje se asigna un valor de *true* a la variable **mensajeEnviado**. También, se asigna a la variable **alarmaActivada** el estado *true*, la variable **cambioDesdeTelegram** se coloca en *false*, la variable **z** se establece como *true* y se imprime en el *LCD* la frase “ALARMA ENCENDIDA” encendiendo la iluminación mediante la función **EncenderLCD()**.

Caso contrario si la variable **onoff** tiene valor *false*, significa que el *switch* se encuentra en estado de apagado para la alarma, en este caso se realiza las mismas acciones del paso anterior, solo que dirigidas al estado de apagado. En el *LCD* se imprime el mensaje “ALARMA APAGADA”, se envía un mensaje al *bot* con la frase “Alarma APAGADA”, se asigna a la variable **estadoLed12** el valor de 0. Finalmente, la variable que controla el estado de la alarma **alarmaActivada** se coloca en *false*, al finalizar el *if* se asigna a la variable **cloudprevio** el valor de la variable **cloudactual**.

```

----- Control desde Arduino Iot Cloud -----
cloudactual = onoff; // Guardo el valor verdadero o falso de la variable del switch de cloud
if(cloudprevio != cloudactual){ // Si el estado previo del switch es verdadero y el estado actual es falso existio un cambio
if (onoff) { // Si el switch es verdadero ejecuto acciones de encendido de la alarma
  ArduinoCloud.update();
if (onoff && !mensajeEnviado) // Verificar si switch está encendido y el mensaje no se ha enviado a telegram
{
  if (!cambioDesdeTelegram) // Verificar si el cambio no fue desde Telegram
  {
    estadoLed12 = 1;
    bot.sendMessage(ID_Chat, "Alarma ENCENDIDA", ""); // Envia notificación a telegram del encendido de la alarma
  }
  if (!mensajeEnviado) // Verificar si el mensaje no se ha enviado
  {
    mensajeEnviado = true;
  }
}
}
alarmaActivada = true; // Variable global que guarada el estado Encendido o Apagado de la alarma
cambioDesdeTelegram = false;
z = true; //Coloco en verdadero la variable para evaluar el llamado a la funcion del buzzer para iniciar la alarma
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("ALARMA ENCENDIDA"); // Proyecto en el LCD
EncenderLCD(); // LLamo a la funcion para encender el LCD
}
else {
  ArduinoCloud.update();
if (!onoff && mensajeEnviado) // Verificar si SWITCH está apagada y el mensaje se ha enviado
{
  if (!cambioDesdeTelegram) // Verificar si el cambio no fue desde Telegram
  {
    estadoLed12 = 0;
    bot.sendMessage(ID_Chat, "Alarma APAGADA", ""); // Notifico a telegram del APAGADO de la alarma
  }
  if (mensajeEnviado) // Verificar si el mensaje se ha enviado
  {
    mensajeEnviado = false;
  }
}
}
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("ALARMA APAGADA");
EncenderLCD();
alarmaActivada = false;
cambioDesdeTelegram = false;
}
}
cloudprevio = cloudactual; //Guardo el estado actual del switch en el estado previo
}
}

```

**Figura 3.33** Acciones desde la *app* de *Arduino IoT Cloud*

Hasta este punto se puede resumir que el sistema de la alarma ejecuta tres acciones divididas en el *keypad*, el *bot* de *Telegram* y la aplicación en *Arduino IoT Cloud*. Dependiendo de la forma en que se manipule la alarma se asigna un valor a la variable **alarmaActivada**, el valor o estado de la variable se guarda en la memoria *EEPROM* del *ESP32*, como se observa en la Figura 3.34, con la finalidad de que, al existir un corte de energía, el sistema recupere el último estado de la alarma, sea está encendida o apagada.

```

// Guardar el estado de alarmaActivada en la EEPROM
EEPROM.begin(512);
EEPROM.write(direccionAlarmaActivada, alarmaActivada ? '1' : '0');
EEPROM.commit();
EEPROM.end();

```

**Figura 3.34** Almacenamiento del estado de la alarma en la *EEPROM*

Finalmente, en la parte final del *loop* se evalúa el estado que tiene la variable **alarmaActivada**. Si tiene un valor verdadero se evalúa mediante un *if* el estado de la variable **z**, la cual se asigna como verdadero en cada encendido de la alarma, si se



cumple dicha condición se llama a la función **buzzer()** para emitir una serie de pitidos durante 30 segundos, con la finalidad de que el usuario pueda salir del domicilio antes de que los sensores inicien la detección. Acto seguido se coloca la variable *z* como *false* y se llama a la función **alarmaOn()**, caso contrario, si tiene un estado falso en la variable **alarmaActiva**, se llama a la función **alarmaOff()**, como se muestra en el código de la Figura 3.35.

```

if (alarmaActivada) { // Si el estado de la alarma es verdadero ejecuto acciones de alarm On
  if(z){ // Evalua el estado de la variable z es verdadero, otorgado en el encendido de la alarma
    buzzer();} //Llama a la función buzzer para emitir pitidos durante 30 segundos
    z = false; //Coloca la variable z como false para evitar el llamado a a funcion buzzer de manera constante
    alarmaOn(); //Llama a ala funcion alarma on para acciones de lectura de sensores y notificacion
  } else { // Si es falso ejecuto acciones de alarma apagada
    alarmaOff();
  }
}

```

**Figura 3.35** Ejecución del Encendido o Pagado de la alarma

La función **buzzer()**, que puede visualizarse en la Figura 3.36, reproduce una secuencia de sonidos utilizando el *buzzer*. La secuencia comienza estableciendo el tiempo de inicio y el tiempo transcurrido en cero. Luego, dentro de un bucle, se calcula el progreso actual de la secuencia y se ajusta la frecuencia del *buzzer* en función de ese progreso. Se reproduce un tono con la frecuencia actual durante 100 (ms), se espera un breve intervalo de 300 (ms) y se detiene el sonido durante otros 300 (ms) antes de pasar a la siguiente iteración. Este proceso continúa hasta que el tiempo transcurrido alcance la duración especificada. Al final de la secuencia, se reproduce un último tono más fuerte durante 100 (ms), seguido de una espera de 300 (ms) antes de detener completamente el sonido del *buzzer*.

```

void buzzer(){
  unsigned long startTime = millis(); // tiempo de inicio
  unsigned long elapsedTime = 0; // tiempo transcurrido

  while (elapsedTime < tiempo_ejecute * 1000) {
    float progress = (float)elapsedTime / (tiempo_ejecute * 1000); // progreso actual (0-1)
    int currentFrequency = map(progress, 0, 1, frecuencia_inicio, frecuencia_final);
    tone(BUZZER_PIN, currentFrequency, 100); // reproduce el tono con la frecuencia actual durante 100ms
    delay(300); // espera 200ms antes de cambiar a la siguiente frecuencia
    noTone(BUZZER_PIN); // detiene el sonido durante 200ms
    delay(300); // espera 200ms antes de continuar

    elapsedTime = millis() - startTime; // actualiza el tiempo transcurrido
  }

  // Último pitido más fuerte
  tone(BUZZER_PIN, frecuencia_final, 100);
  delay(300); // espera 300ms con el último tono fuerte
  noTone(BUZZER_PIN); // detiene el sonido
  //delay(1000); // espera 1 segundo antes de volver a empezar
}

```

**Figura 3.36** Función *buzzer*

La función **alarmaOn()** es la encargada de ejecutar las acciones de lectura y notificación a cada una de las plataformas, del estado de los sensores utilizados, tal como se

muestra en la codificación de la Figura 3.37. Para iniciar se lee los pines donde están conectados los dos sensores magnéticos y el sensor *PIR*. En la variable **sensorValue** se almacena el valor de la variable **ven**, que es la que tiene asignado el pin para el sensor magnético de la ventana, en la variable **sensorValuep** se almacena el valor de la variable **pue**, la cual está relacionado con el sensor magnético de la puerta y la variable **motionsStateCorrente** almacena el valor de la variable **MOTION\_SENSOR\_PIN**, la cual tiene asignado el pin para el *PIR*.

Al llamar a la función **alarmaOn()**, se realiza el encendido de un *LED*, mediante el cambio de estado a la variable **ledOn**, colocándolo en estado *HIGH*. Mediante la estructura *if* se evalúa el estado actual del sensor *PIR* al comparar si la variable **motionStatePrevius**, que tiene el último valor de la variable **motionsStateCorrente**, se encuentra en estado *LOW* y si la variable **motionsStateCorrente** tiene el estado *HIGH*, al cumplirse significa que existió un cambio de estado en el sensor *PIR*, por lo tanto, se detectó la presencia de una persona o animal. Acto seguido, se evalúa si existe conexión con la plataforma *Arduino Cloud*, si es así se coloca en estado *HIGH* a la variable **pir**, la cual está asignada a un *widget status* en la aplicación de *Arduino IoT Cloud*. También, se envía una notificación al *bot* de *Telegram* con la frase “MOVIMIENTO EN LA SALA !!!”.

Cada que exista un cambio de estado en el sensor *PIR*, se encenderá un *LED* asignado en la variable **ledPin**. A la variable **ac**, que se mencionó en pasos anteriores, se le asigna el mensaje “CON MOVIMIENTO”. Por otro lado, si el estado del *PIR* pasa de *HIGH* a *LOW* se coloca el *LED*, mediante la variable **ledPin**, en estado *LOW*. Por otro lado, con otra estructura *if* se evalúa si el estado de la variable **sensorValue** se encuentra en estado *HIGH*, si se cumple la condición significa que el sensor magnético de la ventana se abrió. Se evalúa si existe conexión con la plataforma de *Arduino Cloud* para asignar el estado *HIGH* a la variable **ventana**, que está relacionado con *widget status* en la aplicación de *Arduino IoT Cloud*. Además, se envía una notificación a *Telegram* con la frase “VENTANA ABIERTA !!!” y se asigna el mensaje “ABIERTA” a la variable **vn**.

Acción similar ocurre con la variable **sensorValuep**, relacionada con la variable del sensor de la puerta. Finalmente se evalúa en un *if* el estado de los tres sensores, si cualquiera de ellos se colocó en estado alto, se procede a colocar en estado *HIGH* la variable **BUZZER\_PIN**, es decir sonara de manera constante hasta que se realice el apagado de la alarma. También, se asigna el valor de 1 a la variable **mov**, la cual está

asignada a un *widget chart* y *value selector* en la aplicación. En el *LCD* se imprime un mensaje “INTRUSO !!!” acompañado del área comprometida.

```
//-----Funcion para ejecutar acciones de los sensores cuando esta la alarma ON -----
void alarmaOn() {
  int sensorValue = digitalRead(ven); //Lee el valor del sensor de ventana
  int sensorValuep = digitalRead(pue); //Lee el valor del sensor de puerta
  motionStatePrevious = motionStateCurrent; //Asigno el valor del estado actual del PIR a la variable de estado previo
  motionStateCurrent = digitalRead(MOTION_SENSOR_PIN); // Asigo el valor del sensor PIR a la variable de estadoa ctual
  digitalWrite(ledOn, HIGH); // Enciende el LED para mostrar que la alarma esta encendida
  sla = digitalRead(MOTION_SENSOR_PIN);
  if (motionStatePrevious == LOW && motionStateCurrent == HIGH) { // Evaluo si el estado del PIR cambio de bajo a alto
    if (ArduinoCloud.connected()) { //Evaluo si existe conexion a una de las plataformas, lo que significa conexion
      pir = HIGH; //Si hay conexion informo del estado del PIR a cloud y telegram
      bot.sendMessage(ID_Chat, "MOVIMIENTO EN LA SALA !!!", "");
    }
    digitalWrite(ledPin, HIGH); //Se enciende el led rojo con cada movimiento
    ac = "CON MOVIMIENTO"; // Asigno una frase para utilizarla a la hora de ver el Estado en telegram
  } else if (motionStatePrevious == HIGH && motionStateCurrent == LOW) { // Evaluo si el estado del PIR paso de alto a bajo
    digitalWrite(ledPin, LOW); //Apago el led
  }

  if (sensorValue == HIGH) { //Evaluo si el estado del sensor de ventan es alto
    if (ArduinoCloud.connected()) { //Evaluo la conexion
      ventana = HIGH; //Muestro el estado de activo en cloud
      bot.sendMessage(ID_Chat, "VENTANA ABIERTA !!!", ""); //Notifico en telegram
      vn = "ABIERTA"; //Asigno la frase para el Estado en Telegram
    }
  }

  if (sensorValuep == HIGH) { // Evaluo Si el estado del sensor de Puerta es alto
    if (ArduinoCloud.connected()) { //Evaluo si hay conexion
      puerta = HIGH; //Muestro el estado en cloud
      bot.sendMessage(ID_Chat, "PUERTA ABIERTA !!!", ""); //Notifico en telegram
      pu = "ABIERTA"; //Asigno la frase para el Estado de Telegram
    }
  }

  if (motionStateCurrent || sensorValuep || sensorValue) { //Si alguno de los sensores se activo
    digitalWrite(BUZZER_PIN, HIGH); // El buzzer emite un sonido constante
    if (ArduinoCloud.connected()) { //Evaluo la conexion
      mov = 1; //Asigno estado de 1 al chart de cloud
    }
  }
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(" INTRUSO !!!"); // Proyecto en el LCD un mensaje de ALERA con el sensor activado
  if(motionStateCurrent){
    lcd.setCursor(0, 1);
    lcd.print("SALA MOVIMIENTO");
  }else if(sensorValuep){
    lcd.setCursor(0, 1);
    lcd.print("PUERTA ABIERTA");
  }else if(sensorValue){
    lcd.setCursor(0, 1);
    lcd.print("VENTANA ABIERTA");
  }
  EncenderLCD();//Enciendo el LCD
}
}
```

**Figura 3.37** Función para encendido de la alarma

La función **alarmaOff()** es la encargada de las acciones del apagado a la alarma. En la Figura 3.38 se muestra los códigos que la componen. Al llamar a la función se coloca la variable **ledOn** en estado **LOW**, al igual que la variable **BUZZER\_PIN**. Se evalúa si existe conexión a la plataforma para asignar el valor de 0 a la variable **mov** y asignar el valor actual de la lectura del sensor **PIR**, mediante la variable **pir**. La variable **sla** almacena el estado o valor del sensor **PIR**, la cual se evalúa para asignar un estado al **LED** en la variable **ledPin** y asignar a la variable **ac** la frase “CON MOVIMIENTO” o “SIN MOVIMIENTO”, esto dependiendo del estado del sensor **PIR**. Respecto a los sensores magnéticos de la puerta y ventana se realiza una acción similar a la del **PIR**.

```

//-----Funcion para el Apagado de la Alarma-----
void alarmaOff() {
  digitalWrite(ledOn, LOW);           //Apago el led que muestra el estado encendido o apagado
  digitalWrite(BUZZER_PIN, LOW);     //Silencio el Buzzer
  if (ArduinoCloud.connected()) {    //Verifico la conexion
    mov = 0;
    pir = digitalRead(MOTION_SENSOR_PIN);
  }
  digitalWrite(ledPin, digitalRead(MOTION_SENSOR_PIN)); //Enciendo el led acorde al estado del PIR
  sla = digitalRead(MOTION_SENSOR_PIN);
  if(sla){
    ac = "CON MOVIMIENTO";
    digitalWrite(ledPin, HIGH);
  }else{
    ac = "SIN MOVIMIENTO";
    digitalWrite(ledPin, LOW);
  }
}

ventana = digitalRead(ven);           //Aigno el estado del sensor de ventana a la variable en cloud
vtana = digitalRead(ven);
if (vtana){
  vn = "ABIERTA";
}else{
  vn = "CERRADA";
}

puerta = digitalRead(pue);           //Aigno el estado del sensor de Puerta a la variable en cloud
prta = digitalRead(pue);
if (prta){
  pu = "ABIERTA";
}else{
  pu = "CERRADA";
}
}

```

**Figura 3.38** Función para apagado de la alarma

Es importante mencionar que en las líneas de código se realiza en varios momentos la evaluación de existencia de conexión con la plataforma de *Arduino Cloud*, esto se lleva a cabo con la finalidad de evitar que al ejecutar el programa se busque a las variables virtuales de la aplicación y del *bot* de *Telegram*, si no existe conexión a *Internet*, ya que puede ocasionar retardos en la ejecución de sistema.

En las siguientes Figura 3.39, Figura 3.40, Figura 3.41, Figura 3.42, Figura 3.43 y Figura 3.44 se puede visualizar el diagrama de flujo del código del sistema de la alarma. El diagrama muestra el orden secuencial en que se realizan cada acción.

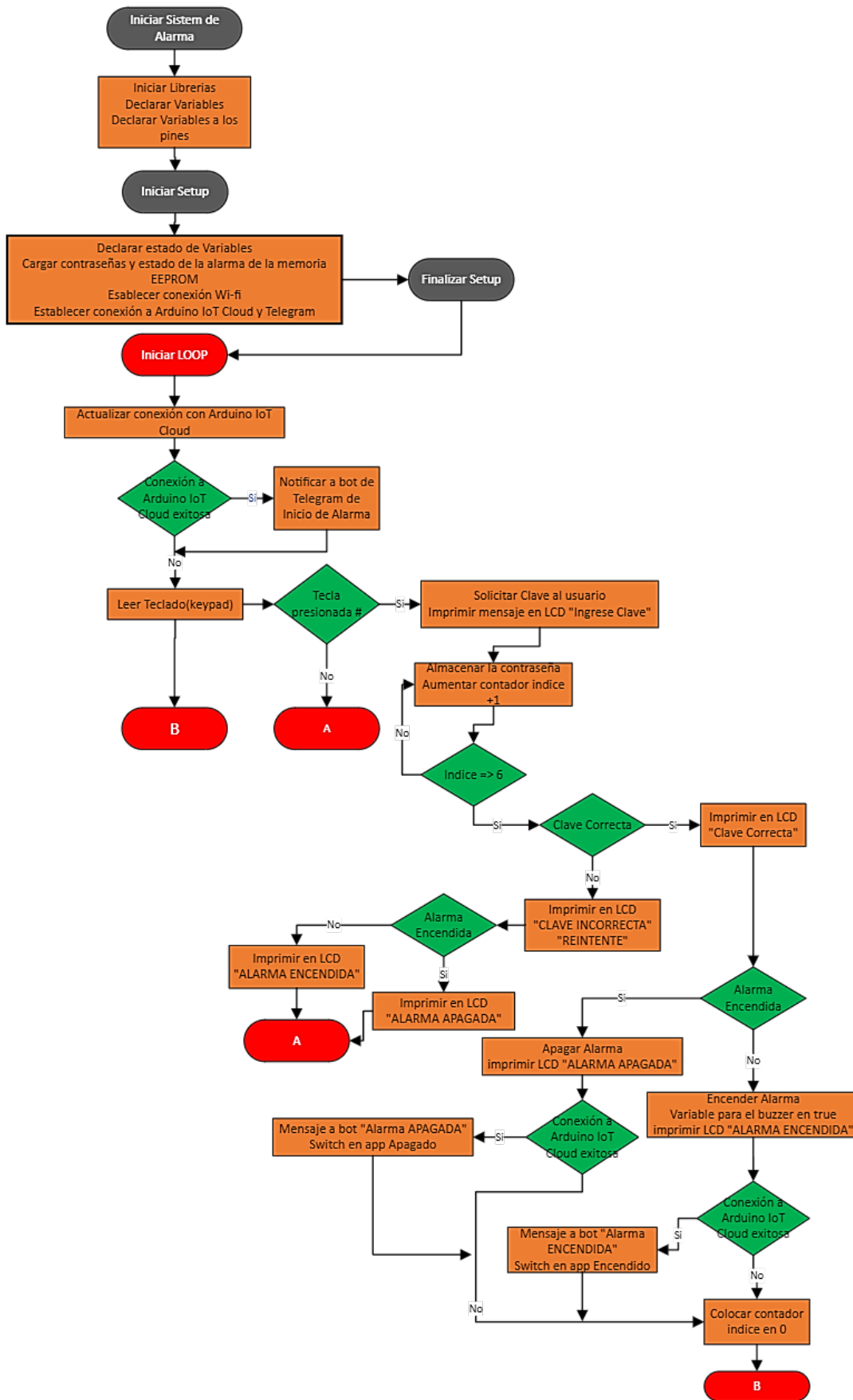


Figura 3.39 Diagrama de Flujo de Alarma-Parte 1

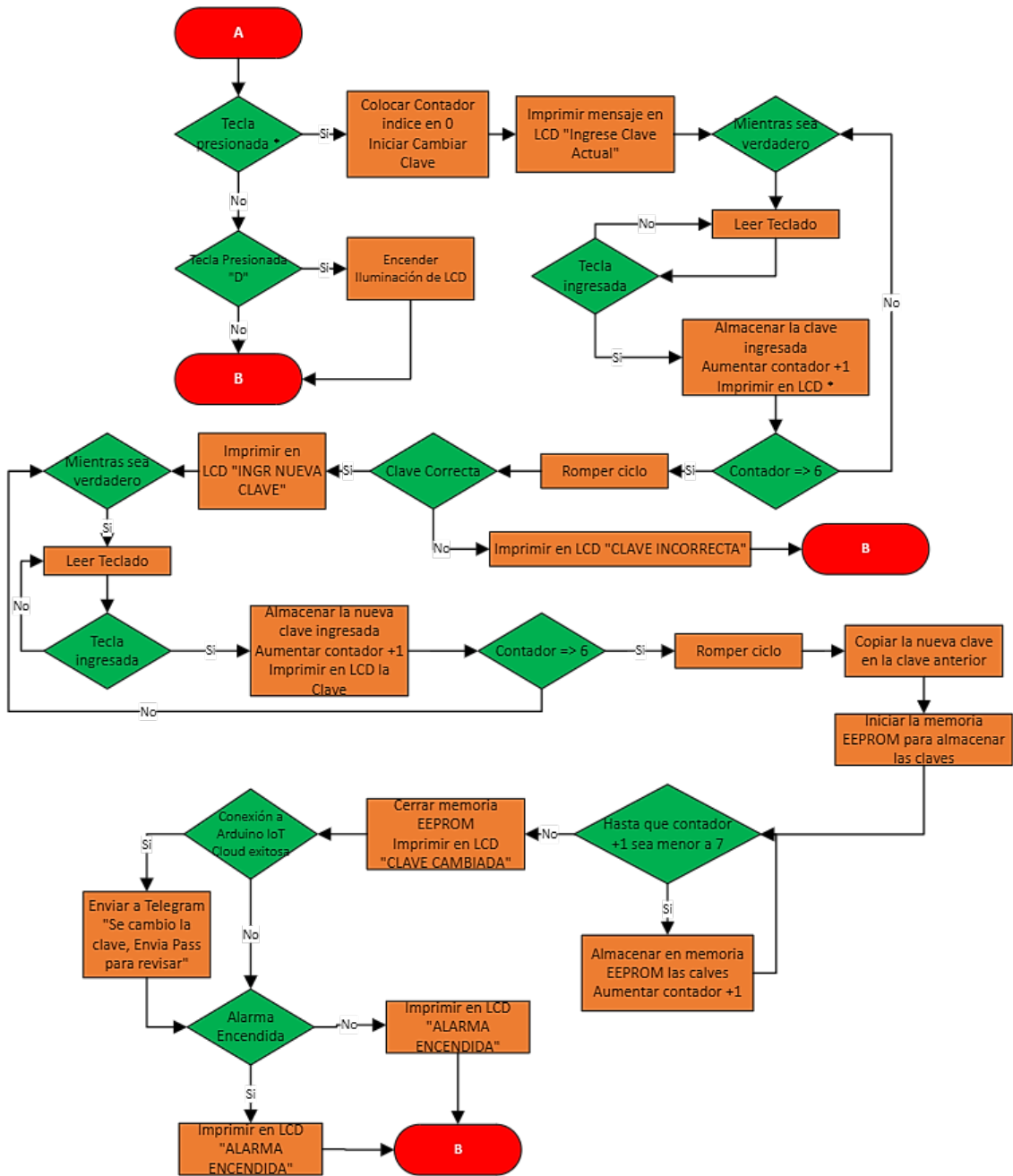


Figura 3.40 Diagrama de Flujo de Alarma-Parte 2

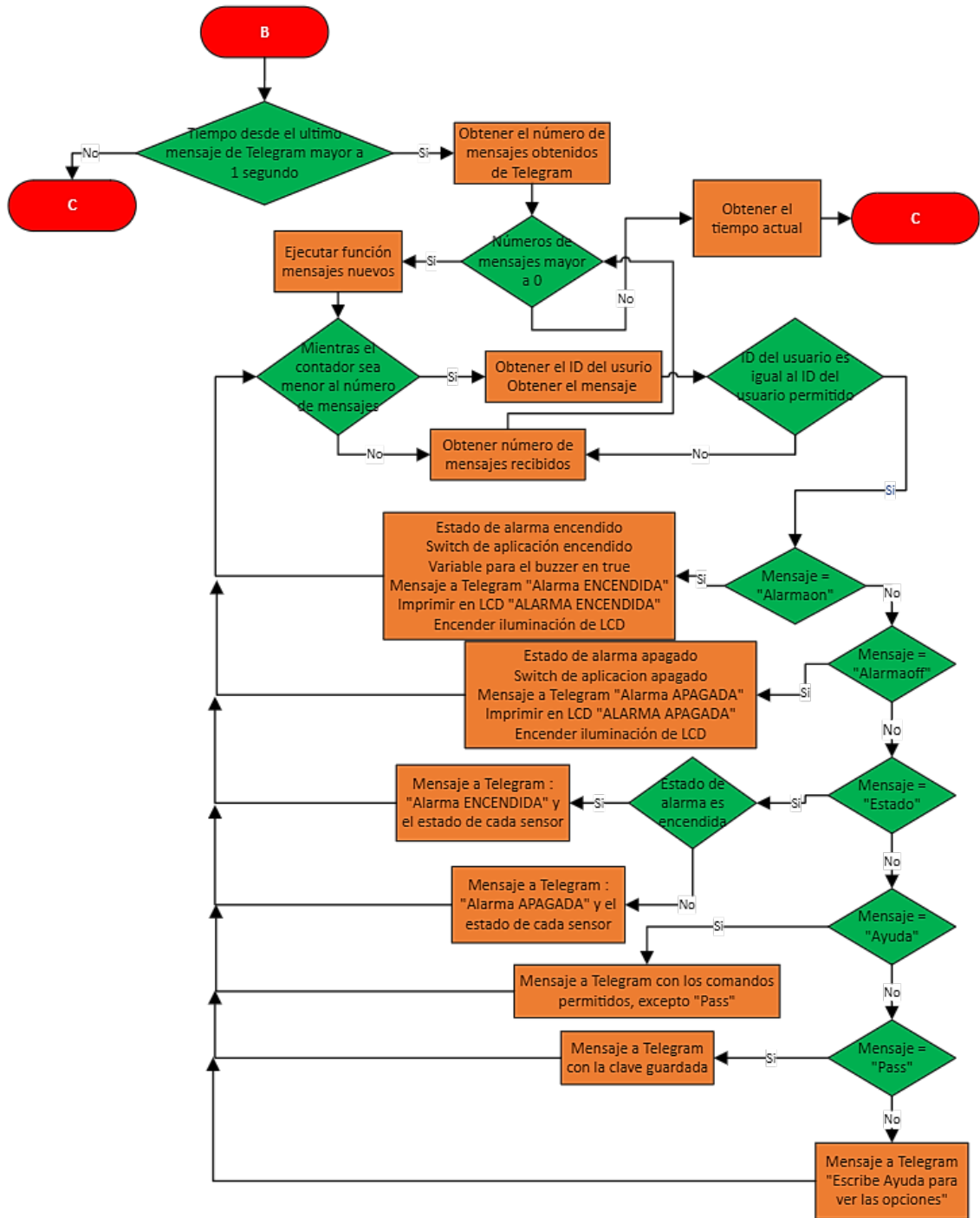


Figura 3.41 Diagrama de Flujo de Alarma-Parte 3





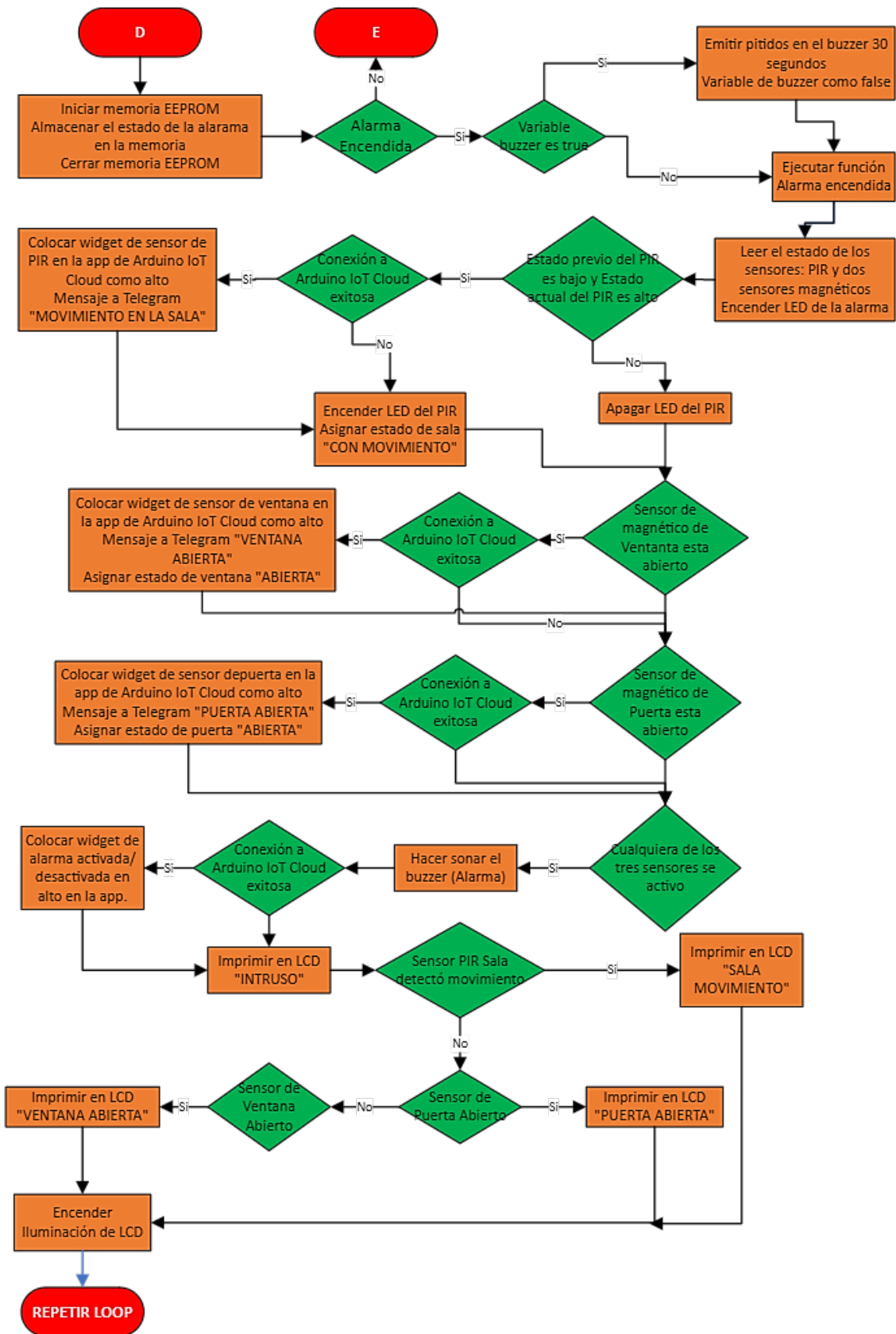


Figura 3.43 Diagrama de Flujo de Alarma-Parte 5

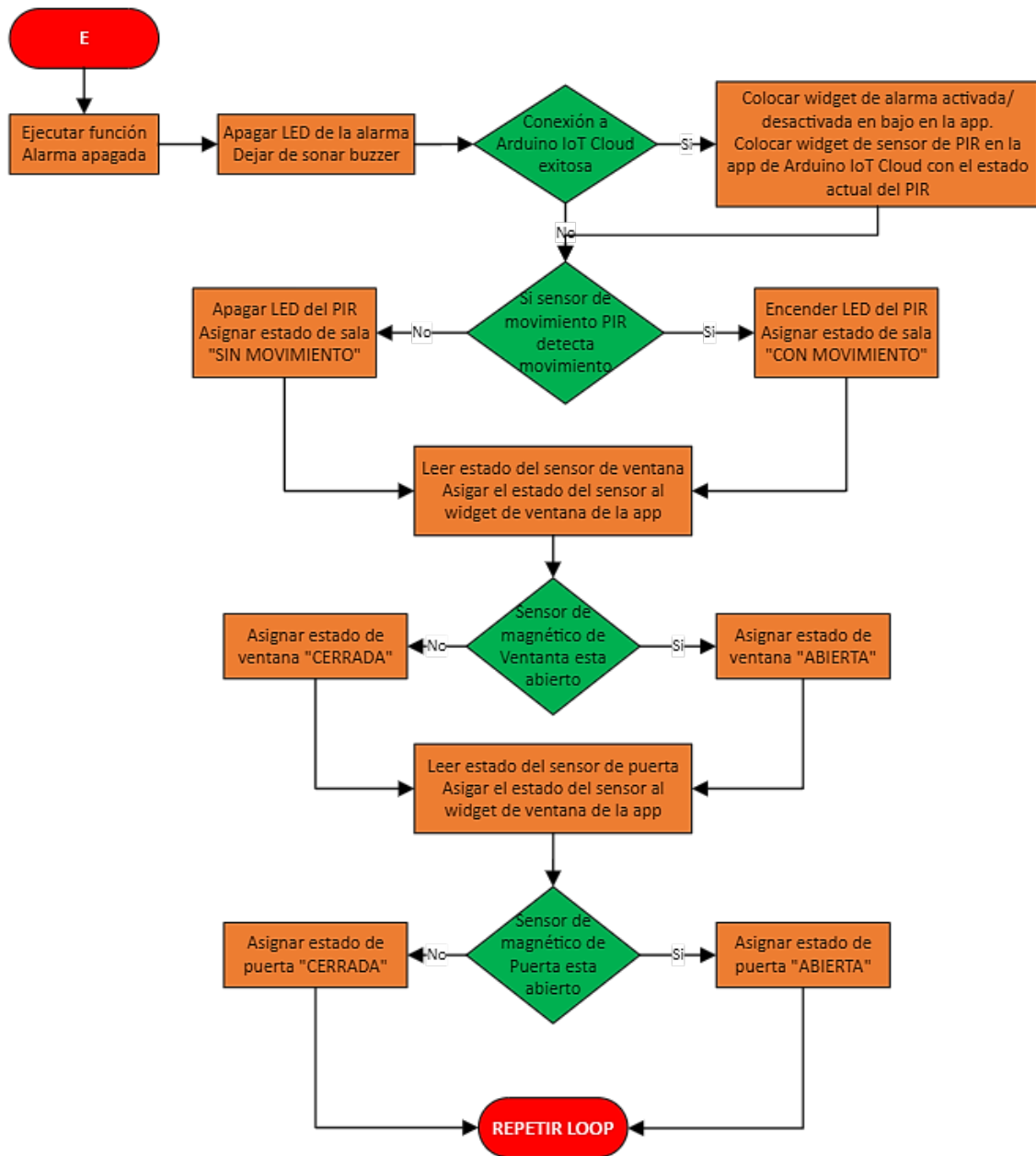
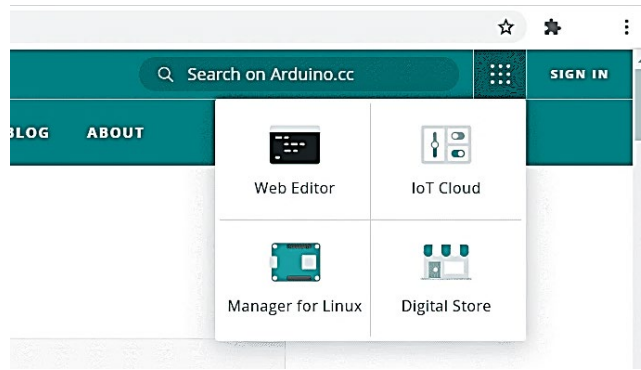


Figura 3.44 Diagrama de Flujo de Alarma-Parte 6

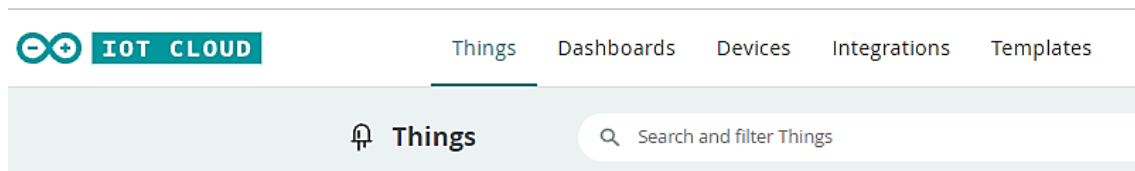
### Creación de la aplicación

En esta sección se llevará acabo la creación de la aplicación en la plataforma *Arduino IoT Cloud*. Para ingresar en la plataforma es necesario que el usuario cuente con las credenciales de autenticación, las cuales se utilizó para desarrollar el código en *Web Editor*. En la plataforma *Arduino IoT Cloud*, sección *IoT Cloud*, que muestra la Figura 3.45, es en donde se desarrollara la aplicación para el control de la alarma a través de *Internet*.



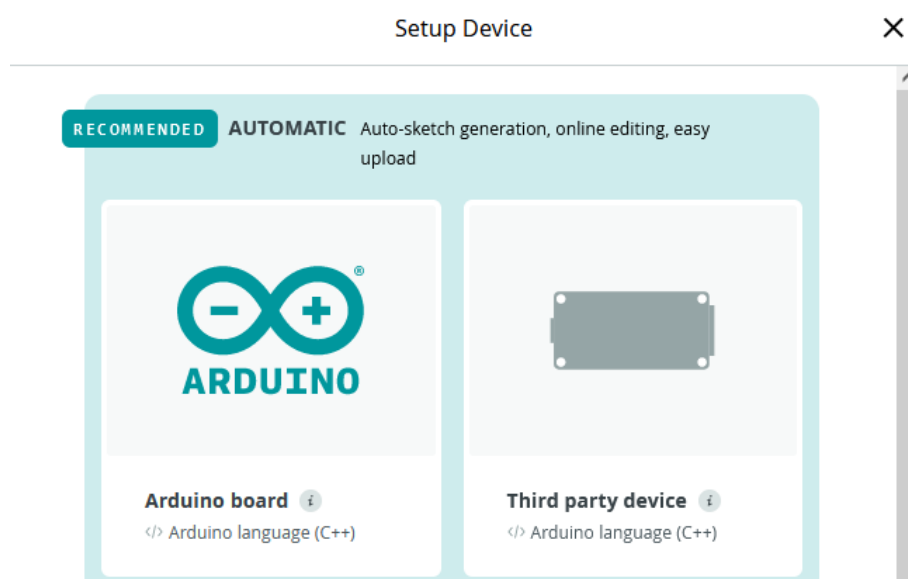
**Figura 3.45 IoT Cloud**

Una vez dentro se mostrarán cinco secciones: *Things*, *Dashboards*, *Devices*, *Integrations* y *Templates*, tal como muestra la Figura 3.46. De las cuales se utilizaron las tres primeras



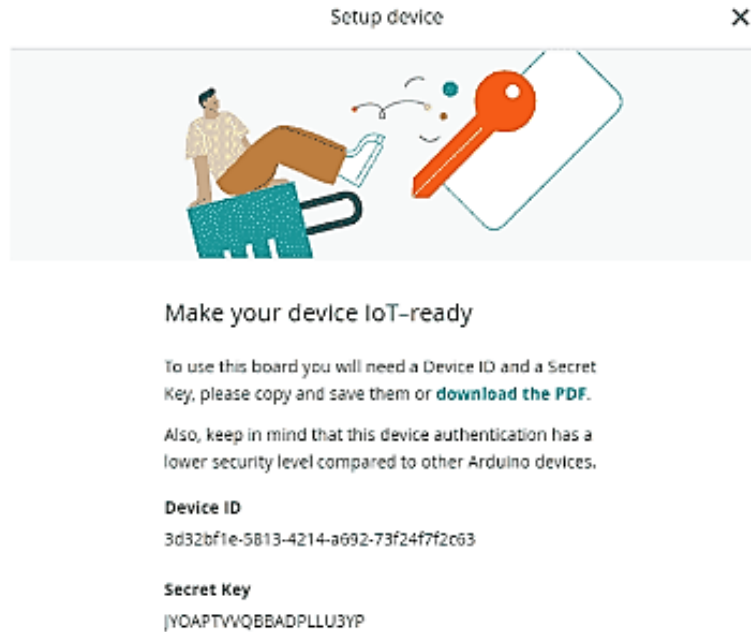
**Figura 3.46 Secciones de IoT Cloud**

La primera configuración se llevó a cabo en la sección *Devices*, la cual permite vincular la placa de desarrollo a ser utilizada, en este caso el *ESP32*. En la Figura 3.47 se muestra el apartado para la selección del dispositivo, para este proyecto fue en *Third party device*. Siguiendo las instrucciones se procedió a vincular la placa, es necesario mencionar que se debe contar con el modelo exacto de la placa para que no existan errores de compilación con el código.



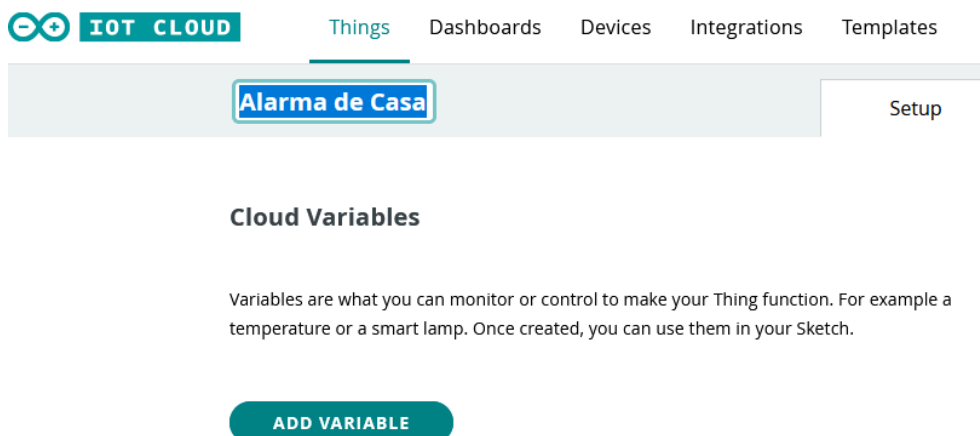
**Figura 3.47 Setup Device**

Finalizado la vinculación de la placa *ESP32*, se entrega al usuario una llave secreta y un ID relacionados con el *ESP32*, como se visualiza en la Figura 3.48, los cuales permitirán establecer la comunicación entre la aplicación creada en *Arduino IoT Cloud* y el microcontrolador. Estas credenciales se guardan de manera automática en la librería *thingProperties.h*, utilizada en el desarrollo del código.



**Figura 3.48** Key e ID de *ESP32*

El siguiente paso es desarrollar una cosa o *thing*. En esta sección, que observa en la Figura 3.49, se establecerá las variables a ser utilizadas, así como vincular el dispositivo ya creado e ingresar las credenciales de la red *Wi-fi* a utilizar para la conexión de la placa de desarrollo con *Internet*.



**Figura 3.49** *Thing*

En la Figura 3.50 se muestran las variables utilizadas en el proyecto, descritas en el apartado de desarrollo del código. Estas variables se definen en la sección *setup* de *things*. Se debe mencionar que al utilizar la versión gratuita de *Arduino IoT Cloud*, solo se tiene acceso a 5 variables, lo que para el desarrollo del prototipo son suficientes.

**Alarma de Casa** Setup

**Cloud Variables** ADD

Name ↓	Last Value	Last Update
<input type="checkbox"/> <b>mov</b> int mov;	1	27 Jun 2023 08:01:25
<input type="checkbox"/> <b>onoff</b> bool onoff;	false	21 Jun 2023 22:03:30
<input type="checkbox"/> <b>pir</b> bool pir;	false	21 Jun 2023 22:01:52
<input type="checkbox"/> <b>puerta</b> bool puerta;	false	21 Jun 2023 21:53:49
<input type="checkbox"/> <b>ventana</b> bool ventana;	false	21 Jun 2023 22:03:31

**Figura 3.50** Declaración de variables

En esta misma sección se realiza la asociación del dispositivo *ESP32* y el establecimiento de las credenciales de la red *Wi-fi*, como muestra la Figura 3.51.

**Associated Device**

**My\_esp32**

ID: 569471ae-bef2-4025-b476-f...  
 Type: NodeMCU-32S  
 Status: Offline

Change Detach

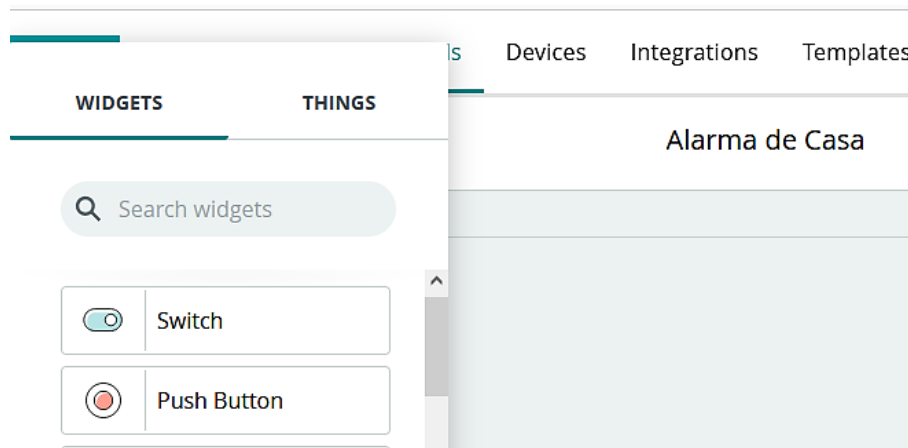
**Network**

Wi-Fi Name: TUNITA  
 Password: .....  
 Secret Key: .....  
 Change

**Figura 3.51** Device y Network

Tanto las variables, el dispositivo asociado y las credenciales de la red *Wi-fi* se almacenan dentro de la librería *thingProperties.h*. La programación del código se puede realizar en el apartado *Sketch* de *Things* o como se realizó en el paso anterior en *Web Editor*.

El siguiente paso es desarrollar el *dashboard* o tablero. En esta sección se seleccionan cada uno de los *widgets* que se vincularán con las variables creadas para la *thing* de la Alarma de Casa. El aparatado widget del *dashboard*, que se observa en la Figura 3.52, permite elegir cada elemento para que realice una acción acorde a las variables.

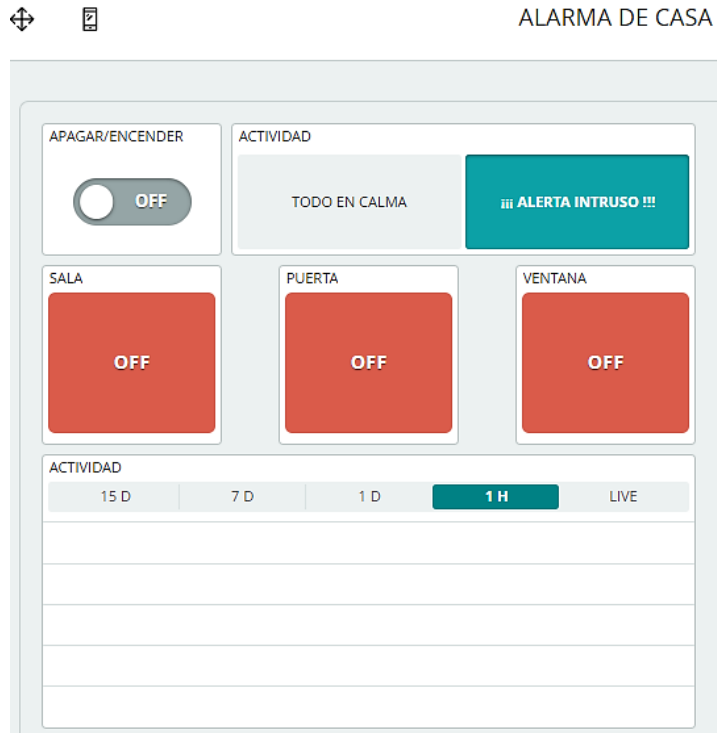


**Figura 3.52** Widget - Dashboard

Para el control de la alarma se seleccionaron los siguientes *Widgets*:

- *Switch*: Vinculado en la variable *onoff*, este *widget* permitirá encender o apagar la alarma desde el *dashboard* o desde la aplicación móvil.
- *Value Selector*: Vinculado con la variable *mov*, este *widget* permitirá visualizar como se encuentra la alarma, colocando de color el mensaje “TODO EN CALMA” o “!!! ALERTA INTRUSO !!!”, dependiente del valor 0 o 1 que tenga la variable *mov*.
- *Status*: Se utilizaron tres *widgets*, vinculados a las variables *pir*, *puerta* y *ventana*. Estos *widgets* se colocarán con *off* si el sensor o variable este en estado bajo, caso contrario se colocara en *on*, respectivamente en cada sensor.
- *Chart*: Vinculado con la variable *mov*. Este *widget* permitirá mostrar una gráfica sobre la actividad de la alarma, colocando un 1 si cualquier sensor se activó o 0 si cualquier sensor no se activó, siempre que la alarma este encendida. Se debe mencionar que al ser la versión gratuita los datos se almacenarán por 24 horas.

En la Figura 3.53 se puede observar el *dashboard*, con los *widgets* detallados, y que permite realizar el control de la alarma, así como visualizar el estado de los sensores utilizados.



**Figura 3.53** *Dashboard* de Alarma de Casa

En esta misma sección se puede colocar la posición de los *widgets* y que se utilizaran desde la aplicación móvil, para este proyecto se utilizó la distribución acorde a la Figura 3.54.



**Figura 3.54** *Dashboard* móvil

## Creación del *bot* de *Telegram*

Para crear el *bot* se utilizó el *bot BotFather* que integra la plataforma *Telegram*. Esta herramienta permite crear el *bot* de manera sencilla. En la sección de búsqueda de *Telegram* se digita *BotFather*, se selecciona y una vez dentro mediante el comando */start* se desplegará las opciones que ofrece *BotFather* para crear todo tipo de *bot*.

Mediante la instrucción */newbot* se procede a crear el nuevo *bot*, con el nombre **Alarma\_casa\_bot**. También, se seleccionó el nombre que tendrá el usuario y finalmente, se mostrará un *token* de acceso para poder utilizar el *bot* desde otra herramienta, en este caso desde *ESP32*. Todo este proceso se visualiza en la Figura 3.55.

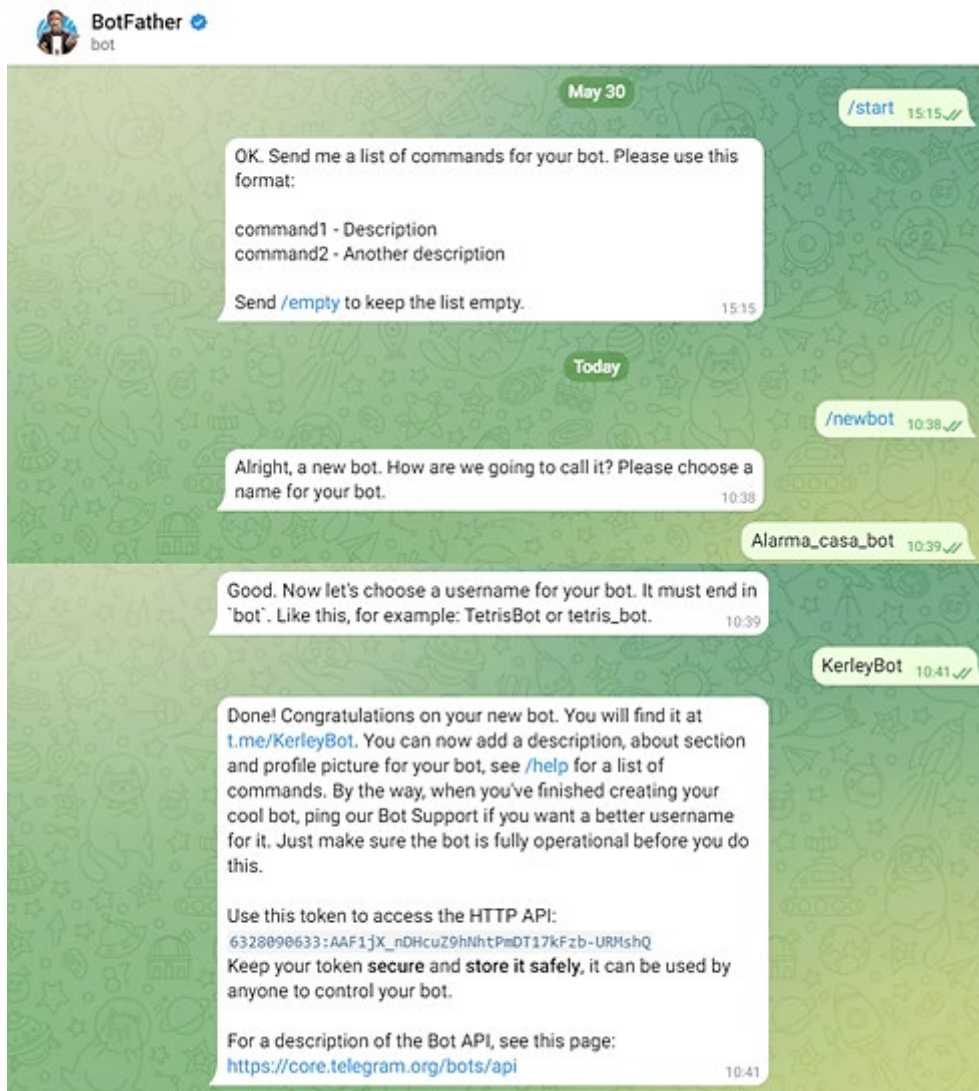
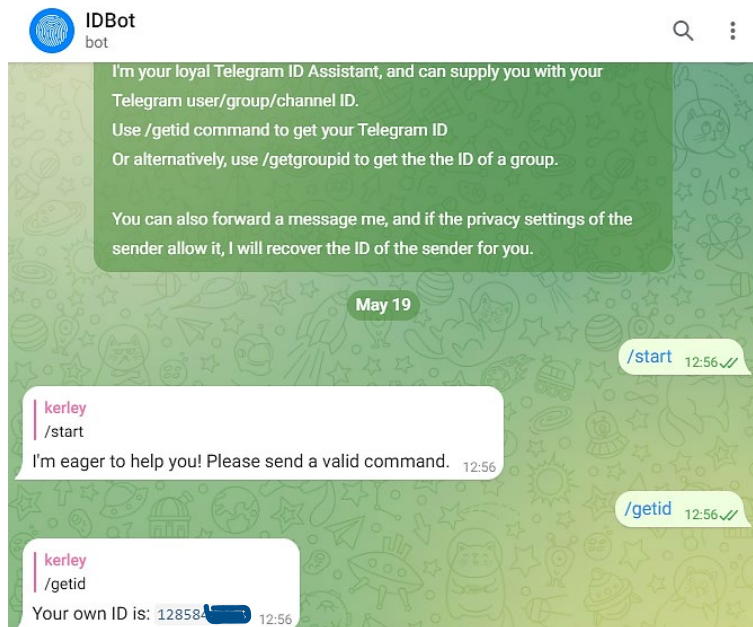


Figura 3.55 *Bot* de *Telegram*



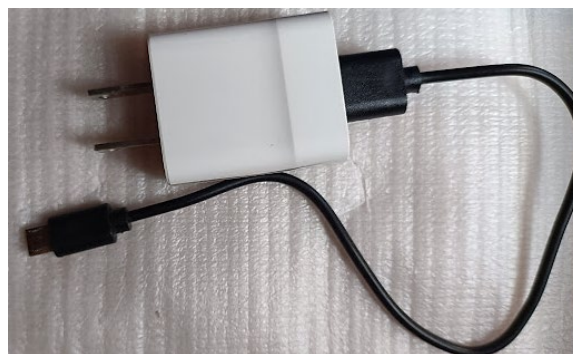
Mediante otro *bot* de la misma plataforma llamado *IDBot* se puede obtener el ID personal del usuario que estará autorizado para utilizar el *bot* creado en *Telegram*, como se observa en la Figura 3.56. Tanto el *token* como el ID se utilizaron para el desarrollo del código.



**Figura 3.56** ID personal

## Alimentación

El sistema final se alimentará con un puerto micro-USB, integrado en el microcontrolador *ESP32*. La conexión será mediante un cargador similar al de la Figura 3.57, que proporciona un voltaje de 5 (V) a la salida, el mismo que será conectado al tendido eléctrico del domicilio. El cargado a utilizar en el prototipo, es de la marca comercial *Huawei*, que cuenta con un voltaje de salida de 5 (V) y corriente máxima de salida de 1 (A). Dado que el valor de voltaje de operación del *ESP32* está en el rango de 3.3 (V) a 5 (V) y corriente máxima de 500 (mA), el cargador elegido es capaz de proporcionar el voltaje y corriente para un correcto funcionamiento.



**Figura 3.57** Cargador micro USB

### 3.4 Implementación del prototipo

El proceso de implementación del prototipo se desarrolló sobre una maqueta que emula un domicilio unifamiliar del Ecuador.

#### Desarrollo de la placa de circuito impreso

El desarrollo de la placa de circuito impreso (PCB) se llevó a cabo siguiendo el diseño creado en el *software proteus*. *Proteus* permitió desarrollar el PCB acorde a las necesidades del circuito, el cual se implementará sobre una baquelita de 9 x 8 (cm) de ancho y largo, respectivamente. Debido a la inexistencia de algunas librerías, para los elementos utilizados en el circuito, fue necesario utilizar herramientas de las librerías ya existentes para emular los orificios en donde se soldarán los elementos. En la Figura 3.58 se observa el diseño final del PCB a implementarse sobre la baquelita.

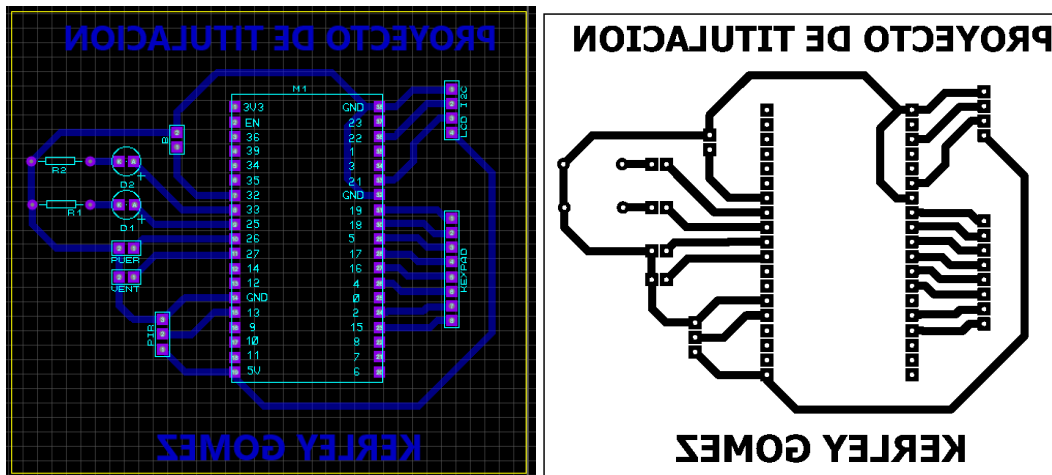


Figura 3.58 Placa de circuito impreso

Debido a la cercanía de las pistas y de los orificios se requirió de una gran exactitud a la hora de desarrollar la placa PCB, por lo cual se optó por utilizar la técnica de control numérico computarizado (CNC), que se puede observar en la Figura 3.59. Mediante CNC se logró que la placa cumpla con las medidas adecuadas, obteniendo el resultado mostrado en la Figura 3.60.

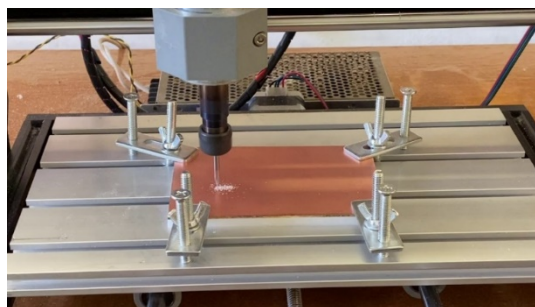
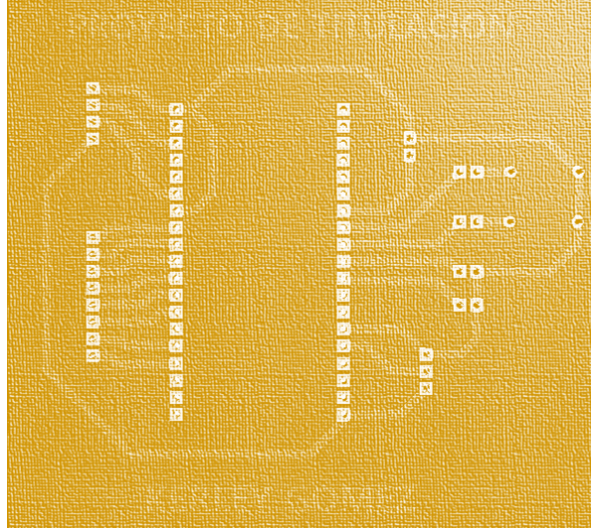
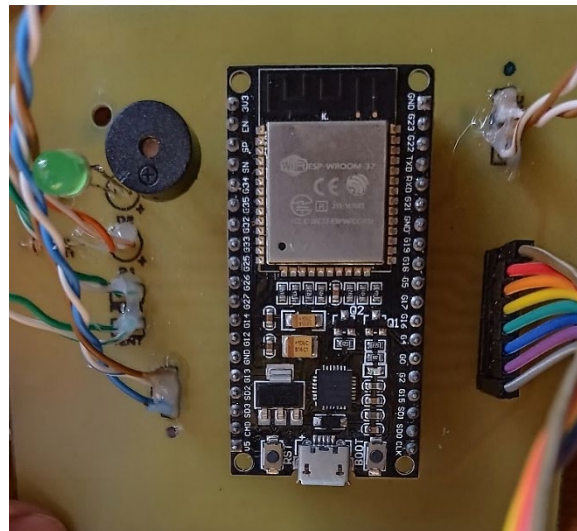


Figura 3.59 CNC



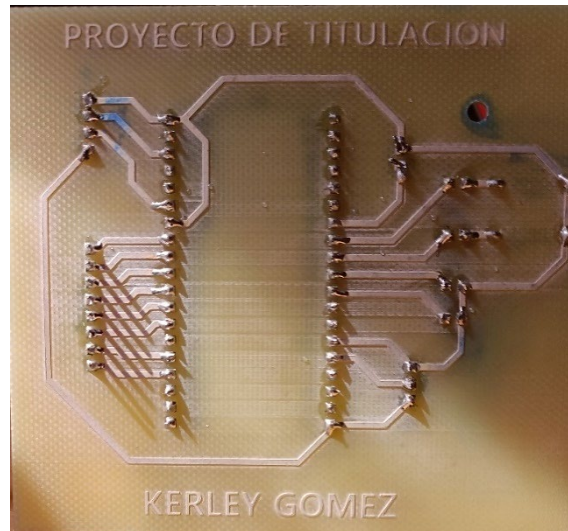
**Figura 3.60** Placa PCB de Alarma

Con la placa lista y perforada se procedió a colocar los elementos acordes al circuito eléctrico, tal como se muestra en la Figura 3.61. En la placa los elementos fijos a quedar fueron el *ESP32*, el *LED* verde y el *buzzer*. Los demás elementos requieren de cableado para colocarse en el lugar final de funcionamiento. Para el *keypad* se requirió de jumper macho – macho, con un extremo soldado a la placa y el otro conectado al teclado. Por otro lado, el *LCD* y los sensores fueron conectados mediante cable *AWG (American Wire Gauge)* calibre 24.



**Figura 3.61** Elementos montados sobre PCB

Con cada elemento colocado de manera correcta acorde al diseño se soldó en el reverso de la placa cada uno de los pines sobre las pistas del PCB. El soldado se realizó usando cañón y estaño, tratando de evitar el contacto entre dos puntos, por su cercanía. Proceso que se puede visualizar en la Figura 3.62.



**Figura 3.62** Soldado y placa final

### **Elaboración de la caja contenedora del circuito**

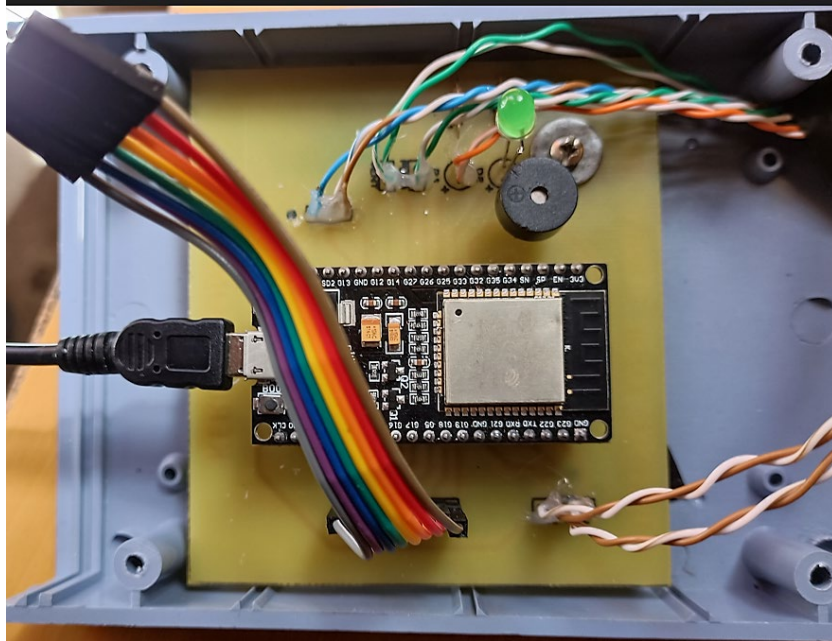
Debido a que el sistema de alarma será colocado en el interior de un domicilio, se decidió por utilizar una caja plástica con dimensiones de 150 (mm) x 100 (mm) x 50 (mm), tal como se muestra en la Figura 3.63. Estas dimensiones permiten una correcta manipulación de los elementos y cables en el interior de la caja, de esta manera se evita algún contacto innecesario. Debido a que la caja es de venta comercial, se requirió realizar algunos cortes para los orificios en donde se colocara el *LCD*, el *keypad*, el LED verde y para la salida de los cables de cada sensor.



**Figura 3.63** Caja contenedora del circuito

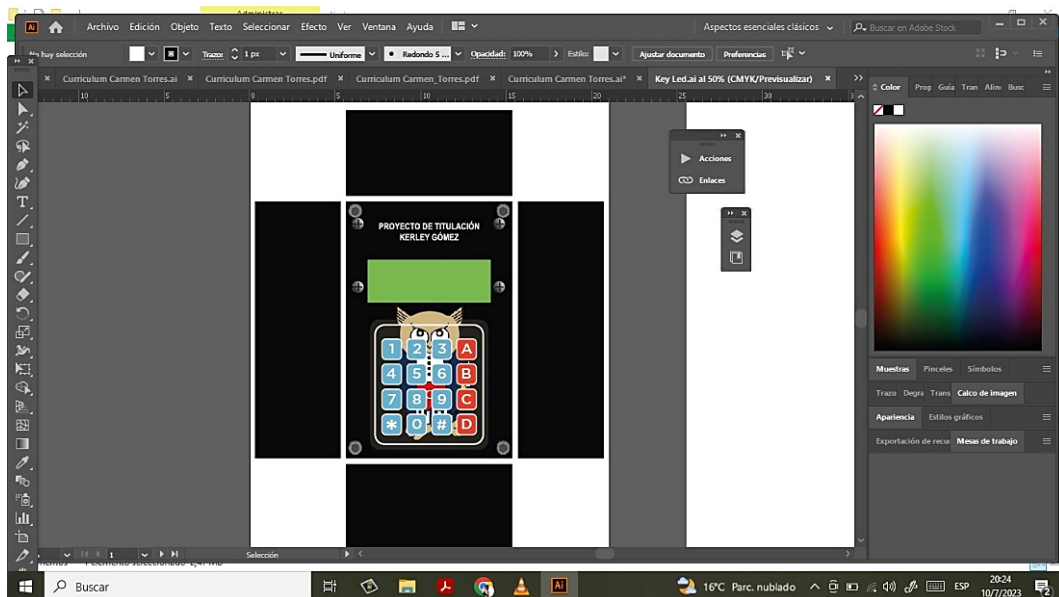
Las dimensiones de la caja permiten que el PCB ocupe el espacio en el interior de una manera correcta, evitando desplazamientos innecesarios, como se muestra en la Figura 3.64.





**Figura 3.64** Caja con el PCB

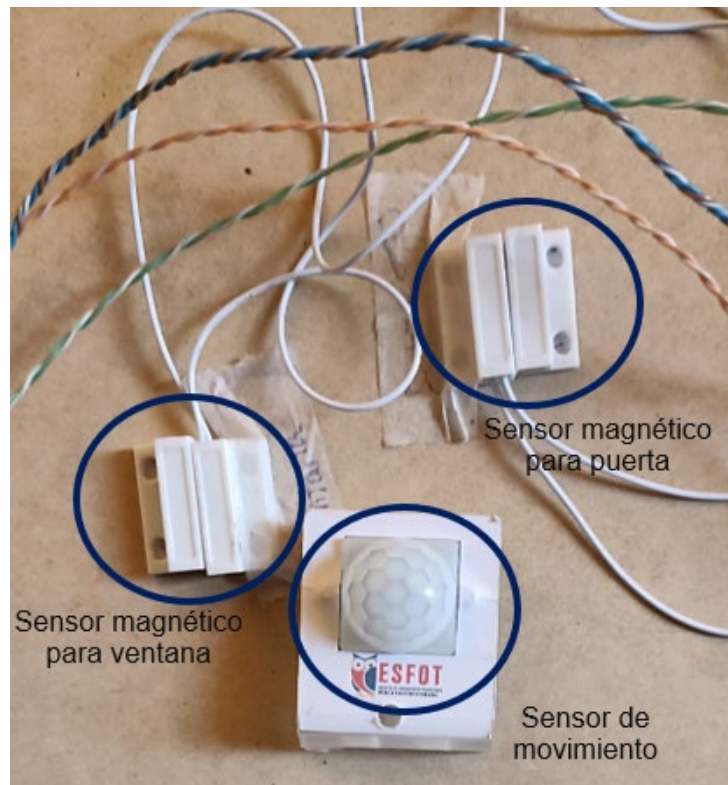
Mediante el *software* de diseño *Adobe Illustrator* se creó una cubierta para la caja, con el objetivo de otorgar una presentación más formal y amigable, como se observa en la Figura 3.65.



**Figura 3.65** Diseño de cubierta

### **Conexión de los sensores**

Los tres sensores para utilizar en el prototipo de la alarma, que se puede observar en la Figura 3.66, mantienen la conexión de manera cableada mediante cable AWG 24.



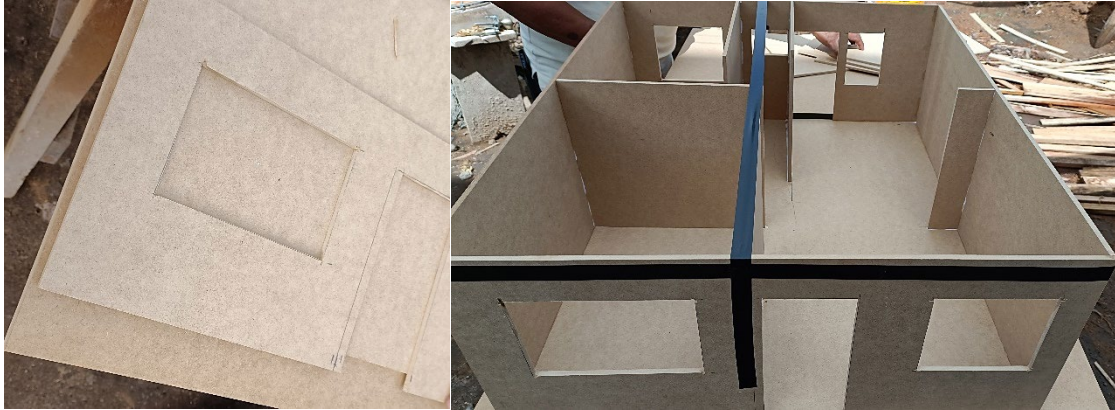
**Figura 3.66** Sensores

La distancia del cableado utiliza una escala de 1:10, ya que esta es la medida que se utilizó en el diseño de la maqueta para el domicilio. Cada cable utilizado tiene una distancia de 120 (cm), representando una distancia de 12 (m).

Como el proyecto es un prototipo, se optó por usar tres sensores ubicados en los sitios más vulnerables. El sensor de movimiento se colocó en el espacio que emula la sala, apuntando hacia la puerta principal, de esta forma detecte movimiento. Un primer sensor magnético se ubicó en la ventana del dormitorio principal para emular la seguridad para las ventanas. Se colocó un segundo sensor magnético en la puerta secundaria de la cocina, para emular seguridad en las puertas.

### **Elaboración de la maqueta residencial**

La maqueta donde se colocó el sistema de alarma se desarrolló con panel de fibra de densidad media (MDF). Este material permite acondicionar los espacios de un domicilio de manera sencilla. El área de 49 (m<sup>2</sup>), datos del Ministerio de Desarrollo Urbano y Vivienda (MIDUVI), que según el informe *Casa para todos* definió el área mínima para un domicilio unifamiliar, contando con estos espacios físicos como mínimo: dos dormitorios, un baño, sala-comedor y cocina [26]. Las medidas para la maqueta se realizaron mediante una escala de 1:10. El resultado parcial durante la elaboración de la maqueta se puede visualizar en la Figura 3.67.



**Figura 3.67** Elaboración de la maqueta

### **Montado del sistema sobre la maqueta**

La caja principal de la alarma, en donde se encuentra la placa con el microcontrolador *ESP32*, se optó por ubicar en el interior del domicilio. La ubicación adecuada para el panel o caja se especifica ser en un sitio de fácil acceso para el usuario y que esté protegido lo más posible a la manipulación de extraños [27]. Debido a que la utilización principal del sistema de alarma se fundamenta en el control mediante *Telegram* y una aplicación mediante *Internet*. Se optó por colocar la caja en el dormitorio principal, ya que el manejo mediante el teclado será una opción si no existiese conexión a *Internet*, algo que el usuario debe garantizar. En la Figura 3.68 se encuentra la caja en el dormitorio principal, también se observa la salida de cables, de los sensores, por la parte superior guiados por una canaleta y por la parte inferior el cable de alimentación.



**Figura 3.68** Ubicación de la caja de alarma



El sensor magnético correspondiente para la puerta se ubicó sobre la puerta de la cocina. En la Figura 3.69 se observa la ubicación del sensor, el cableado llega hasta la cocina desde la caja principal por medio de canaletas.



**Figura 3.69** Sensor magnético para la puerta

El segundo sensor magnético, que corresponde a la ventana, se colocó en la ventana del dormitorio principal. En la Figura 3.70 se observar el sitio final para el sensor, de similar manera al sensor anterior, el cableado llega hasta el punto mediante canaletas.



**Figura 3.70** Sensor magnético para la ventana



Finalmente, el sensor de movimiento se colocó en la esquina superior de la sala, diagonal a la entrada principal al domicilio. La Figura 3.71 muestra la ubicación del *PIR*, que también tiene el LED rojo para mostrar la detección de movimiento, como los otros sensores, el cableado se hizo por medio de canaletas desde la caja principal de la alarma.



**Figura 3.71** Sensor de movimiento

El resultado final de la implementación, de la parte física, del prototipo de alarma sobre la maqueta de una residencia unifamiliar de 49 (m<sup>2</sup>), se puede visualizar en la Figura 3.72. Cada área está identificada con la finalidad de localizar cada sensor, se observa la caja principal de la alarma colocada en la pared del dormitorio principal, el sensor magnético de la ventana se ubica también en el dormitorio.

El sensor de movimiento se ubica en la sala de la residencia, apuntando a la entrada principal, el segundo sensor magnético se ubica en la puerta de entrada y salida de la cocina, hacia un patio trasero. Todo el cableado se realiza con canaletas, para enrutarlo de manera correcta y ordenada, ofreciendo protección a los cables [28].



Figura 3.72 Maqueta sistema de alarma

### Ejecución del bot de Telegram

Con el sistema físicamente implementado, se procedió a iniciar el bot Alarma\_casa\_bot de Telegram. Al ser un prototipo la utilización del bot creado se ha limitado a un solo usuario, el cual se identifica mediante el ID que se obtuvo en el paso Creación del bot de Telegram. El ID se utilizó en el código, con lo cual la comunicación con el sistema de alarma será exclusiva del usuario relacionado al ID. Esto se realizó con la finalidad de evitar que la alarma pueda ser controlada por cualquier persona que encuentre el bot en Telegram.

Para localizar el bot creado, en el buscador de la aplicación de Telegram se debe digitar el nombre del bot, para el presente proyecto el bot fue Alarma\_casa\_bot. En la Figura 3.73 se muestra el resultado de la búsqueda del bot.

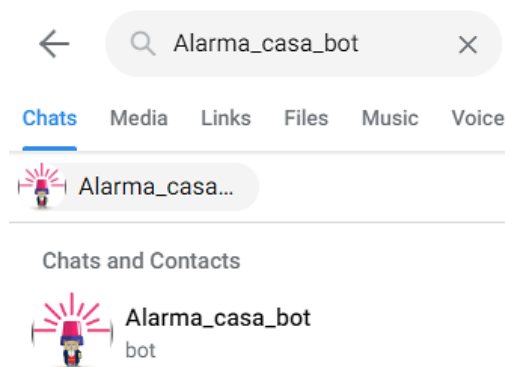
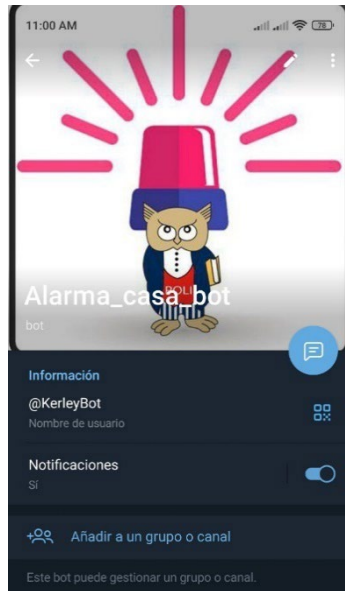


Figura 3.73 Bot de alarma en Telegram

Localizado el *bot*, se puede visualizar la información en donde destaca el usuario que lo creó, tal como muestra la Figura 3.74. De esta forma se puede iniciar el uso del *bot* junto al sistema de alarma creado mediante los comandos que se especificaron en la etapa de Desarrollo del código en *ESP32*.



**Figura 3.74** Información del *bot* en *Telegram*

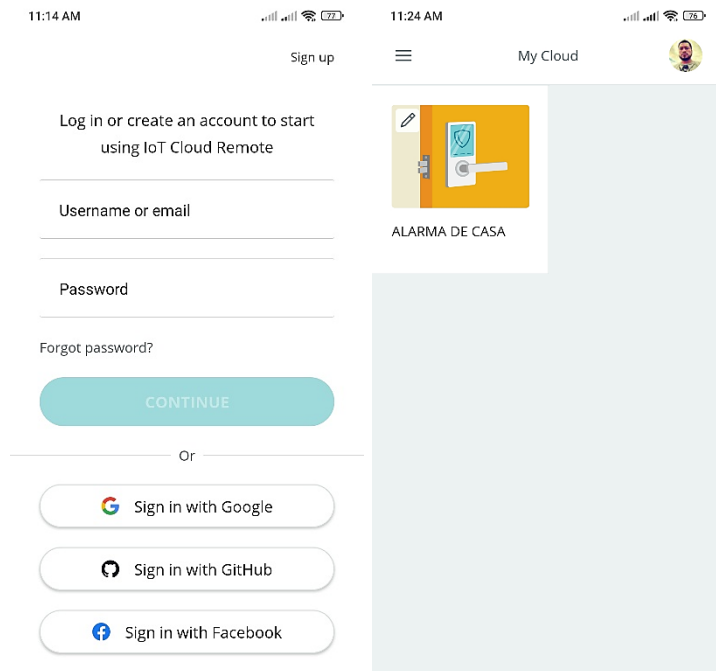
### **Instalación y acceso a la aplicación en *Android***

Para instalar la aplicación desarrollada en la plataforma *Arduino IoT Cloud* es necesario acceder a la *Play Store* de los equipos *Android*. En la sección de búsqueda se digitó el nombre de la aplicación *Arduino IoT Cloud*, tal como se muestra en la Figura 3.75. Localizada la aplicación se instalará en el terminal del usuario que usará el sistema de alarma desarrollado.



**Figura 3.75** *Arduino IoT Cloud* en *Play Store*

Una vez dentro de la aplicación de *Arduino IoT Cloud* se solicitará al iniciar las credenciales del usuario. Estas credenciales deben ser las mismas utilizadas en la etapa Creación de la aplicación, estas credenciales también son las utilizadas durante el desarrollo del Código en el *Web Editor*. Una vez dentro con las credenciales se visualizará el nombre del *dashboard* creado, en este caso para el presente proyecto tiene el nombre de **Alarma de Casa** como se observa en la Figura 3.76.



**Figura 3.76** Aplicación en *Arduino IoT Cloud*

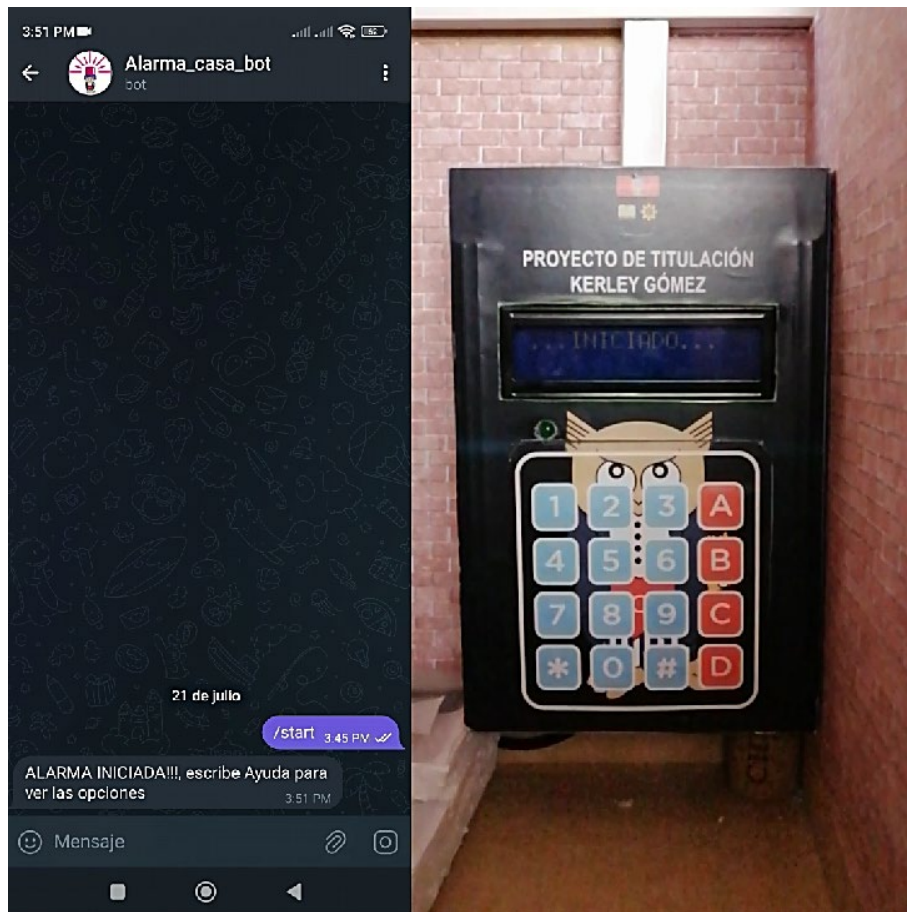
### 3.5 Realización de pruebas de funcionamiento del prototipo

Al finalizar con las etapas de identificación de requerimientos, selección de *hardware* y *software*, diseño e implementación del prototipo de la alarma, se llevaron a cabo las pruebas de funcionamiento sobre la maqueta, que certifiquen el cumplimiento de los requerimientos establecidos

#### Inicio del sistema de alarma

Como se especificó en la etapa de Ejecución del *bot* de *Telegram* e Instalación y acceso a la aplicación en *Android*, el usuario debe contar con las aplicaciones instaladas en su dispositivo móvil. Para iniciar el sistema debe ser energizado mediante el cargador elegido, cuando esto ocurre se establecerá la comunicación con la red *Wi-fi*, especificando que al ser un prototipo las credenciales de acceso se colocaron de manera permanente en el código y que el usuario debe garantizar que la red esté funcionando y con acceso a *Internet*.

Al iniciar en la pantalla se muestra el mensaje de “INICIADO” y de manera inmediata se envía una notificación o mensaje al bot Alarma\_casa\_bot con el mensaje “ALARMA INICIADA!!!, escribe Ayuda para ver las opciones”, todo lo anterior se observa en la Figura 3.77. De esta manera el sistema está listo para su control.

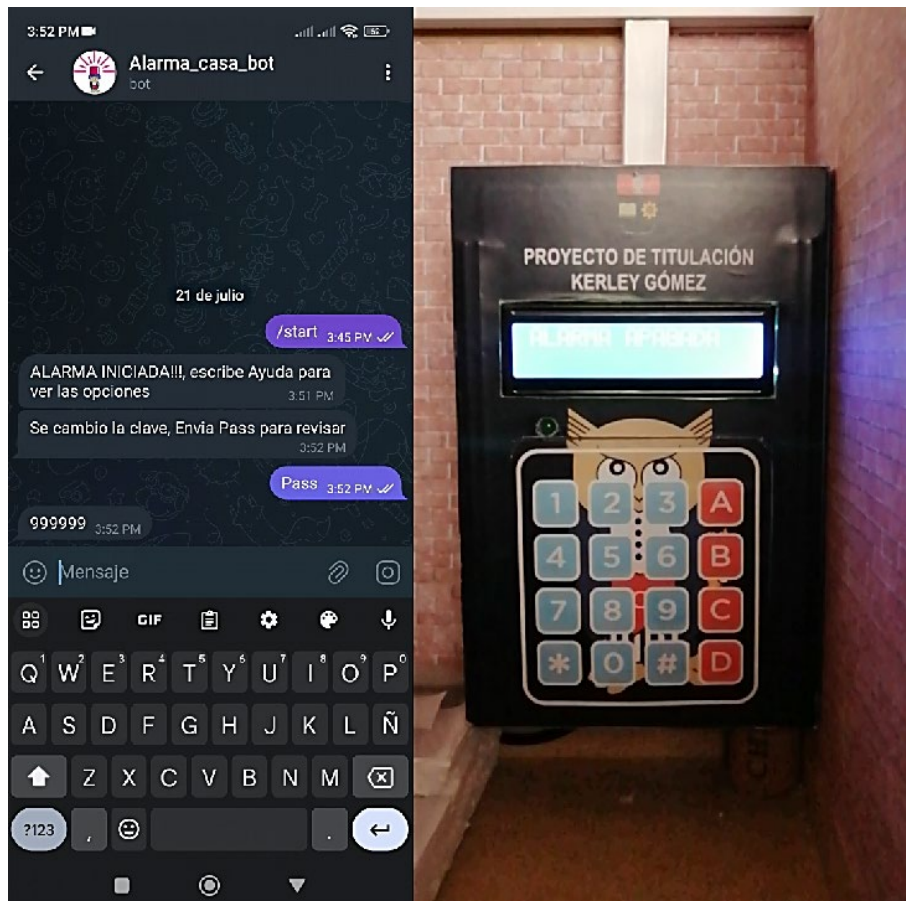


**Figura 3.77** Inicio del Sistema de Alarma

### **Cambio de clave de encendido o apagado**

El sistema cuenta con una clave por defecto, la cual se definió en “ABCD45”, y será el usuario quien decida si cambiarla o no. Por seguridad se recomienda cambiar la clave y que sea de uso personal. Para poder cambiar la clave se debe presionar en el teclado físico la tecla asterisco (\*), al hacerlo se solicita que se ingrese la clave actual, si esta es correcta se debe digitar la nueva clave, la misma que debe tener una extensión de 6 dígitos entre números y letras. Al finalizar el ingreso de la nueva clave se notifica a él *bot* del cambio mediante el mensaje “Se cambió la clave, Envía *Pass* para revisar”, cuando se escribe *Pass* el *bot* devolverá la clave nueva. Todo esto lo realizado se muestra en la Figura 3.78.

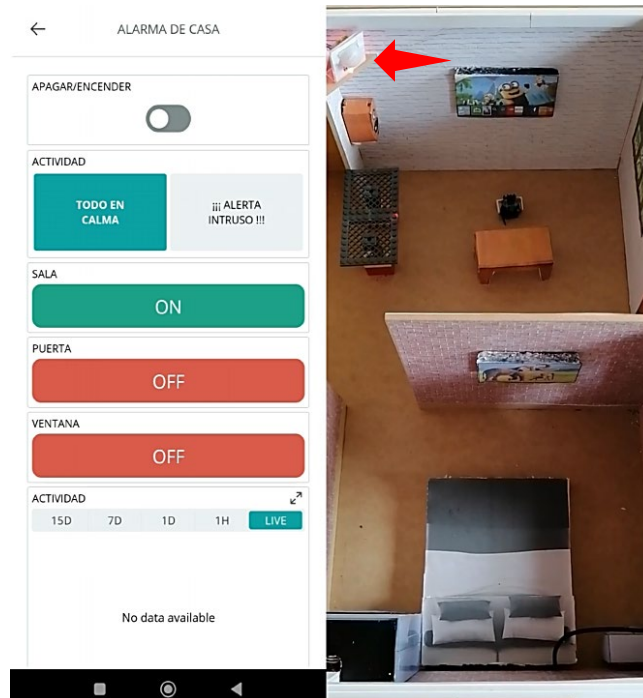




**Figura 3.78** Cambio de clave

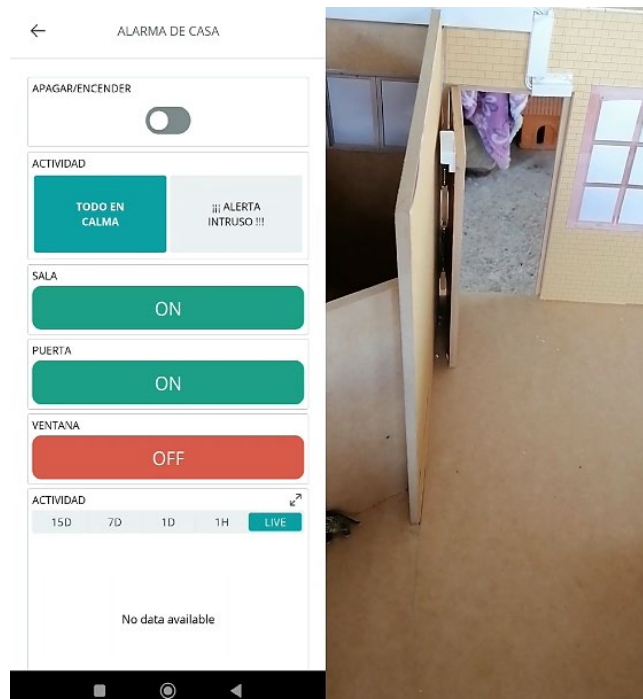
### **Revisión de sensores**

Cuando el sistema ha iniciado, pero no se ha encendido, se puede revisar el estado de los sensores por medio de la aplicación en *Arduino IoT Cloud* y el *bot de Telegram*. Desde la aplicación se observa tres *widgets* que se comunican con los sensores, cuando el *PIR* detecta presencia se muestra el *widget* sala en color verde con la palabra *on*, de manera física se observa el *LED* rojo que está encendido debido a la detección, cada vez que el sensor de movimiento detecte presencia ocurre lo antes mencionado, tal como muestra la Figura 3.79.



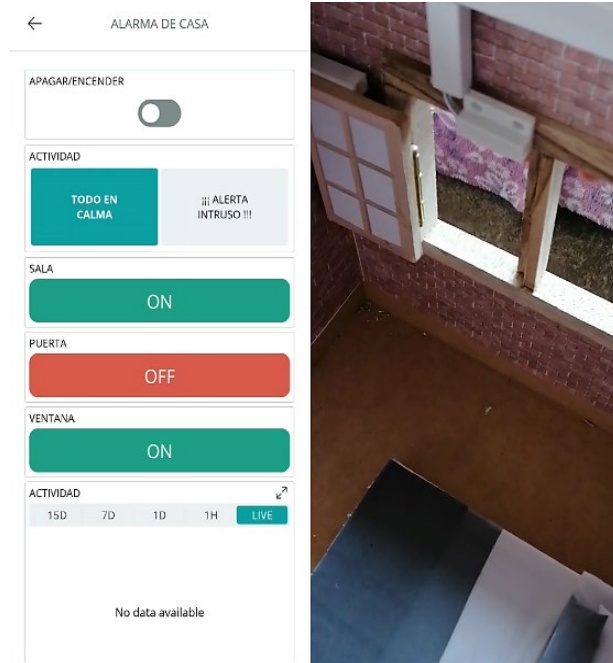
**Figura 3.79** Estado del sensor de movimiento desde la aplicación

Cuando se abre la puerta de la cocina, el sensor magnético se separa, mostrando en la aplicación el *widget* puerta en color verde con la palabra *on*. Como muestra la Figura 3.80



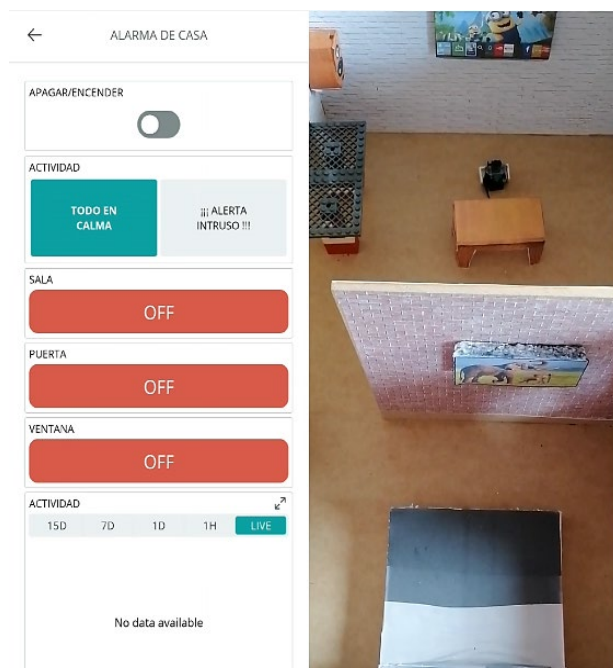
**Figura 3.80** Estado del sensor magnético puerta desde la aplicación

Cuando la ventana del dormitorio se abre, el sensor magnético se separa, mostrando el *widget* ventana de color verde con la palabra *on*, como se observa en la Figura 3.81.



**Figura 3.81** Estado del sensor magnético ventana desde la aplicación

Cuando los sensores se encuentran en calma, se puede visualizar que los *widgets* se colocan en color rojo con la palabra *off*. Como se muestra en la Figura 3.82.

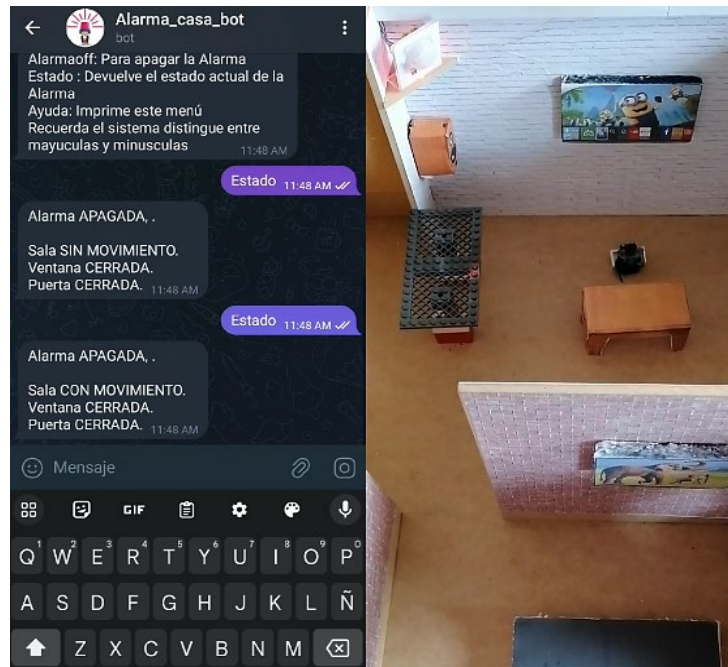


**Figura 3.82** Estado de los sensores desde la aplicación

Desde el *bot* el de *Telegram* se revisa el estado de los sensores, para esto se digita ayuda y se visualiza cual es el mensaje que permite revisar los sensores, mensaje que se definió como "Estado". Al escribir el mensaje se observa el estado de la alarma, que está apagada y cuál es el estado de cada sensor. Al detectarse presencia por parte del

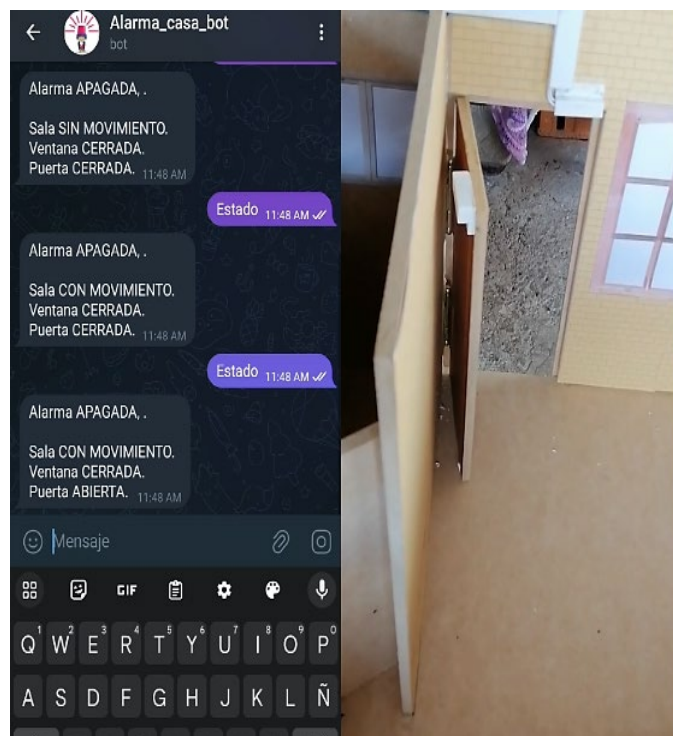


PIR, se enciende el *LED* rojo, y al escribir Estado en el *bot* se recibe como respuesta que la alarma está apagada y que la sala esta con movimiento. Tal como se visualiza en la Figura 3.83.



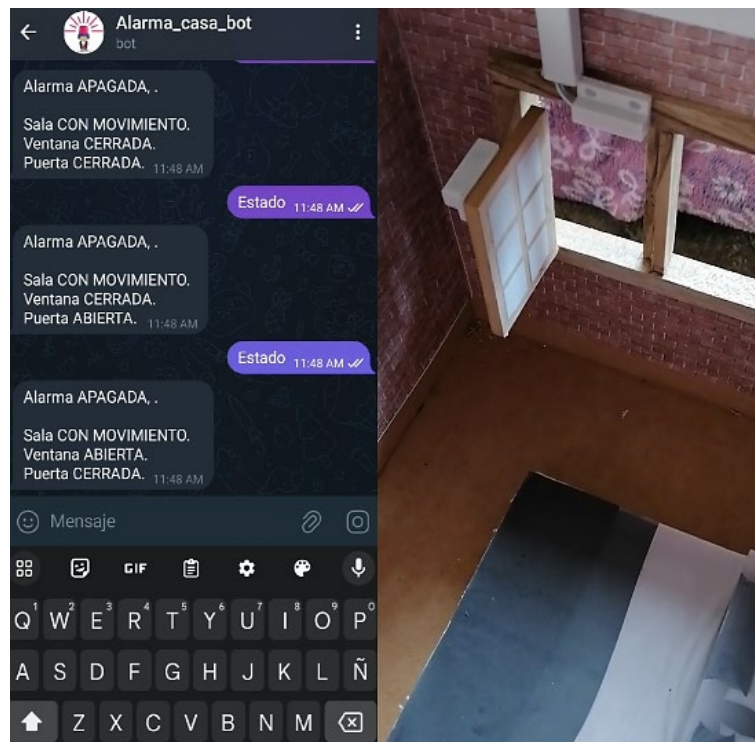
**Figura 3.83** Estado del sensor de movimiento desde el *bot* de *Telegram*

Al abrir la puerta de la cocina y escribir estado desde el *bot* se recibe que la puerta está abierta, como se puede visualizar en la Figura 3.84.



**Figura 3.84** Estado del sensor magnético puerta desde el *bot* de *Telegram*

Finalmente, al abrir la ventana del dormitorio y revisar con el mensaje Estado, se puede observar que la ventana está abierta, como se muestra en la Figura 3.85.



**Figura 3.85** Estado del sensor magnético ventana desde el *bot* de *Telegram*

Con las pruebas anteriores se verifica que es posible revisar los sensores cuando el sistema está apagado.

### **Encendido y apagado de la alarma**

Para encender o apagar la alarma se puede hacerlo de tres maneras detalladas a continuación. Utilizando la aplicación, mediante el *switch*, se coloca en estado de encender la alarma, tomando un color verde, al encender la alarma en la pantalla se observa el mensaje de “ALARMA ENCENDIDA”, como se visualiza en la Figura 3.86, como el sistema mantiene la comunicación también con *Telegram*, se recibirá una notificación alertando al usuario del encendido de la alarma, como muestra la Figura 3.87.

El proceso de encendido total tiene una duración de 30 segundos, tiempo en el cual se emite una serie de pitidos por parte del *buzzer* interno de la caja contenedora del circuito principal. Este tiempo permite que el usuario pueda salir de su residencia sin que algún sensor lo detecte, finalizado este tiempo la alarma está encendida y para saberlo se coloca un *LED* verde con iluminación tal como se observa en la Figura 3.88.

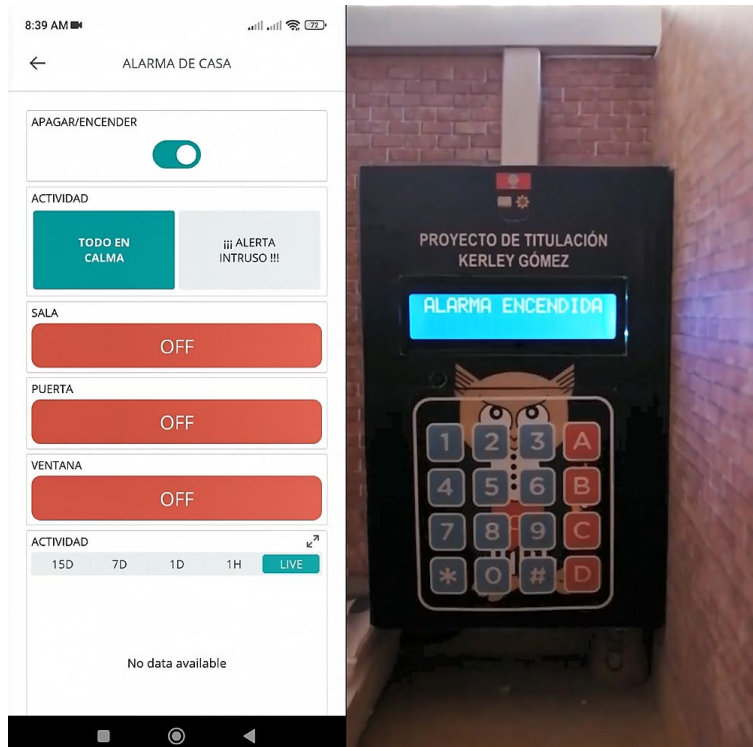


Figura 3.86 Encendido de la alarma desde la aplicación

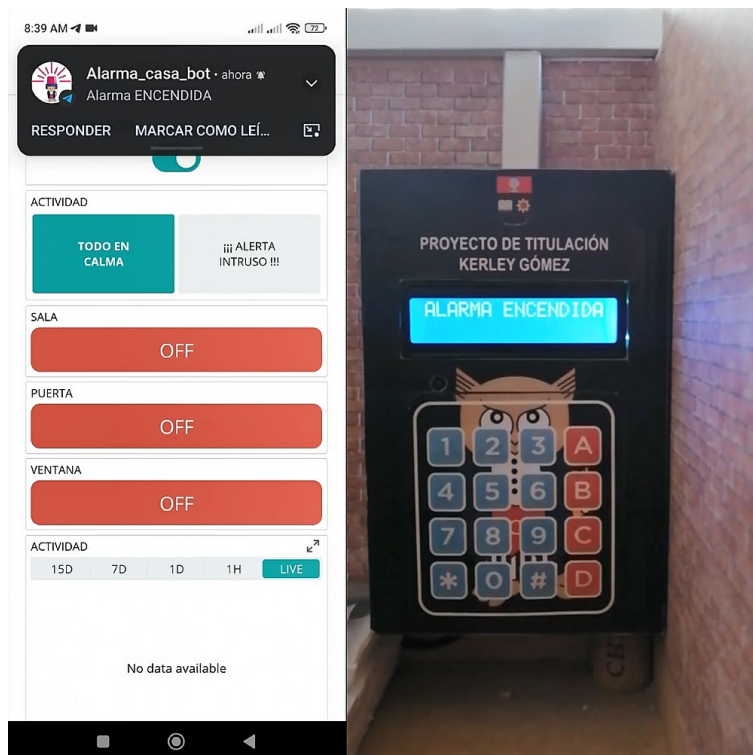


Figura 3.87 Notificación al bot del encendido de la alarma

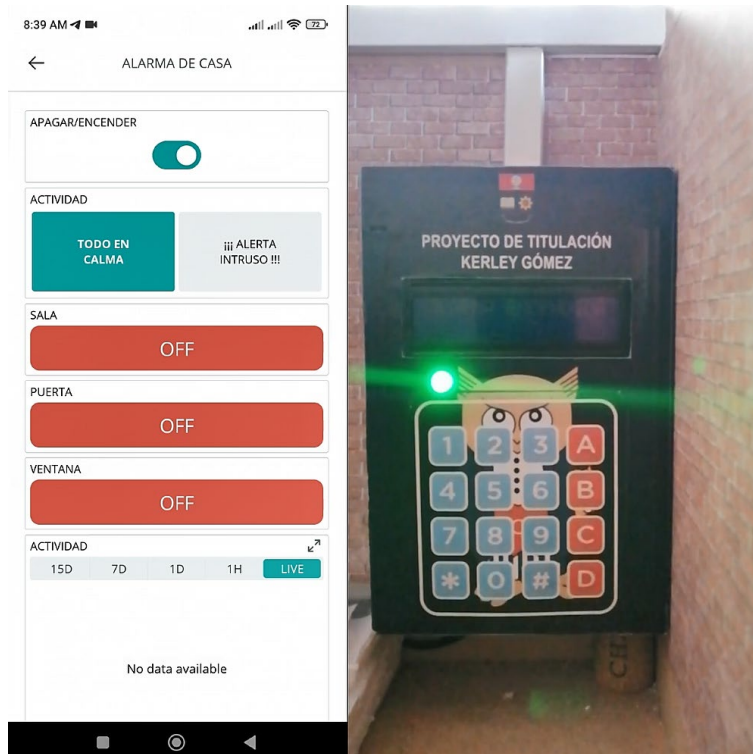


Figura 3.88 Alarma encendida

Para apagar la alarma, el *switch* virtual de la aplicación se coloca en apagar, con esta acción el sistema se apaga, mostrando el mensaje de “ALARMA APAGADA” en el LCD, apagando el LED verde y notificando al *bot* Alarma\_casa\_bot del apagado de la alarma, como se muestra en la Figura 3.89.

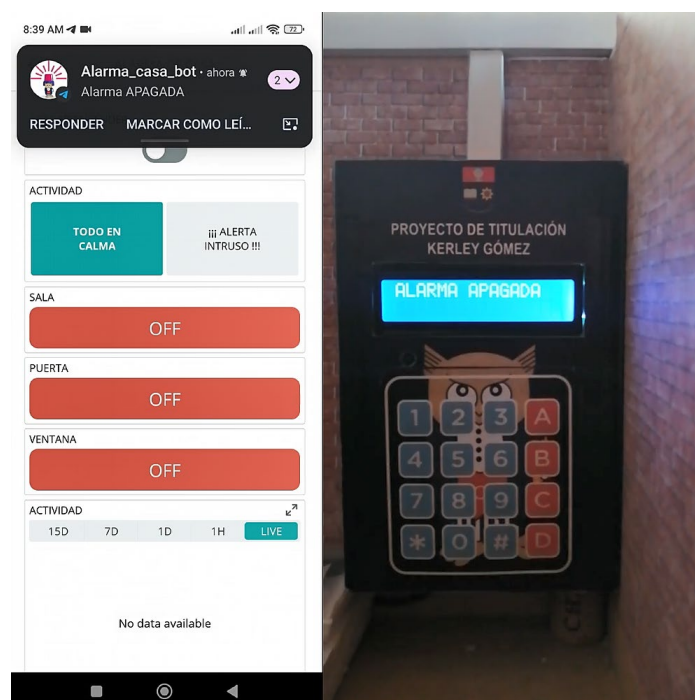
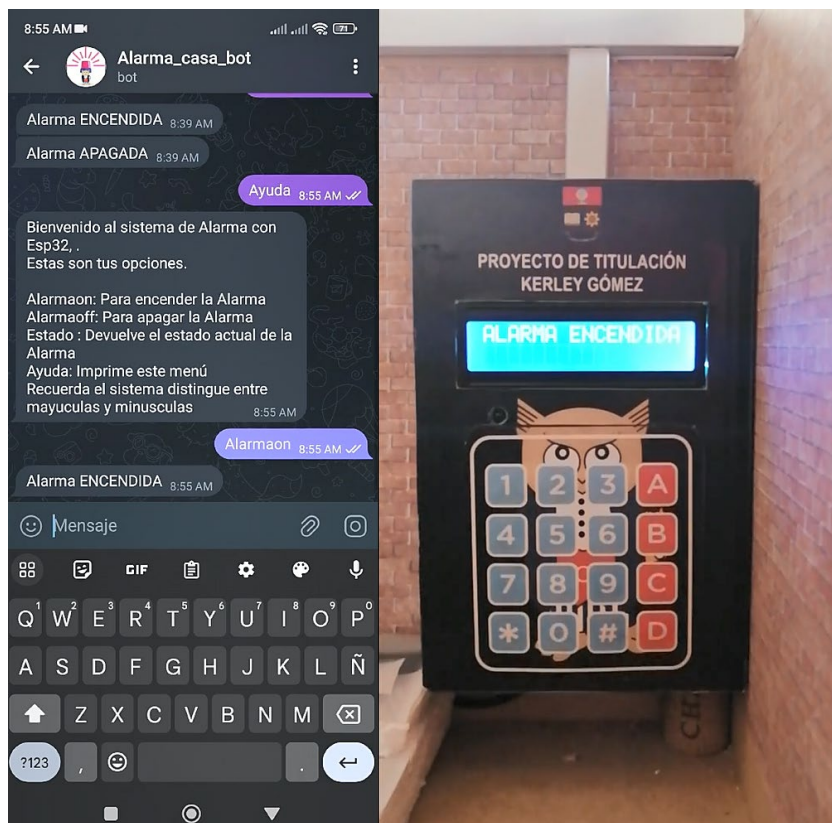


Figura 3.89 Apagado de la alarma desde el bot Alarma\_casa\_bot

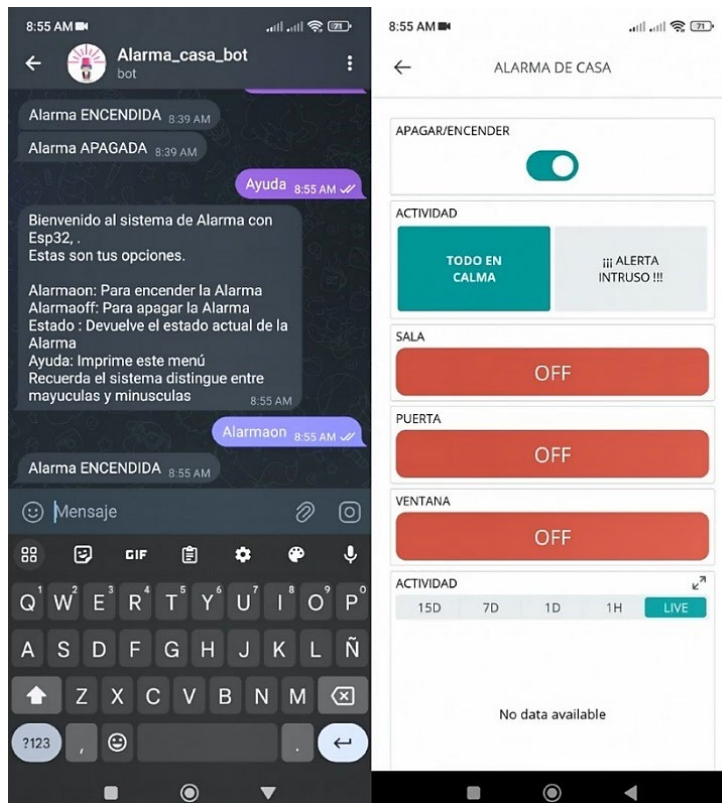


Desde el *bot* Alarma\_casa\_bot, en *Telegram*, se escribe “Ayuda” con la finalidad de saber qué mensaje o comando es el que permite encender o apagar la alarma. Para encender se observa que se debe escribir “Alarmaon”, al hacerlo se recibe como respuesta de “Alarma ENCENDIDA”, en el *LCD* se imprime el mensaje de “ALARMA ENCENDIDA”, tal como se muestra en la Figura 3.90. Similar al encendido desde la aplicación, se tendrá 30 segundos acompañados de pitidos constantes antes que el sistema se encienda por completo, después de este tiempo se encenderá el *LED* verde para avisar al usuario que este encendido la alarma.



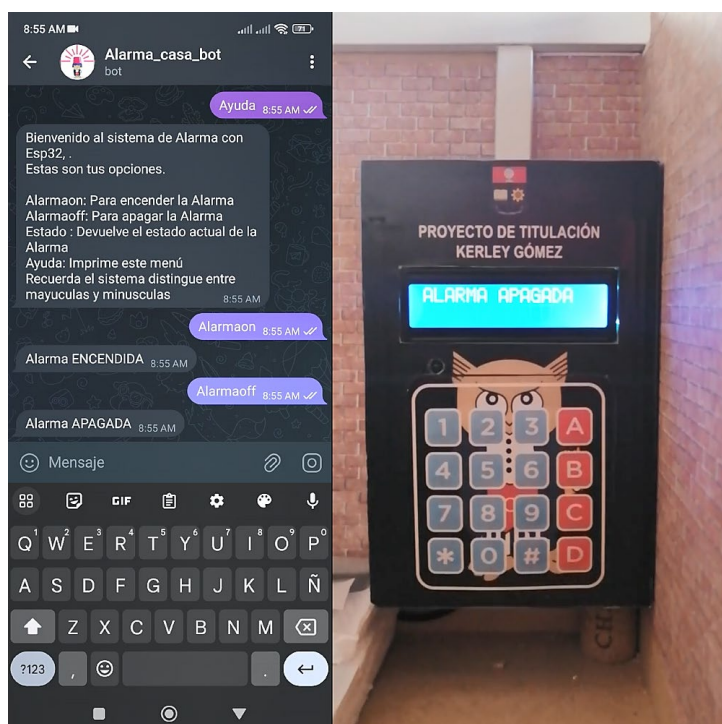
**Figura 3.90** Encendido de la alarma desde el *bot* Alarma\_casa\_bot

Como el sistema mantiene la comunicación con la aplicación de *Arduino IoT Cloud*, el encendido de la alarma se sincroniza de manera inmediata en la aplicación, como se observa en la Figura 3.91, de esta forma se evita conflictos al controlar la alarma desde cualquiera de las plataformas y tener que sincronizarlos de manera manual.



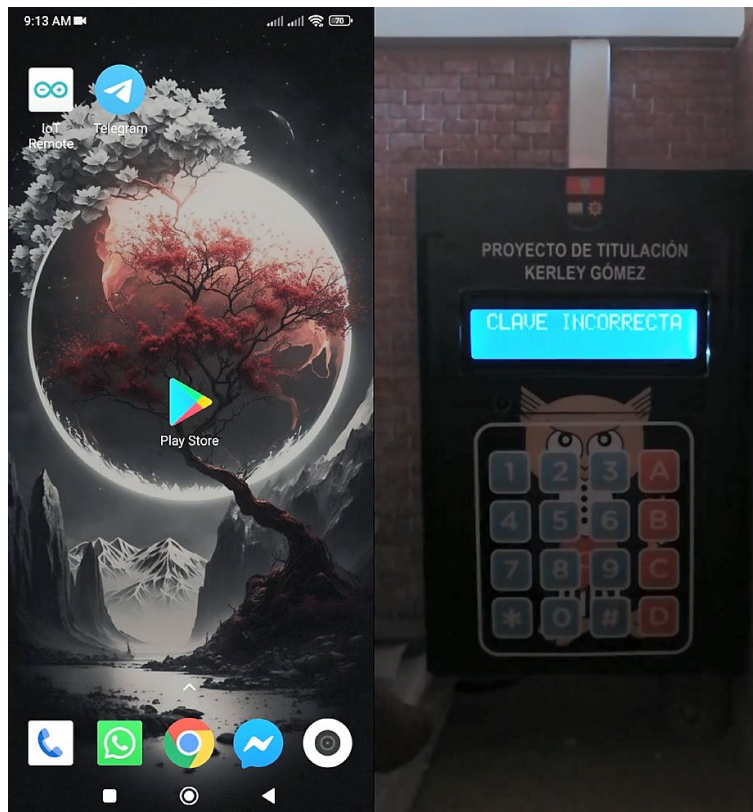
**Figura 3.91** Sincronización del bot y la aplicación

Para apagar la alarma desde el bot, se envía el mensaje “Alarmaoff”, recibiendo como respuesta “Alarma APAGADA” e imprimiendo el mensaje “ALARMA APAGADA” en el LCD, como se puede observar en la Figura 3.92.



**Figura 3.92** Apagado de la alarma desde el bot Alarma\_casa\_bot

Finalmente, la última forma de poder encender o apagar la alarma es mediante la utilización del teclado físico del sistema, para lo cual se presiona la tecla numeral (#), en donde se solicita en el *LCD* que se ingrese la clave, si se ingresa una clave incorrecta se muestra en pantalla el mensaje “CLAVE INCORRECTA”, tal como se muestra en la Figura 3.93.

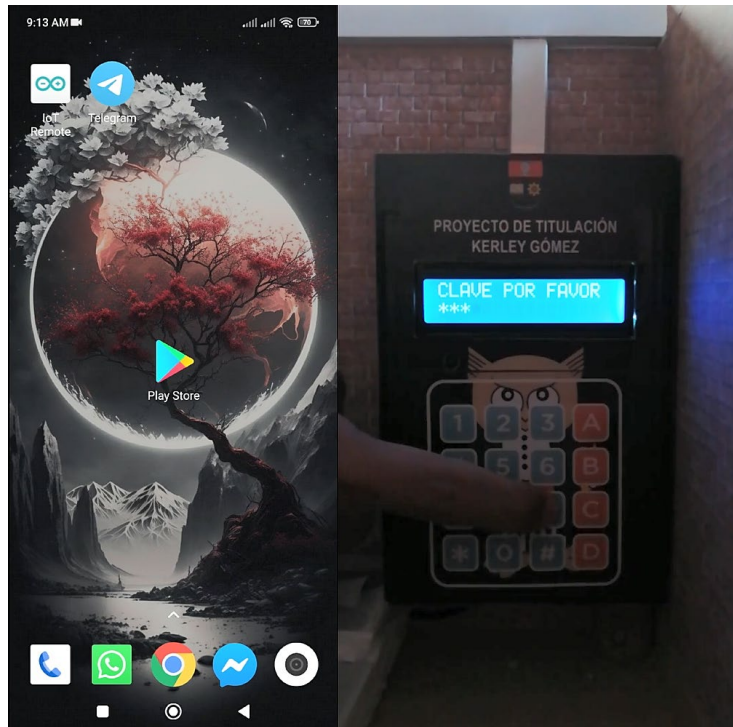


**Figura 3.93** Ingreso de clave incorrecta

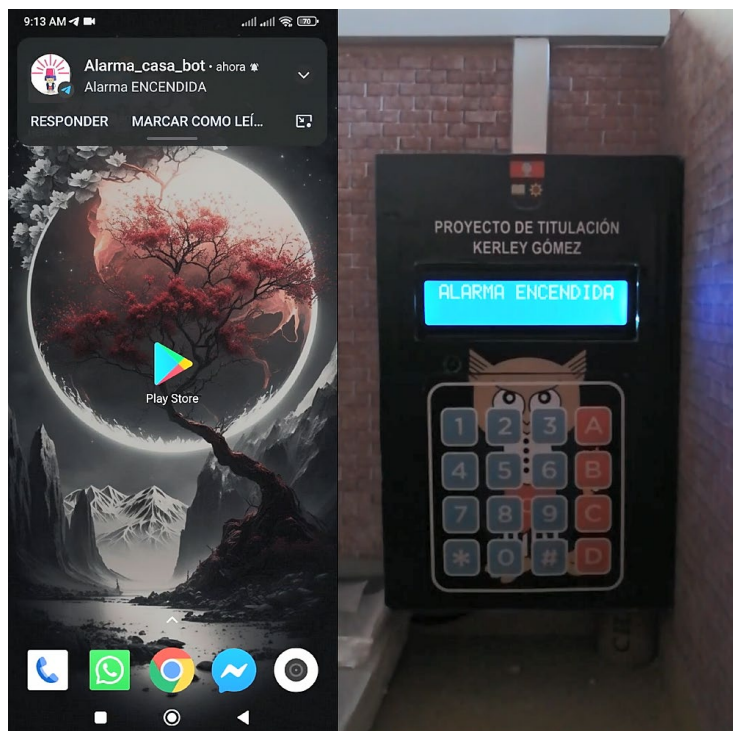
Para volver a ingresar la clave se presiona nuevamente la tecla numeral (#), se solicita que se ingrese la clave, al presionar los 6 caracteres se observa en la Figura 3.94, en la pantalla, que se encuentran ocultos, protegiendo de esta manera que alguien cercano pueda visualizar cual es la clave.

Al finalizar el ingreso se imprime en pantalla el mensaje “ALARMA ENCENDIDA” y también se notifica al *bot* Alarma\_casa\_bot, como se muestra en la Figura 3.95, del encendido de la alarma. Además, se actualiza el estado del *switch* en la aplicación de *Arduino IoT Cloud*. Como en los casos anteriores existe 30 segundos para el encendido total de la alarma, tiempo que ayuda a que el usuario salga de su residencia sin activar algún sensor, en especial si se enciende la alarma de manera manual desde el teclado físico del sistema.





**Figura 3.94** Ingreso de la clave para encender o apagar la alarma

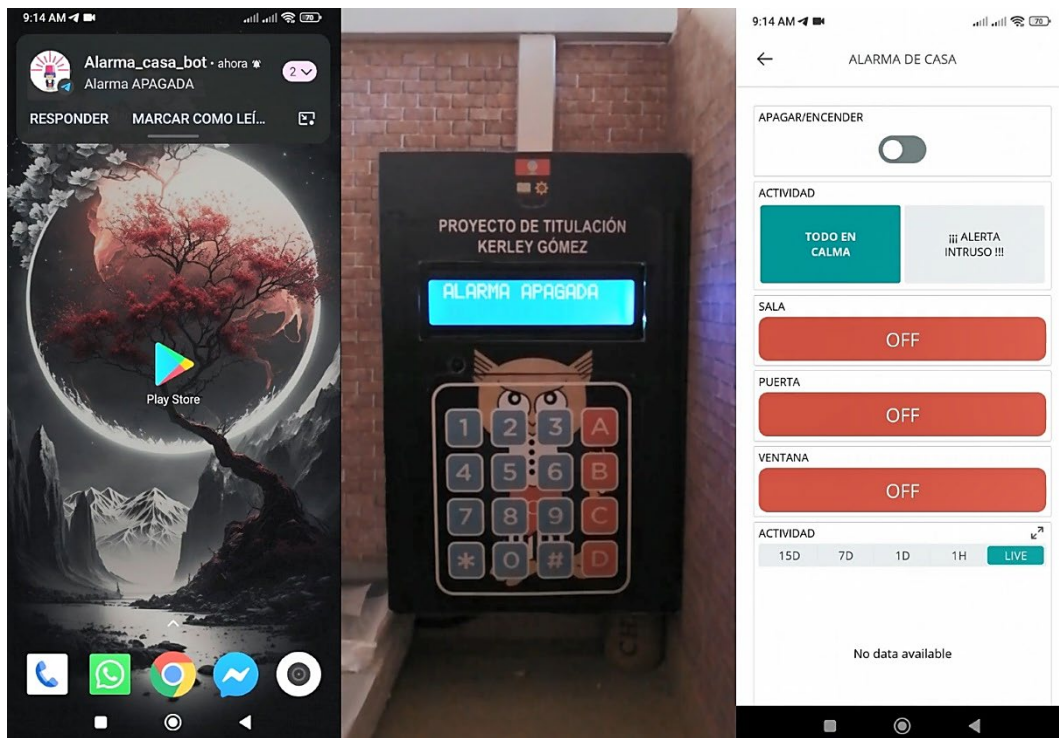


**Figura 3.95** Encendido de la alarma desde el teclado

Para el apagado de la alarma desde el teclado se presiona nuevamente la tecla numeral (#) y se ingresa la clave, al ingresar de manera correcta se muestra en pantalla el mensaje “ALARMA APAGADA”, se apaga el *LED* verde y se notifica al *bot* del apagado



de la alarma, así como actualizar el estado en la aplicación de *Arduino IoT Cloud*, tal como se observa en la Figura 3.96.



**Figura 3.96** Apagado de la alarma desde el teclado

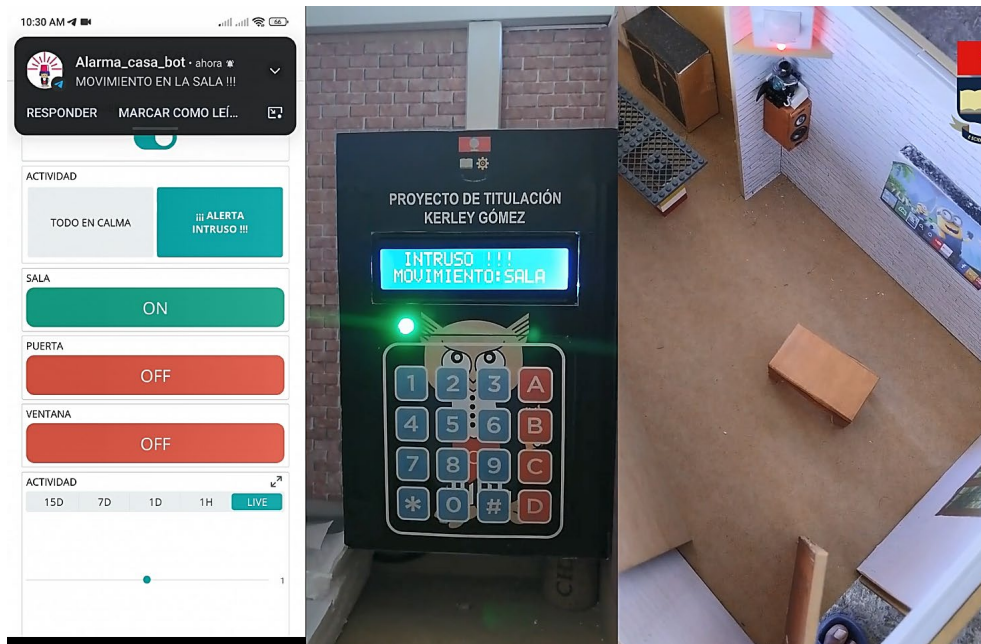
Las pruebas realizadas para el encendido y apagado de la alarma permiten comprobar que dichas acciones se las puede realizar sin ningún inconveniente por cualquier forma, sea esta por la aplicación, el bot o el teclado físico.

### **Detección de movimiento mediante el *PIR***

Cuando el sistema este encendido y el sensor de movimiento *PIR* detecta algún intruso en el área de la sala, como se muestra en la Figura 3.97, el *LED* de color rojo que acompaña al sensor se enciende, de manera inmediata en la pantalla de la caja de la alarma se imprime el mensaje de alerta junto con el área comprometida, en este caso “INTRUSO”, “MOVIMIENTO: SALA”. Se emite un sonido constante por parte del *buzzer*, el cual sirve de alerta, y que no se detendrá hasta realizar el apagado de la alarma, ya sea por la aplicación, el *bot* o el teclado, como se mostró en el apartado Encendido y apagado de la alarma.

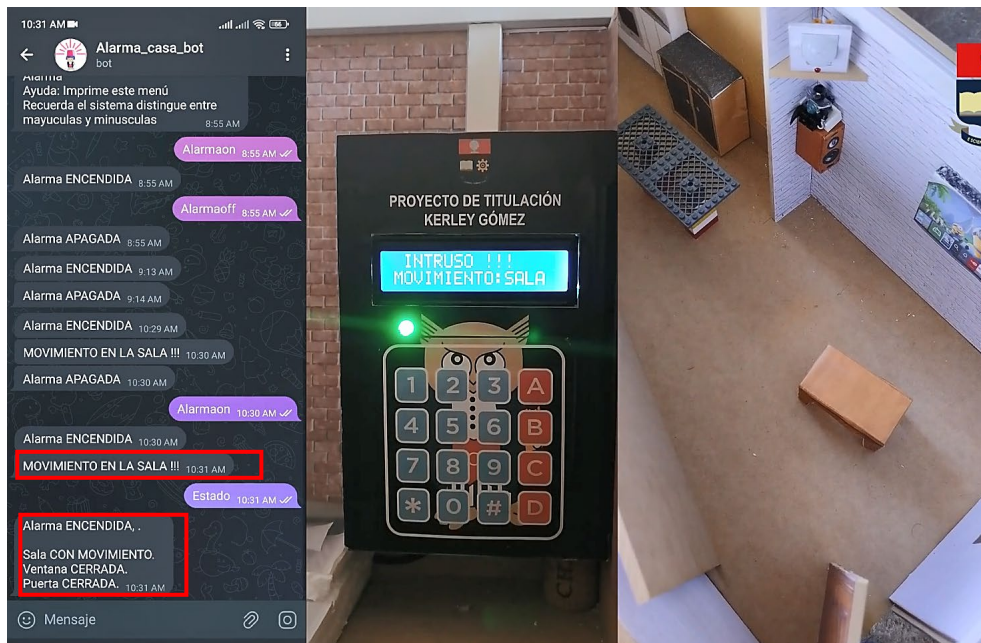
Durante la detección se notifica al *bot* Alarma\_casa\_bot mediante una notificación, con el mensaje “MOVIMIENTO EN LA SALA”, también, en la aplicación se coloca el *widget* de actividad en “ALERTA INTRUSO”, el *widget* de sala se coloca en verde con la palabra

on y el *chart* de actividad se coloca en 1. El *chart* sirve para tener un registro de la detección del sistema.



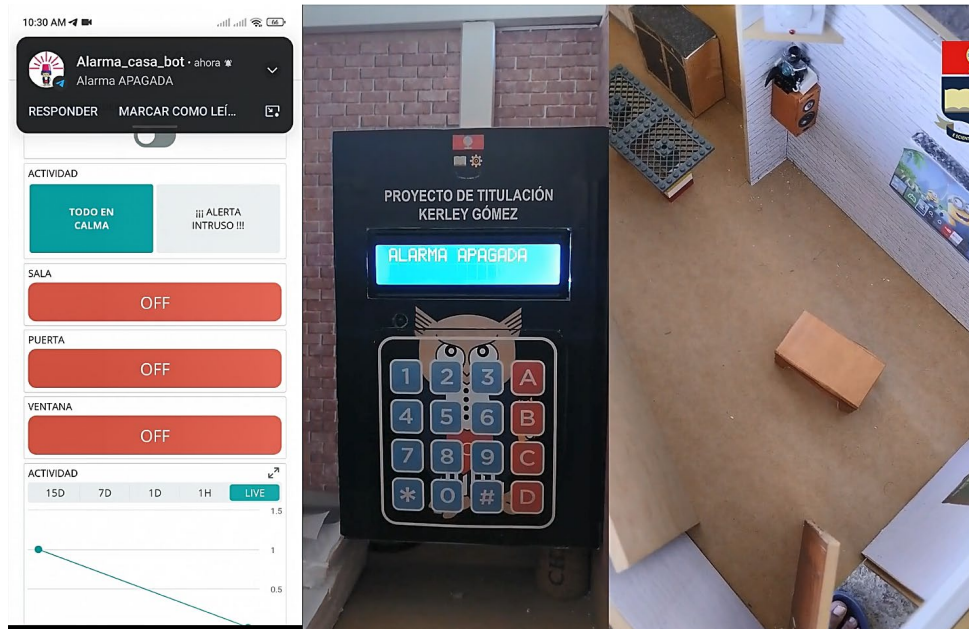
**Figura 3.97** Detección mediante el *PIR* - aplicación

Si se revisa el *bot* se puede verificar que el encendido se realizó desde *Telegram*, también la recepción del mensaje debido a la actividad en la sala y al enviar el mensaje “Estado” se recibe que la alarma está encendida y que la sala esta con movimiento, tal como se observa en la Figura 3.98. Aunque el intruso ya no está en el rango de detección, el sistema sigue alertando sobre la detección ocurrida al inicio.



**Figura 3.98** Detección mediante el *PIR* - Alarma\_casa\_bot

Para detener el sonido de alerta emitido por el sistema, se debe apagar la alarma por cualquier medio, en este caso se lo hace por medio de la aplicación, si el sensor de movimiento no tiene presencia, se coloca el *widget* sala en *OFF*. El *chart* de actividad se coloca en 0 y el *widget* actividad se coloca en el mensaje “TODO EN CALMA” de color verde, tal como se muestra en la Figura 3.99.



**Figura 3.99** Detención de la alerta de la alarma

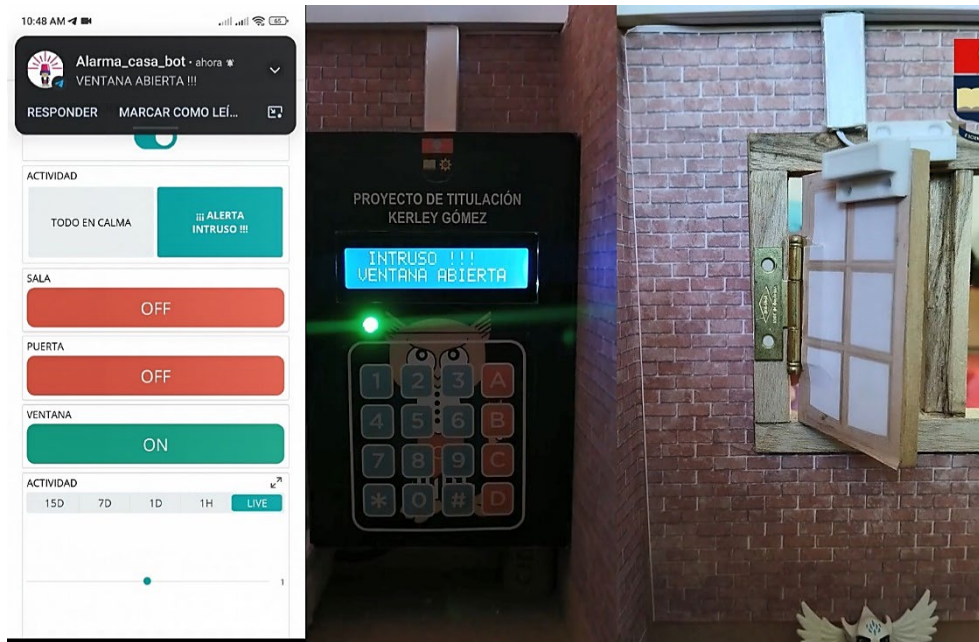
De esta forma se prueba que el sensor *PIR* funciona correctamente durante la detección de intrusos y que el sistema recibe la alerta por parte del *bot*, así como los cambios en los estados de los *widgets* de la aplicación. También la operatividad por parte del *buzzer* y el *LCD* en la alerta es correcta.

### **Detección mediante el sensor magnético de la ventana**

El sensor magnético colocado en la ventana, del dormitorio principal, alerta de la actividad en el momento que la ventana se abre, separándose una distancia de 15 a 25 (mm). Cuando un intruso ingresa por la ventana, en la pantalla de la caja ubicada en el dormitorio se imprime el mensaje “INTRUSO!!!”, “VENTANA ABIERTA”, se emite el sonido de alerta por parte del *buzzer*. Al revisar la aplicación de *Arduino IoT Cloud* se observa como el *widget* ventana se coloca en *ON* de color verde, el *chart* de actividad se coloca en 1 y el *widget* de actividad se coloca en verde con el mensaje “!!! ALERTA INTRUSO !!!”.

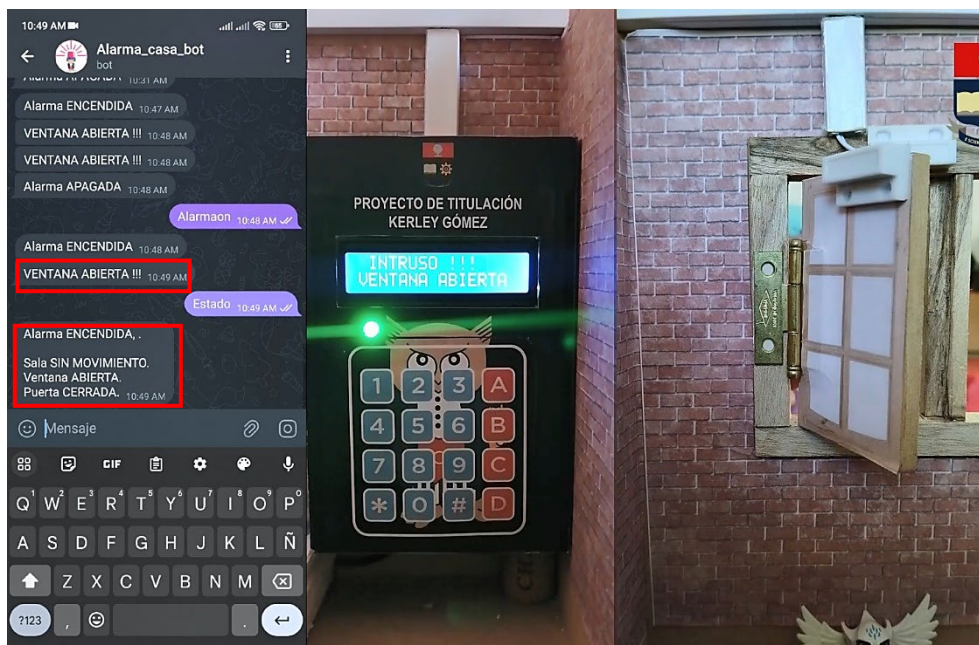
Respecto al *bot* Alarma\_casa\_bot se observa la llegada de la notificación, con el mensaje de “VENTANA ABIERTA !!!”. Es decir, el sistema esta sincronizado, tal como se visualiza en la Figura 3.100.





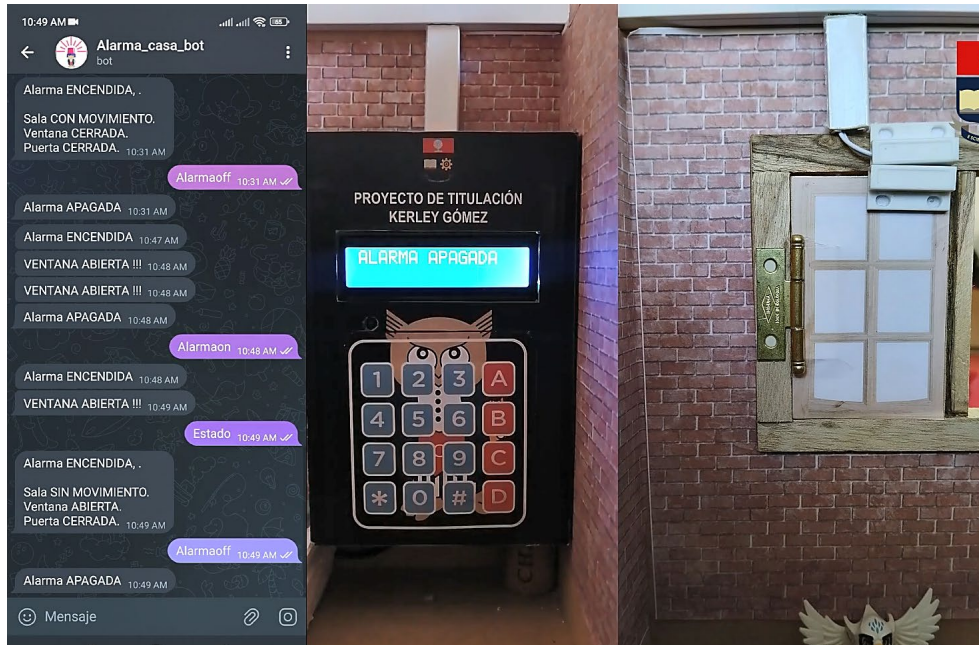
**Figura 3.100** Detección mediante el sensor magnético de la ventana – aplicación

En la Figura 3.101 se observa las acciones realizadas en el bot, en esta prueba el encendido se realiza desde *Telegram*. Cuando la ventana se ha abierto se recibe la alerta mediante la notificación “VENTANA ABIERTA !!!”. Para verificar que sensor es el comprometido se escribe “Estado”, verificando que la alarma está encendida y la ventana abierta.



**Figura 3.101** Detección mediante el sensor magnético de la ventana – Alarma\_casa\_bot

El detenido de la alerta emitido por el *buzzer* se lo hace al apagar la alarma, en este caso mediante el *bot* Alarma\_casa\_bot, mediante el mensaje “Alarmaoff”, tal como se muestra en la Figura 3.102.



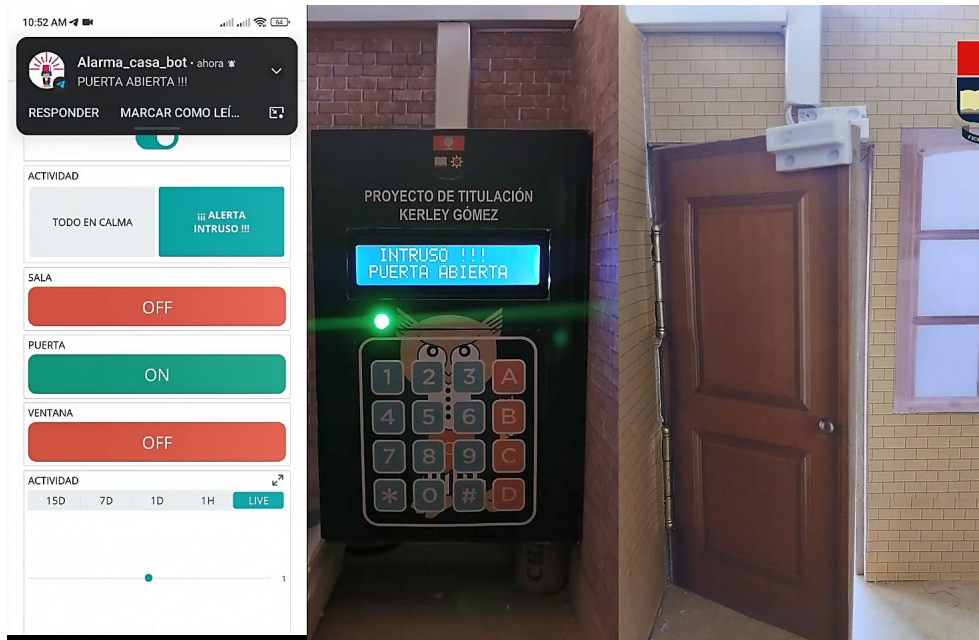
**Figura 3.102** Detención de la alerta de la alarma desde el *bot* Alarma\_casa\_bot

Así se prueba el funcionamiento del sensor magnético que emula detectar intrusos, con el ingreso en ventanas, de la residencia. En esta acción tanto el *bot*, la aplicación y la pantalla *LCD* se encuentran en correcto sincronismo.

### **Detección mediante el sensor magnético de la puerta**

El sensor magnético que está colocado en la puerta de la cocina permite detectar la intrusión en el instante que se separe las dos partes del sensor, una distancia de 15 a 25 (mm). En el instante que ocurre la separación, el sistema emite el sonido de alerta por parte del *buzzer*, al revisar la aplicación de *Arduino IoT Cloud* se observa el *widget* de actividad que esta de color verde en el mensaje “!!! ALERTA INTRUSO !!!”. El *widget* puerta está en *ON* de color verde, el *chart* de actividad se coloca en 1. La aplicación se encuentra con los cambios correctos respecto a la apertura de la puerta.

También, se observa la llegada de la notificación por parte del *bot* Alarma\_casa\_bot, el cual alerta del evento mediante el mensaje “PUERTA ABIERTA”. Los tres métodos de alerta el *bot*, la aplicación y la caja, mediante el *buzzer* y el *LCD*, está funcionando en conjunto, todo lo mencionado se puede observar en la Figura 3.103.



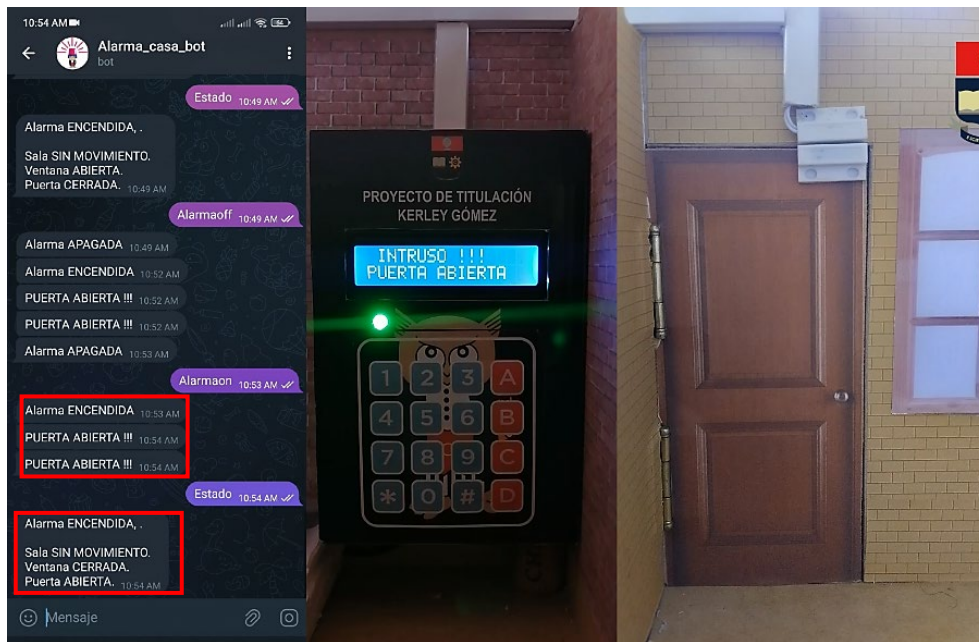
**Figura 3.103** Detección mediante el sensor magnético de la puerta – aplicación

La misma acción de abrir la puerta se realiza nuevamente, la alarma se enciende a través del *bot* Alarma\_casa\_bot mediante el mensaje “Alarmaon”, cuando el sistema haya encendido después de los 30 segundos se abre la puerta por unos instantes y se la vuelve a cerrar, con la finalidad de probar que a pesar de que el sensor magnético, sea de la ventana o de la puerta, se vuelvan a cerrar, la alerta ya es emitida, tal como se puede observar en la Figura 3.104. El sonido que emite el *buzzer* no se detendrá, aunque los sensores recuperen su estado de calma.

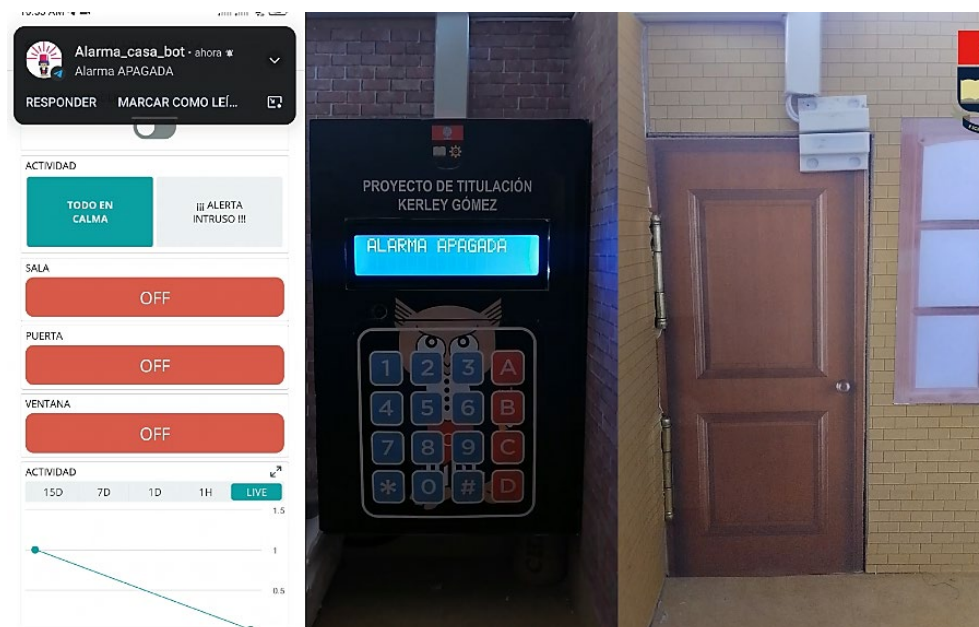
Las notificaciones durante el encendido de la alarma, durante la alerta de la puerta abierta se muestran en el *bot*. Se ha recibido dos notificaciones de altera con el mensaje “PUERTA ABIERTA”, debido a que la puerta se mantuvo abierta por un tiempo. El sistema emite las notificaciones, del sensor comprometido mientras estén separados sus partes, pero en el instante que se vuelven a unir y recuperan su estado de calma deja de hacerlo, pero esto no significa que la alarma dejará de emitir el sonido, y que al revisar el estado de los sensores este esté colocado como cerrado.

Al enviar el mensaje “Estado” desde el *bot*, se observa que la alarma está encendida y que la puerta está abierta, a pesar de que físicamente esta está cerrada. La alarma mientras este encendida mantendrá el estado del sensor cuando exista activad, no importa si la ventana o la puerta se abran por unos segundos y se los cierre de inmediato, la intrusión ha ocurrido y el sistema alertara mediante el sonido, notificará sobre el evento a *Telegram*, así como mantener los *widgets* en la aplicación en estado de actividad.





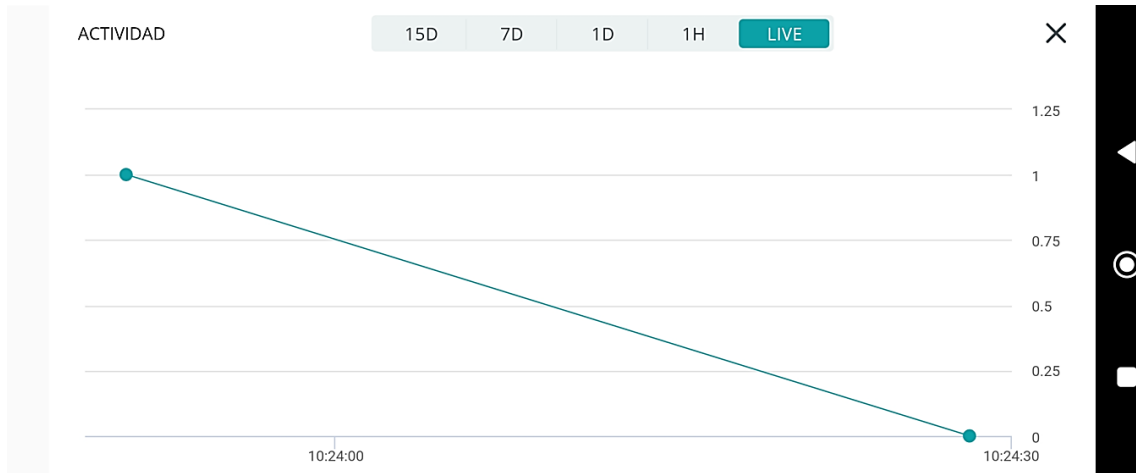
**Figura 3.104** Detección mediante el sensor magnético de la puerta – Alarma\_casa\_bot  
 De manera similar a los sensores anteriores, para detener la alerta del sistema se apaga la alarma, en este caso se lo hace utilizando la aplicación, en donde se muestra que se notifica mediante el *bot* sobre el apagado de la alarma, y como ahora que la alarma se apaga, toma el *widget* de puerta el estado actual, es decir de la puerta cerrada. Tal como muestra la Figura 3.105.



**Figura 3.105** Detención de la alerta de la alarma desde la aplicación

Durante las pruebas de detección mientras está encendida la alarma, en la aplicación existe un *widget* que se encarga de guardar los eventos de las alertas ocurridas en el

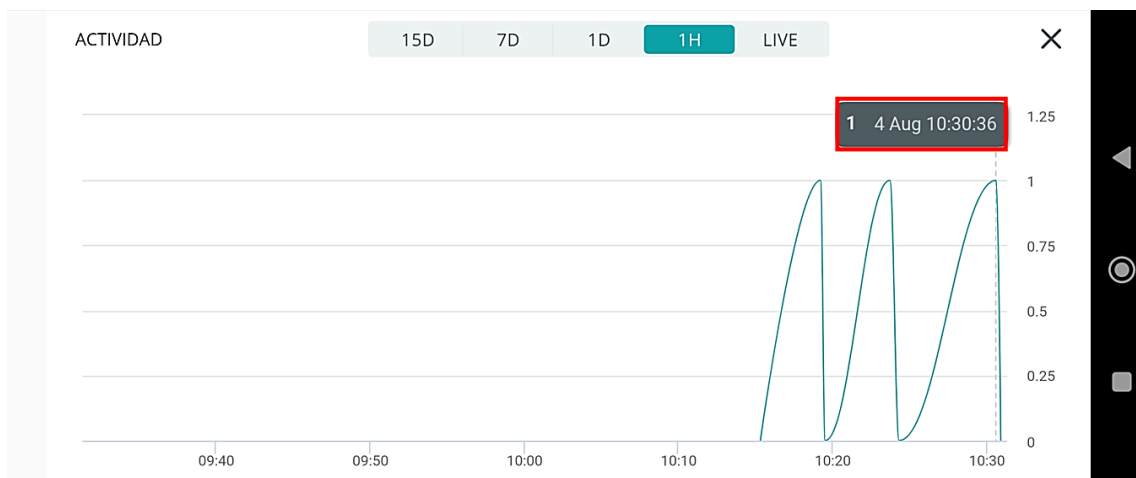
tiempo. El *widget chart* de actividad mediante una gráfica muestra los eventos ocurridos, colocando en 1 si ha ocurrido una emisión de alerta por parte del sistema, sin importar de que sensor se trate. Toma el valor de 0 mientras no exista algún evento de alerta, la gráfica muestra en tiempo real los eventos, tal como se observa en la Figura 3.106



**Figura 3.106** *Chart* de actividad

Este *widget* permite monitorear el rendimiento de la alarma, permite identificar qué horas o días son los más propensos a sufrir algún intento de intrusión en la residencia. La gráfica se puede analizar en vivo, por hora, por día, por semana o por 15 días. En el proyecto al estar utilizando la versión gratuita los datos estadísticos solo se almacenan durante 24 horas.

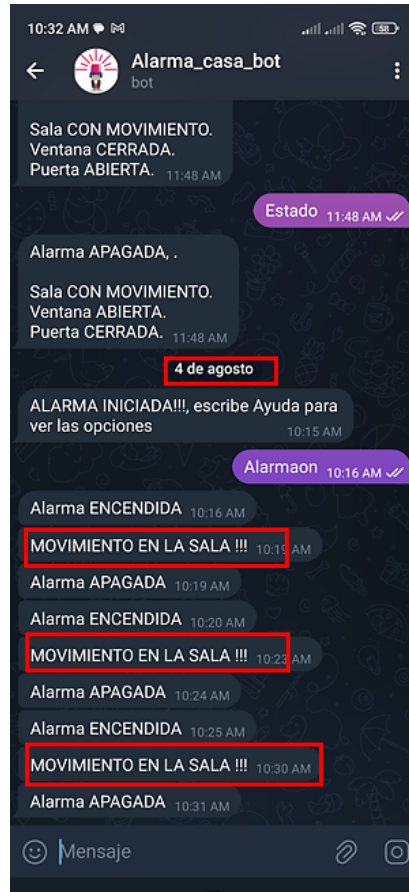
Al realizar pruebas para comprobar su funcionamiento se ha realizado tres intrusiones detectadas por medio del *PIR*, las cuales se muestran en la Figura 3.107.



**Figura 3.107** Eventos almacenados en el *chart*



Los tres eventos tienen el valor de 1 en la curva de la gráfica que otorga el chart. Si se observa el último evento, este ocurrió el 4 de agosto de 2023 a las 10:30 am, si se verifica en el *bot* se observa que efectivamente existieron tres detecciones en la sala, siendo el último movimiento en la sala el día 4 de agosto de 2023 a las 10:30 am, tal como se muestra en la Figura 3.108.



**Figura 3.108** Eventos en el *bot* Alarma\_casa\_bot

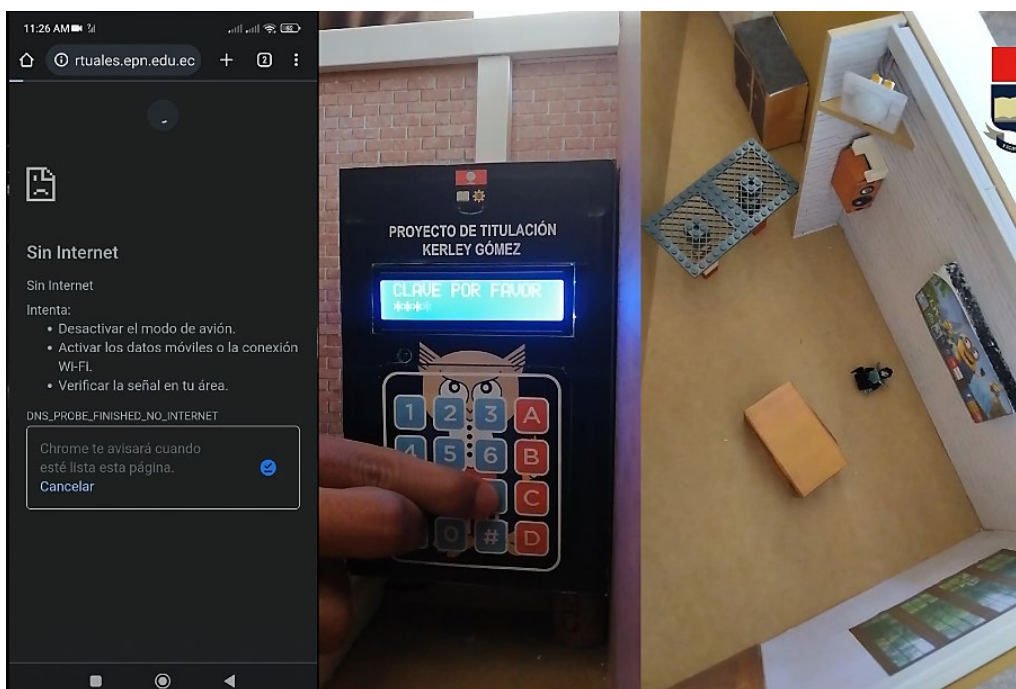
Se ha comprobado que el sistema funciona de manera correcta, operando los tres sensores acordes a los requerimientos establecidos. Se debe especificar que al igual que los sensores magnéticos, el sensor de movimiento *PIR* alertará de la detección siempre que el intruso este en el rango de detección, pero no importa que salga del rango ya que la primera alerta será la encargada de emitir la alerta hasta que se apague la alarma.

Todas las pruebas realizadas se han hecho basadas en que la residencia contará con conexión a *Internet* de manera permanente, algo que el usuario debe garantizar. Pero, para que el sistema no quede obsoleto si no existe *Internet*, se realizan pruebas mediante su funcionamiento solo por medio del *LCD* y teclado físico ubicados en la caja de la alarma.

## Funcionamiento de la Alarma sin *Internet*

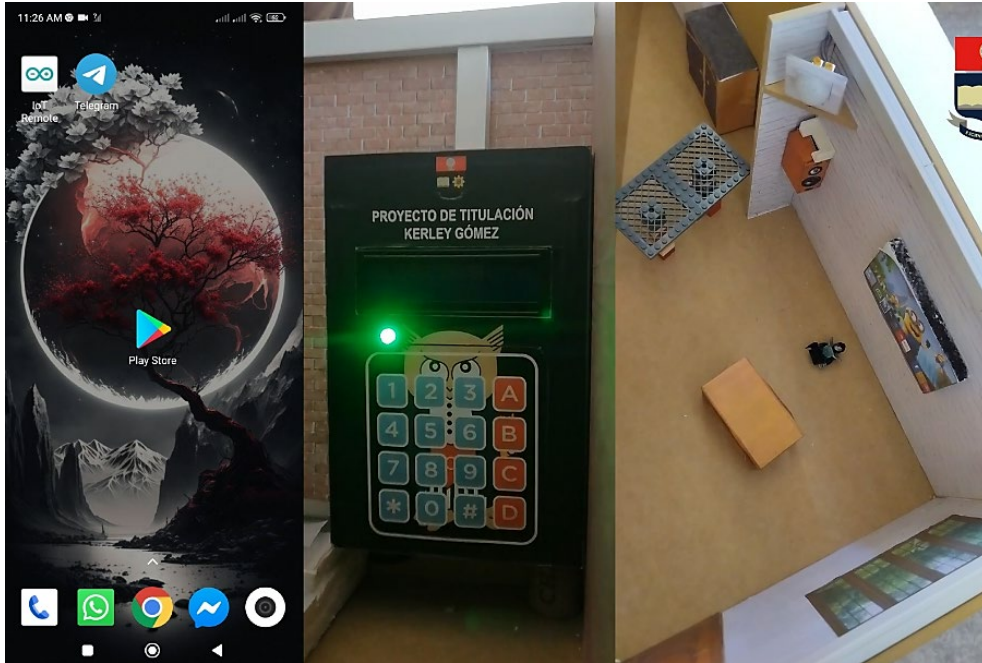
El funcionamiento del prototipo sigue el alcance planteado, donde el control de la alarma debe ser por medio de *Internet*, ya sea por el *bot* de *Telegram* o por la aplicación en *Android*. Si se da el caso que el servicio de *Internet* no esté disponible, el sistema garantiza que alertará por medio del *buzzer* y *LCD* de la detección de algún intruso, por cualquiera de los tres sensores implementados.

En la Figura 3.109 se observa cómo no existe acceso a *Internet* en la red *Wi-fi* a la que está conectada la alarma y que se definió en el desarrollo del código del sistema. Internamente durante la ejecución del sistema se realiza evaluaciones de conectividad con la finalidad de que, al no existir *Internet*, como es el presente caso, el sistema funcione de manera normal en la emisión de alerta por medio de los elementos físicos, que son el *LCD* y el *buzzer*. Se realiza el encendido de la alarma mediante la utilización del teclado, presionando la tecla numeral (#) e ingresando la clave.



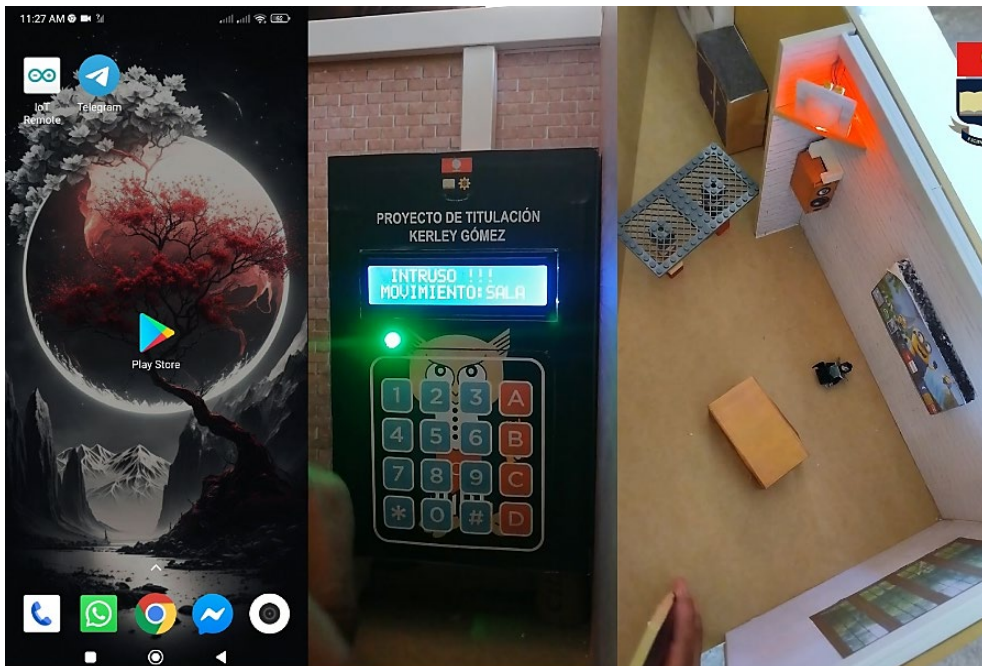
**Figura 3.109** Sistema sin *Internet*

El proceso de encendido del sistema se realiza de manera normal, esperando los 30 segundos posteriores al ingreso de la clave. Como se observa Figura 3.110, al encender la alarma en ningún momento se recibe la notificación al *bot* Alarma\_casa\_bot, ya que no hay conexión a *Internet* por parte del sistema, pero esto no impide que el sistema inicie con normalidad y pueda detectar intrusos.



**Figura 3.110** Sistema encendido sin *Internet*

Para probar que el sistema funciona sin problemas al no tener *Internet*, se realiza la intrusión en la sala, donde el sensor de movimiento *PIR* detecta la presencia. En el *LCD* se observa el mensaje “INTRUSO!!!”, “MOVIMIENTO:SALA”, el *buzzer* emite la alerta mediante el sonido constante, pero en ningún momento llega la notificación a el móvil, dado que no hay *Internet*. Tal como se muestra en la Figura 3.111.



**Figura 3.111** Detección mediante el *PIR* – Sin *Internet*



Para detener la alerta emitida por el *buzzer* se debe apagar la alarma, en este caso al no haber *Internet*, se lo hace por medio del teclado, presionando # e ingresando la clave. Así se prueba que el sistema funciona sin problemas al no haber *Internet*, especificando que su control será con el teclado en conjunto con el *LCD*.

### Reconexión del sistema a *Internet*

Continuando con el caso anterior el sistema se encuentra sin *Internet*, una de las facilidades que tiene *Arduino IoT Cloud*, mediante la librería *thingProperties.h*, es que al contar con las credenciales de conexión de la red *Wi-fi* y al utilizar la función *ArduinoCloud.update()*, el sistema intenta conectarse a la red *Wi-fi*, de ser el caso que no esté disponible, o permite evaluar el acceso a *Internet*. En este caso la red está disponible pero no cuenta con acceso a *Internet*, por tal motivo no puede comunicarse con las plataformas.

En el momento que se reestablece el *Internet*, el sistema se conectará automáticamente, y para poder comprobar llega una notificación al *bot* *Alarma\_casa\_bot* sobre el inicio del sistema de alarma, como se muestra en la Figura 3.112.

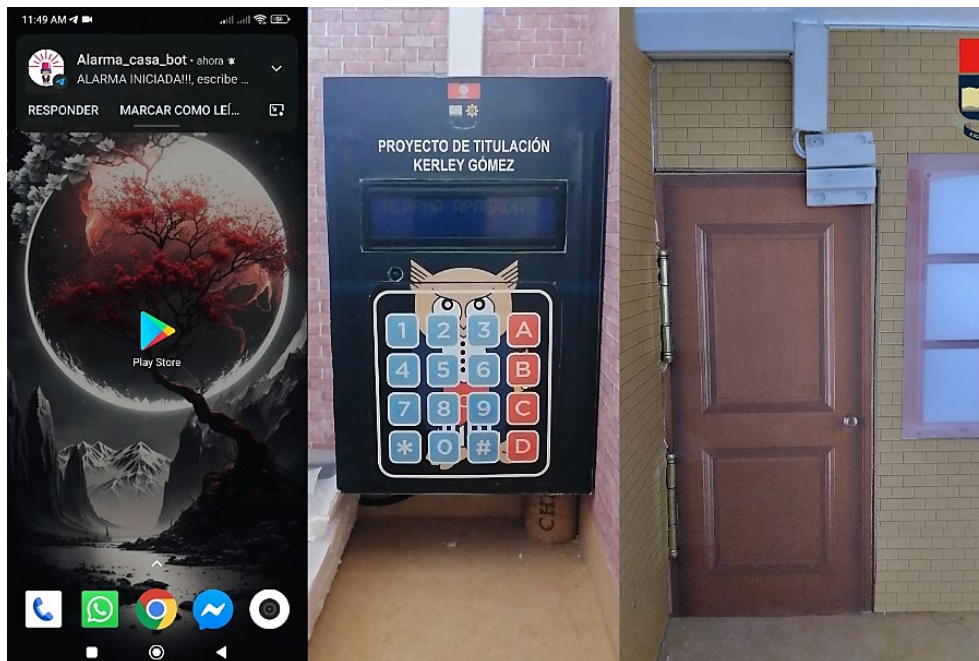
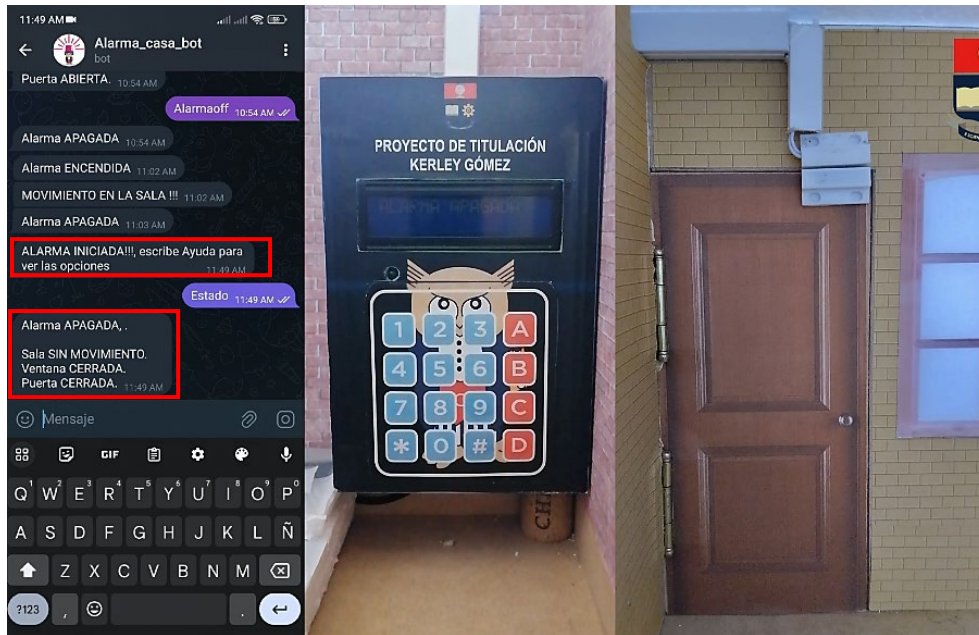


Figura 3.112 Reconexión del sistema a *Internet*

En este caso la alarma se encuentra apagada y para comprobar la existencia de la comunicación con las plataformas, una vez reestablecido el *Internet*, se envía el mensaje “Estado” desde el *bot*, recibiendo como respuesta que la alarma está apagada y que la sala esta sin movimiento, la ventana está cerrada, la puerta cerrada. Tal como se muestra en la Figura 3.113.

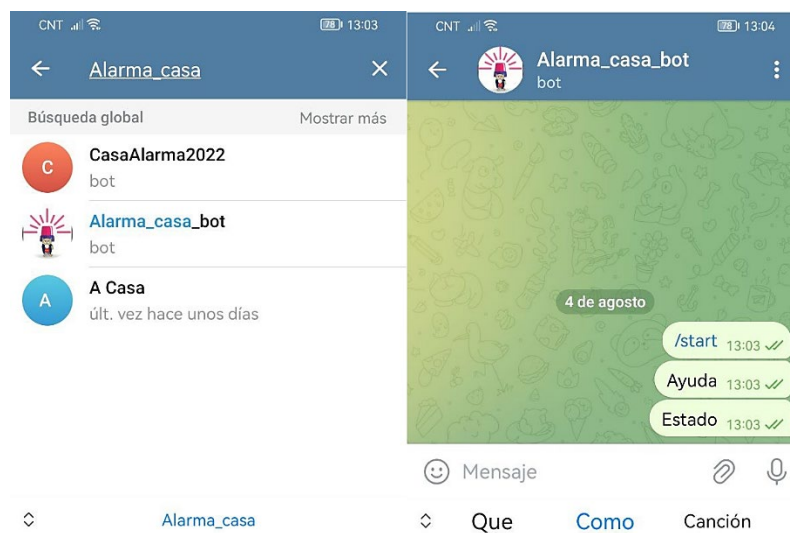


**Figura 3.113** Verificación de la comunicación con la alarma

De esta manera se prueba que el sistema permite tener una reconexión automática a la red.

### Funciones adicionales

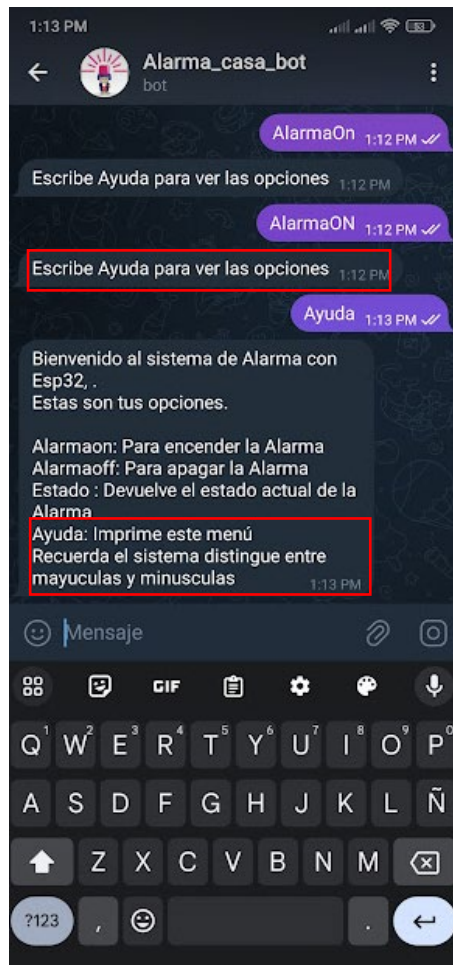
El *bot* Alarma\_casa\_bot está disponible y puede ser localizado en *Telegram*, pero al ingresar en él e iniciar el envío de comandos, por parte de otro usuario, el *bot* no responderá, es decir el sistema de alarma no podrá ser controlado por cualquier usuario, como muestra la Figura 3.114.



**Figura 3.114** Acceso al *bot* Alarma\_casa\_bot desde otro usuario

De esta forma se prueba que el uso del *bot* es exclusivo para el usuario que se definió en el código mediante el ID.

Durante la utilización del *bot*, mediante los comandos o mensajes ya establecidos, la alarma solo responderá si el mensaje se envía tal cual está especificado, es decir por ejemplo si se desea encender la alarma y se envía el mensaje “AlarmaOn”, comando que no existe, se recibe como respuesta “Escribe Ayuda para ver las opciones”, esto ocurrirá siempre que se escriba mal los mensajes aceptados, para lo cual se escribe Ayuda para conocer la opciones disponibles y en donde se especifica que el sistema distingue entre mayúsculas y minúsculas. Tal como se muestra en la Figura 3.115.



**Figura 3.115** Comandos aceptados por parte del *bot* Alarma\_casa\_bot

Otra funcionalidad que se debe mencionar es que durante el uso del prototipo se observó que la pantalla *LCD* deja de iluminar el texto, esto se ha definido para evitar que resulte molesto en especial en la noche. Si el usuario desea visualizar el mensaje impreso en la pantalla en cualquier momento, debe presionar la tecla “D” del teclado físico, de esta forma se iluminará la pantalla *LCD* por un momento y luego se apagará nuevamente. En la Figura 3.116 se muestra el encendido de la iluminación del *LCD*, logrando observar que la alarma está apagada.



**Figura 3.116** Encendido de la iluminación en el *LCD*

En la Tabla 3.2 se muestra el resumen de todas las pruebas llevadas a cabo durante el funcionamiento del prototipo de alarma implementado en la maqueta.

**Tabla 3.2** Pruebas de Funcionamiento

Prueba realizada	Funcionamiento	
	Correcto	Incorrecto
Sensor de movimiento <i>PIR</i>	X	
Sensor magnético ventana	X	
Sensor magnético puerta	X	
<i>LED</i> de encendido y <i>LED</i> de movimiento	X	
Encendido de la Alarma desde la <i>app</i> de <i>Arduino IoT Cloud</i>	X	
Apagado de la Alarma desde la <i>app</i> de <i>Arduino IoT Cloud</i>	X	
Encendido de la Alarma desde el <i>bot</i> <i>Alarma_casa_bot</i>	X	
Apagado de la Alarma desde el <i>bot</i> <i>Alarma_casa_bot</i>	X	
Encendido de la Alarma desde el teclado físico	X	

Prueba realizada	Funcionamiento	Funcionamiento
	Correcto	Incorrecto
Apagado de la Alarma desde el teclado físico	X	
Control de la Alarma desde la <i>app</i> de <i>Arduino IoT Cloud</i>	X	
Control de la Alarma desde el <i>bot</i> <i>Alarma_casa_bot</i>	X	
Revisión del estado de los sensores desde la <i>app</i> de <i>Arduino IoT Cloud</i>	X	
Revisión del estado de los sensores desde el <i>bot</i> <i>Alarma_casa_bot</i>	X	
Cambio de la clave de encendido/apagado de la Alarma desde el teclado físico	X	
Notificación a él <i>bot</i> <i>Alarma_casa_bot</i> de la detección de intrusos	X	
Cambio de estado de los <i>widgets</i> en la <i>app</i> de <i>Arduino IoT Cloud</i> en la detección de intrusos	X	
Emisión constante de alerta por parte del <i>buzzer</i> durante la detección	X	
Sincronismo por parte del <i>bot</i> , la <i>app</i> y la parte física de la alarma	X	
Funcionamiento de la Alarma sin <i>Internet</i>	X	
Uso exclusivo del usuario permitido al <i>bot</i> <i>Alarma_casa_bot</i>	X	
Reconexión a <i>Internet</i> del sistema de Alarma	X	

Todas las pruebas realizadas permiten comprobar que el prototipo de alarma residencial permite su control por medio del *bot* *Alarma\_casa\_bot* en *Telegram* y desde la aplicación en *Arduino IoT Cloud*. Permitiendo alertar a el usuario sobre la intrusión de personas no autorizadas a la residencia.

La operación del prototipo se puede visualizar mediante el video demostrativo accesible mediante el código QR del Anexo II.



## Costo del prototipo

En la Tabla 3.3 se detallan los costos referenciales de cada material utilizado durante el desarrollo del prototipo de alarma residencial, los costos obtenidos son en base a los expuestos en Mercado Libre Ecuador.

**Tabla 3.3** Tabla de costos del prototipo de alarma

Material	Cantidad	Precio unitario	Precio Total
<i>ESP32</i>	1 (u)	\$ 13.00	\$ 13.00
Sensor <i>PIR</i>	1 (u)	\$ 2.50	\$ 2.50
Sensor magnético	2 (u)	\$ 2.50	\$ 5.00
<i>LED</i>	2 (u)	\$ 0.02	\$ 0.04
<i>Buzzer</i>	1 (u)	\$ 2.50	\$ 2.50
<i>LCD 2x16 I2C</i>	1 (u)	\$ 6.50	\$ 6.50
<i>Keypad 4x4</i>	1 (u)	\$ 3.65	\$ 3.65
Cable <i>protoboard</i> 10 (cm) M-M	8 (u)	\$ 0.09	\$ 0.90
Cable <i>AWG 24</i>	8 (m)	\$ 0.20	\$ 1.60
Caja plástica	1 (1)	\$ 4.99	\$ 4.99
Placa PCB	1 (u)	\$ 12.00	\$ 12.00
Mano de obra	20 (h)	\$ 10.00	\$ 200.00
		<b>Total</b>	\$ 252.68

## 4 CONCLUSIONES

- Se concluyó que uno de los requerimientos más importantes, para el diseño del prototipo de alarma residencial, es contar con una plataforma que permita acceder a las herramientas más básicas para el desarrollo de la aplicación. La plataforma de *Arduino IoT Cloud* permitió desarrollar la aplicación para el control del sistema de alarma por medio de *Internet*, aprovechando las herramientas de su versión gratuita, permitiendo que el usuario pueda monitorear el estado de los sensores implementados en el domicilio, además de encender o apagar la alarma. Su interfaz amigable hace que el usuario controle la alarma de una manera intuitiva.

- La selección de *hardware* y *software* mediante el análisis de los requerimientos permitió elegir al microcontrolador *ESP32* como elemento central del prototipo de alarma. El *ESP32* permite reducir la cantidad de elementos requeridos en desarrollo del prototipo, en donde sobresale contar con conectividad *Wi-fi*, de esta forma permite ubicar el circuito principal en cualquier lugar del domicilio ya que se podrá conectar a la red de manera más rápida sin necesitar de cableado adicional al ya existente para los sensores. La plataforma *Web Editor* disponible en *Arduino Cloud* permitió trabajar en conjunto con el *ESP32* de manera eficiente durante el desarrollo del código gracias a su amplia compatibilidad con placas de desarrollo de código abierto y a las librerías disponibles.
- Para el diseño del sistema de alarma mediante una investigación se determinó que para crear el *bot*, en la plataforma de *Telegram*, la herramienta más adecuada era el *bot Botfather* gracias a contar con una amplia documentación, así como tutoriales. El control de la alarma debe ser lo más sencilla por parte del usuario, por tal razón los comandos establecidos en el código permiten que el usuario pueda encender y apagar la alarma, revisar el estado de sensores, recibir notificaciones de alerta durante la detección de intruso en el domicilio. Todo el control se lo pude hacer con palabras simples y relacionadas a cada acción.
- La implementación del prototipo permitió unir la parte física con el *software* de una manera correcta. El hecho de haber logrado integrar diferentes tecnologías como *Arduino IoT Cloud* y *Telegram* en un solo sistema sobre el *ESP32*, demostró que la automatización de dispositivos por medio del *IoT* es viable en cualquiera área, en este caso relacionado a la seguridad residencial.
- La serie de pruebas realizadas al prototipo de alarma, instalado sobre la maqueta, permitieron demostrar el cumplimiento de los objetivos planteados en el proyecto de manera correcta. El sincronismo de *Arduino IoT Cloud*, *Telegram* y la parte física del sistema permiten que cualquier acción realizada, por cualquier medio, durante el control de la alarma se vea reflejada en los demás sistemas. Adicionalmente, con la finalidad de evitar conflictos con la aplicación, la notificación de una alerta o acción se recibe solo al *bot* de *Telegram*.
- Durante las pruebas se ha prestado especial atención a la usabilidad y experiencia del usuario, al desarrollar una aplicación y un *bot* de *Telegram* de interfaz intuitiva para controlar la alarma. Este enfoque en la experiencia del

usuario garantiza una interacción sencilla y amigable con la tecnología planteada en el prototipo.

- La implementación de una alarma residencial demuestra un claro enfoque en abordar problemáticas relevantes, como la seguridad del hogar, y aplicar soluciones tecnológicas para mejorarla. Este resultado tiene un impacto potencialmente positivo en la protección y tranquilidad de los usuarios. La concepción de un prototipo funcional y práctico podría abrir oportunidades para explorar la posibilidad de convertir este proyecto en un producto comercializable.
- El proyecto presentado cumplió con los requerimientos establecidos, aunque la operatividad se centra en un control mediante *Internet*, se garantiza como extra que el sistema permita su control físico, así ofrecer al usuario la protección en la residencia, aunque no exista conectividad hacia *Internet*.

## 5 RECOMENDACIONES

- El prototipo de alarma se diseñó para uso exclusivo de un usuario, establecido en el código, por lo que se recomienda como trabajo a futuro complementar el sistema y hacerlo más dinámico, permitiendo añadir o eliminar los usuarios que controlen el sistema de alarma.
- Se sugiere incorporar un sistema de *backup* de energía en el sistema, lo que permitirá que la alarma opere continuamente a pesar de existir algún corte de energía en la residencia.
- La conectividad a la red *Wi-fi* del domicilio se establece mediante las credenciales durante el desarrollo del código, por lo cual se recomienda como desarrollo futuro incorporar las funcionalidades de la librería *Wi-fi manager* del *ESP32*, lo que permitirá conectarse a cualquier red de una manera fácil, y evitar tener que escribir las credenciales en el código cada vez que se desarrolle para un domicilio distinto.
- El prototipo se ha desarrollado para cubrir la seguridad de ventanas, puertas, y detección de movimiento, para lo cual se implementó un solo elemento para cada caso, por lo que se recomienda aprovechar la escalabilidad del prototipo, adecuándolo a la cantidad de sensores que sea necesario, para lo cual también se recomienda evaluar las necesidades de utilizar una versión de paga de la plataforma *Arduino IoT Cloud*.

- El control del sistema de alarma se lo puede hacer por las dos plataformas establecidas, por lo que se sugiere que el usuario establezca patrones de seguridad en el acceso a las aplicaciones, de esta forma evitar que, si el terminal del usuario sufre de algún robo, el delincuente no pueda acceder a las *apps* de manera sencilla.

## 6 REFERENCIAS BIBLIOGRÁFICAS

- [1] R. Cadena, "Ecuador: Con más de 50 000 robos Cierra El Año El País y la inseguridad sofoca a diario a los ciudadanos," Metro Ecuador, <https://www.metroecuador.com.ec/noticias/2023/01/02/ecuador-con-mas-de-50-000-robos-cierra-el-ano-el-pais-y-la-inseguridad-sofoca-a-diario-a-los-ciudadanos/> (accessed Jul. 29, 2023).
- [2] "La Importancia de la Seguridad Residencial," SERVAGRO LTDA, <https://seguridadesvg.com/blog/blog-empresarial-1/la-importancia-de-la-seguridad-residencial-13> (accessed Jun. 6, 2023).
- [3] "Fundamentos y aplicaciones de seguridad en redes wlan," Google Libros, [https://books.google.es/books?hl=es&lr=&id=k3JuVG2D9IMC&oi=fnd&pg=PA1&dq=redes%2Bwlan&ots=8Guc3wdS8Q&sig=HD\\_zkxzHkrjsO-nPW07p7aPLU60#v=onepage&q=redes%20wlan&f=false](https://books.google.es/books?hl=es&lr=&id=k3JuVG2D9IMC&oi=fnd&pg=PA1&dq=redes%2Bwlan&ots=8Guc3wdS8Q&sig=HD_zkxzHkrjsO-nPW07p7aPLU60#v=onepage&q=redes%20wlan&f=false) (accessed May 31, 2023).
- [4] ESP32 series - espressif systems, [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf) (accessed Jun. 1, 2023).
- [5] Especificaciones del módulo ESP32 - blogger, <https://vasanza.blogspot.com/2021/07/especificaciones-del-modulo-esp32.html> (accessed Jun. 8, 2023).
- [6] Paguayo, "¿Cómo funciona el sensor pir HC-SR501 con arduino?," MCI Capacitación, <https://cursos.mcielectronics.cl/2022/12/06/como-funciona-el-sensor-pir-hc-sr501-e-interconectarlo-con-arduino/> (accessed May 31, 2023).
- [7] F. Navarro, "Qué es un sistema de alarmas y Cómo Funciona," Grupo Navarro, <https://gruponavarro.pe/blog/el-sistema-de-alarmas/> (accessed Jun. 5, 2023).
- [8] F. Argüello, "Tecnología IOT en la Industria de Seguridad Electrónica," Infoteknico, <https://www.infoteknico.com/tecnologia-iot-en-la-industria-de-seguridad/> (accessed Jun. 5, 2023).
- [9] 6HQVRU LQIUUDURMR GH PRYLPLHQWR 3,5 +& 65 - punto flotante, <https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf> (accessed Jun. 6, 2023).

- [10] Name \*, "MC-38 Door magnetic sensor switch," Einstronic Enterprise, <https://einstronic.com/product/mc-38-door-magnetic-sensor-switch/> (accessed Jul. 17, 2023).
- [11] T. A. Team, "Arduino Cloud," Arduino Documentation, <https://docs.arduino.cc/arduino-cloud/> (accessed Jun. 6, 2023).
- [12] "Preguntas Frecuentes," Telegram, <https://telegram.org/faq/es#p-que-es-telegram-que-puedo-hacer-aqui> (accessed Jun. 6, 2023).
- [13] J. A. R. Morales, "Esp32 características Y pines - pasión electrónica," NICA, <https://pasionelectronica.com/esp32-caracteristicas-y-pines/> (accessed Jun. 7, 2023).
- [14] LuisLlamas, "ESP8266, La alternativa a Arduino con wifi," Luis Llamas, <https://www.luisllamas.es/esp8266/> (accessed Jun. 7, 2023).
- [15] "Arduino Uno REV3," Arduino Official Store, <https://store.arduino.cc/products/arduino-uno-rev3> (accessed Jun. 7, 2023).
- [16] "16x2 I2C LCD display module with blue backlight," Parallax, <https://www.parallax.com/product/16x2-i2c-lcd-display-module-with-blue-backlight/> (accessed Jul. 17, 2023).
- [17] "Blynk IOT software platform," Blynk IoT Software platform, <https://blynk.io/blynk-iot-low-code-software-platform> (accessed Jun. 7, 2023).
- [18] Ing. M. Salvador, "Teclado matricial 4x4 Arduino," Blog Arduino, LabVIEW y Electrónica, <https://electronicamade.com/teclado-matricial-4x4/> (accessed Jun. 7, 2023).
- [19] R. GR, "Cómo Pueden Ayudarte los bots de telegram y cuáles son Los Mejores," ADSLZone, <https://www.adslzone.net/como-se-hace/telegram/bots-funcionamiento/> (accessed Jun. 12, 2023).
- [20] P. Jecrespom and Jecrespom, "Ide Arduino," Aprendiendo Arduino, <https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/> (accessed Jun. 19, 2023).
- [21] K. the Dev, "How to set up your telegram bot using BotFather," Medium, <https://blog.devgenius.io/how-to-set-up-your-telegram-bot-using-botfather-fd1896d68c02> (accessed Jun. 19, 2023).
- [22] J. M. López, José María López Lee todos mis artículos (1469), V. Castellano, L. C. Cano, and M. Terol, "Asistentes Online Para Crear Tu Propio Bot de Telegram," Blogthinkbig.com, <https://blogthinkbig.com/asistentes-online-crear-tu-propio-bot-de-telegram> (accessed Jun. 19, 2023).
- [23] Fluke, "¿Qué es la ley de ohm?," Fluke, <https://www.fluke.com/es-es/informacion/blog/electrica/que-es-la-ley-de-ohm> (accessed Jun. 19, 2023).

- [24] G. Dice: et al., "Resistencia de led, Como Calcularla en Función a la corriente," HeTPro, <https://hetpro-store.com/TUTORIALES/resistencia-de-led/> (accessed Jun. 19, 2023).
- [25] Admin, "Funciones Arduino Void Loop () y void setup (),” Proyectos con Arduino, <https://proyectosconarduino.com/cursos/funciones-arduino-void-loop-y-void-setup/> (accessed Jun. 21, 2023).
- [26] Miduvi – Ministerio de Desarrollo Urbano y vivienda – somos la entidad ..., <https://www.habitatyvivienda.gob.ec/wp-content/uploads/downloads/2018/09/LINEAMIENTOS-MINIMOS-PARA-REGISTRO-Y-VALIDACION-DE-TIPOLOGIAS-DE-VIVIENDA.pdf> (accessed Jul. 17, 2023).
- [27] N. P.-A. Murcia, "Consejos para la instalación de alarmas de seguridad,” TECNOSeguro, <https://www.tecnoseguro.com/analisis/alarma/consejos-instalacion-alarmas-seguridad> (accessed Jul. 17, 2023).
- [28] Estudio de estándares de diseños físicos de lan y su adecuación a la topología del lugar, <https://www.revista.unam.mx/vol.5/num5/art28/art28-1b.htm> (accessed Jul. 19, 2023).

## **7 ANEXOS**

La lista de los Anexos se muestra a continuación:

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Códigos fuente

# ANEXO I: Certificado de Originalidad

## CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 24 de agosto de 2023

De mi consideración:

Yo, **LEANDRO ANTONIO PAZMIÑO ORTIZ**, en calidad de Director del Trabajo de Integración Curricular titulado **IMPLEMENTACIÓN DE PROTOTIPO DE UNA ALARMA RESIDENCIAL QUE PERMITA SU CONTROL A TRAVÉS DE INTERNET**, componente **IMPLEMENTACIÓN DE PROTOTIPO DE UNA ALARMA RESIDENCIAL QUE PERMITA SU CONTROL A TRAVÉS DE TELEGRAM Y DE UNA APLICACIÓN ANDROID** elaborado por el estudiante **KERLEY SANTIAGO GÓMEZ TAPIA** de la carrera en **TECNOLÓGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES**, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 10%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.

[TIC KGomez-Reporte turnitin.pdf](#)

Atentamente,

Leandro Pazmiño Ortiz

Docente

Escuela de Formación de Tecnólogos



## ANEXO II: Enlaces



**Anexo II.I** Código QR de la implementación y pruebas de funcionamiento del prototipo de alarma

Link: [Video demostrativo KGOMEZ.mp4](#)

## ANEXO III: Códigos Fuente

```
#include <Arduino_ConnectionHandler.h>
```

```
#include <ArduinolotCloud.h>
```

```
#include <WiFi.h>
```

```
#include <WiFiClientSecure.h>
```

```
#include <UniversalTelegramBot.h> //Librería para Telegram
```

```
#include "thingProperties.h" //Librería con las variables de Arduino lot Cloud
```

```
#include <Wire.h> //Librería para el LCD
```

```
#include <LiquidCrystal_I2C.h> //Librería para el LCD
```

```
#include <EEPROM.h> //Librería para guardar en la memoria EEPROM
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
const uint8_t ROWS = 4;
```

```
const uint8_t COLS = 4;
```

```
char keys[ROWS][COLS] = {
```

```
  { '1', '2', '3', 'A' },
```

```
  { '4', '5', '6', 'B' },
```

```
  { '7', '8', '9', 'C' },
```

```
  { '*', '0', '#', 'D' }
```

```
};
```

```
uint8_t colPins[COLS] = { 16, 4, 2, 15 };
```

```
uint8_t rowPins[ROWS] = { 19, 18, 5, 17 };
```

```

char clave_anterior[7]; //Variable para almacenar una clave para el keypad por defecto
para la primera utilización

char nueva_clave[7]; //Variable para almacenar la clave de activación y desactivación
de la alarma

char ingresaPass[7]; //variable para almacenar la clave ingresada por el usuario
mediante le keypad

int indice = 0;

int direccionIndicador = 100; // Dirección en la EEPROM para almacenar el indicador de
primera ejecución

int direccionAlarmaActivada = 200; // Dirección en la EEPROM para almacenar el estado
de alarmaActivada

int pinLedVerde = 12;

int pinLedRojo = 13;

int cantidadTeclas = 6; // Cantidad de teclas que se deben ingresar

int teclasIngresadas = 0; // Contador de teclas ingresadas

#define WIFI_SSID "TUNITA"

#define WIFI_PASSWORD "santiagor9"

#define BOT_TOKEN "6328090633:AAF1jX_nDHcuZ9hNhtPmDT17kFzb-URMshQ"
//Variable que almacena el token del bot de Telegram

#define ID_Chat "1285841367" //Variable que almacena el Id personal de Telegram

const unsigned long tiempo = 1000;

WiFiClientSecure secured_client;

UniversalTelegramBot bot(BOT_TOKEN, secured_client);

unsigned long tiempoAnterior;

int estadoLed12 = 0;

```

String chat\_id; //Variable para almacenar el ID del usuario que envía instrucciones mediante el bot de Telegram

bool alarmaActivada = false; //Variable para almacenar el estado de la alarma activado o desactivado

bool mensajeEnviado = false; //Variable para almacenar el estado de los mensajes enviados a telegram

bool cambioDesdeTelegram = false; //Variable para almacenar el estado activado o desactivado enviado del mensaje a telegram

int inicio = 1;

int ledPin = 25; //Variable para el led del sensor PIR, pin 25

int ledOn = 33; //Variable para el led de encendido o apagado, pin 33

int ven = 26; //Variable para el sensor magnético de la ventana, pin 26

int pue = 27; //Variable para el sensor magnético de la Puerta, pin 27

#define MOTION\_SENSOR\_PIN 13 // Variable para el funcionamiento del sensor PIR, pin 13

#define BUZZER\_PIN 32 // Variable para el funcionamiento del buzzer, pin 32

int motionStateCurrent = LOW; // Variable para almacenar el estado actual del PIR

int motionStatePrevious = LOW; // Variable para almacenar el estado previo del PIR

String ac; //Variable para almacenar el estado en texto del sensor PIR

String pu; //Variable para almacenar el estado en texto del sensor magnético de la puerta

String vn; //Variable para almacenar el estado en texto del sensor magnético de la ventana

bool vtana; //Variable para almacenar el estado actual del sensor PIR

bool prt; //Variable para almacenar el estado actual del sensor magnético

bool sla; //Variable para almacenar el estado actual del sensor magnético

bool cloudprevio = false; //Variable para almacenar el estado previo del *switch* en Arduino IoT Cloud

```

bool cloudactual;      //Variable para almacenar el estado actual del switch en Arduino
IoT Cloud

//-----Variables para el encendido del LCD

bool encender = false;

unsigned long t = 0;

int t_ms_encendido = 7000;

//-----

bool conexionprevia = false; //Variable para almacenar el estado previo de la conexión
con Arduino IoT Cloud

bool conexionactual;    //Variable para almacenar el estado actual de la conexión con
Arduino IoT Cloud

#define frecuencia_inicio 220 // define la frecuencia inicial del sonido

#define frecuencia_final 880 // define la frecuencia final del sonido

#define tiempo_ejecute 30 // define el tiempo en segundos para que el código se ejecute

bool z;

void setup() {

  Serial.begin(115200);

  lcd.init();

  lcd.print("...INICIADO...");

  EncenderLCD();

  EEPROM.begin(512);

  bool primeraEjecucion = EEPROM.read(direccionIndicador) != 'X';

  if (primeraEjecucion) {

    // Establecer valores predeterminados

    strcpy(clave_anterior, "ABCD45");

    strcpy(nueva_clave, "ABCD45");

```

```
alarmaActivada = false; // Valor predeterminado cuando no se ha guardado en la EEPROM
```

```
// Guardar valores predeterminados en la EEPROM
```

```
for (int i = 0; i < 7; i++) {
```

```
    EEPROM.write(i, clave_anterior[i]);
```

```
    EEPROM.write(i + 7, nueva_clave[i]);
```

```
}
```

```
// Guardar indicador de primera ejecución en la EEPROM
```

```
EEPROM.write(direccionIndicador, 'X');
```

```
EEPROM.commit();
```

```
} else {
```

```
// Recuperar valores de la EEPROM, la clave
```

```
for (int i = 0; i < 7; i++) {
```

```
    clave_anterior[i] = EEPROM.read(i);
```

```
    nueva_clave[i] = EEPROM.read(i + 7);
```

```
}
```

```
alarmaActivada = EEPROM.read(direccionAlarmaActivada) == '1'; // Cargar estado de alarmaActivada desde la EEPROM
```

```
}
```

```
EEPROM.end();
```

```
//-----Variables para los sensores, leds y buzzer -----
```

```
pinMode(pue, INPUT_PULLUP);
```

```
pinMode(ven, INPUT_PULLUP);
```

```
pinMode(ledPin, OUTPUT);
```

```
pinMode(ledOn, OUTPUT);
```

```
pinMode(MOTION_SENSOR_PIN, INPUT);
```

```
pinMode(BUZZER_PIN, OUTPUT);
```

```

connectToWiFi();
}

void loop()
{
  ArduinoCloud.update();

  // ----- Verifico si la conexión se perdió y se reestableció para notificar a Telegram --
  -----

  conexionactual = ArduinoCloud.connected();

  if(conexionactual && !conexionprevia){

    bot.sendMessage(ID_Chat, "ALARMA INICIADA!!!, escribe Ayuda para ver las
opciones", "");

    conexionprevia = true;

  }

  if(!conexionactual){

    conexionprevia=false;

  }

  //----- Comprueba si esta presionado una tecla en el keypad -----

  char key = readKeypad();

  if (key == '#') { // Si presiono # Solicito que ingrese la clave para poder Activar o
Desactivar -----

    indice = 0;

    EncenderLCD();

    lcd.clear();

    lcd.print("CLAVE POR FAVOR");

    while (true) {

      char key = readKeypad();

      if (key) {

```



```

    lcd.setCursor(indice, 1);

    lcd.print('*');

    ingresaPass[indice] = key;

    indice++;

}

if (indice >= 6) {

    ingresaPass[indice] = '\0';

    break;

}

}

} else if (key == '*') { // Si la tecla es * Se ingresa para el cambio de clave ----

    EncenderLCD();

    indice = 0;

    cambiar_clave();

} else if (key == 'D' || key == 'd') { // Si la tecla es D se ilumina el LCD para ver el
mensaje ----

    EncenderLCD();

}

if (indice >= 6) { // Si los valores ingresados son iguales o mayores a 6 se
evalua la clave

    if (strcmp(nueva_clave, ingresaPass) == 0) { // Si es igual a la clave guardada se
acepta

        lcd.clear();

        lcd.print("CLAVE CORRECTA");

        if (alarmaActivada) { //Si el estado actual de la alarma es verdadero, lo
cambio a apagado

```

```

        alarmaActivada = false;          //Coloca el valor de la variable en falso para una
acción posterior

        if (ArduinoCloud.connected()) {

            bot.sendMessage(ID_Chat, "Alarma APAGADA", ""); //Envía un mensaje al bot de
Telegram, si existe conexión con Arduino Cloud

            onoff = LOW;    //Cambia el estado del switch a apagado, en la aplicación en
Arduino IoT Cloud

        }

        cambioDesdeTelegram = true;

        estadoLed12 = 0;    //Se asigna el valor de cero a la variable estadoLed12, esto
para uso posterior para verificar el estado de la alarma

        lcd.clear();          //Se limpia la pantalla del LCD

        lcd.setCursor(0, 0);    //Se coloca el cursor en la posición 0,0 en el LCD

        lcd.print("ALARMA APAGADA");    //Se imprime el mensaje de ALARMA
APAGADA en el LCD

    } else {                    //Si el estado es apagado lo cambio a encendido

        alarmaActivada = true;

        z = true;

        if (ArduinoCloud.connected()) {

            onoff = HIGH;

            bot.sendMessage(ID_Chat, "Alarma ENCENDIDA", "");

        }

        cambioDesdeTelegram = true;    //Valor utilizado para verificar que se notificó a
Telegram

        estadoLed12 = 1;

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("ALARMA ENCENDIDA");

    }

```

```

} else {          //Si la clave ingresada no es igual a la almacenada

    lcd.clear();

    lcd.print("CLAVE INCORRECTA"); //Se imprime en el LCD el mensaje CLAVE
INCORRECTA

    delay(500);

    lcd.clear();

    lcd.print("REINTENTE");

    if (alarmaActivada) {          // Si el estado de la alarma es verdadero ejecuto
acciones de alaram On

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("ALARMA ENCENDIDA");

    } else {          // Si es falso ejecuto acciones de alarma apagada

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("ALARMA APAGADA");

    }

}

    indice = 0;          //Reinicio a 0 el valor de la variable utilizada como contador
a la hora de ingresar la clave.

}

if (encender) {

    if (millis() - t > t_ms_encendido) {

        encender = false;

        lcd.noBacklight();

    }

}

```

```

    ArduinoCloud.update();

// -----Verificar mensajes de Telegram Bot API-----

    if (millis() - tiempoAnterior > tiempo) // Verifica si ha pasado el tiempo especificado
desde la última ejecución de este bloque de código

    {

        int numerosMensajes = bot.getUpdates(bot.last_message_received + 1); // Obtiene
el número de mensajes nuevos recibidos por el bot de Telegram

        while (numerosMensajes) // Inicia un bucle while para procesar todos los mensajes
nuevos

        {

            Serial.println("Comando recibido");

            mensajesNuevos(numerosMensajes); // Llama a una función llamada
'mensajesNuevos' y le pasa el número de mensajes nuevos como argumento

            numerosMensajes = bot.getUpdates(bot.last_message_received + 1); // Obtiene el
número de mensajes nuevos recibidos nuevamente para comprobar si hay más
mensajes

        }

        tiempoAnterior = millis(); // Actualiza el valor de 'tiempoAnterior' con el tiempo actual
en milisegundos

    }

//----- Control desde Arduino Iot Cloud -----

    cloudactual = onoff;          // Guardo el valor verdadero o falso de la variable del
switch de cloud

    if(cloudprevio != cloudactual ){ // Si el estado previo del switch es verdadero y el
estado actual es falso existió un cambio

        if (onoff) {              // Si el switch es verdadero ejecuto acciones de encendido
de la alarma

            ArduinoCloud.update();

```

```

    if (onoff && !mensajeEnviado) // Verificar si switch está encendido y el mensaje no
se ha enviado a telegram

    {

        if (!cambioDesdeTelegram) // Verificar si el cambio no fue desde Telegram

        {

            estadoLed12 = 1;

            bot.sendMessage(ID_Chat, "Alarma ENCENDIDA", ""); // Envía notificación a
Telegram del encendido de la alarma

        }

        if (!mensajeEnviado) // Verificar si el mensaje no se ha enviado

        {

            mensajeEnviado = true;

        }

    }

    alarmaActivada = true; // Variable global que guarda el estado Encendido o
Apagado de la alarma

    cambioDesdeTelegram = false;

    z = true;

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("ALARMA ENCENDIDA"); // Proyecto en el LCD

    EncenderLCD(); // Llamo a la función para encender el LCD

}

else {

    ArduinoCloud.update();

```

```

    if (!onoff && mensajeEnviado) // Verificar si SWITCH está apagada y el mensaje
se ha enviado

    {

        if (!cambioDesdeTelegram) // Verificar si el cambio no fue desde Telegram

        {

            estadoLed12 = 0;

            bot.sendMessage(ID_Chat, "Alarma APAGADA", ""); // Notifico a Telegram del
APAGADO de la alarma

        }

        if (mensajeEnviado) // Verificar si el mensaje se ha enviado

        {

            mensajeEnviado = false;

        }

    }

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("ALARMA APAGADA");

    EncenderLCD();

    alarmaActivada = false;

    cambioDesdeTelegram = false;

}

cloudprevio = clouddactual; //Guardo el estado actual del switch en el estado previo

}

// Guardar el estado de alarmaActivada en la EEPROM

EEPROM.begin(512);

```

```

EEPROM.write(direccionAlarmaActivada, alarmaActivada ? '1' : '0');

EEPROM.commit();

EEPROM.end();

if (alarmaActivada) {          // Si el estado de la alarma es verdadero ejecuto acciones
de alarm On

    if(z){
        buzzer();}

    z = false;

    alarmaOn();

} else {                      // Si es falso ejecuto acciones de alarma apagada

    alarmaOff();

}

}

//----- Función para iniciar la conexión a la red y plataformas

void connectToWiFi() {

    Serial.print("Conectando a la red ");

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD); // Conexion a wifi

    secured_client.setCACert(TELEGRAM_CERTIFICATE_ROOT);

    Serial.print("\nConectado a la red Wi-Fi. Dirección IP: ");

    initProperties();          // Inicia las propiedades guardadas en thingProperties.h
de arduino cloud

```



```
    ArduinoCloud.begin(ArduinoIoTPreferredConnection); // Inicia la conexión entre
    arduino iot cloud y el esp32
```

```
    setDebugMessageLevel(2);
```

```
    ArduinoCloud.printDebugInfo();
```

```
}
```

```
//----- Función para ejecutar los mensajes recibidos de Telegram -----
```

```
void mensajesNuevos(int numerosMensajes)
```

```
{
```

```
    for (int i = 0; i < numerosMensajes; i++)
```

```
    {
```

```
        String chat_id = bot.messages[i].chat_id; // Se obtiene el ID del usuario que ha escrito
    al bot
```

```
        String text = bot.messages[i].text;      // Se obtiene el mensaje que se ha enviado
```

```
        if (chat_id == ID_Chat) {                // Comprueba si el ID del mensaje es igual al ID
    del telegram del usuario
```

```
            if (text == "Alarmaon")            // Si el mensaje es igual Alarmaon, ejecuto acciones
    de Encendido
```

```
            {
```

```
                estadoLed12 = 1;
```

```
                onoff = HIGH;                    // Cambio el estado switch del arduino cloud
```

```
                bot.sendMessage(chat_id, "Alarma ENCENDIDA", ""); // Envió notificación del
    encendido al telegram
```

```
                cambioDesdeTelegram = true;
```

```
                alarmaActivada = true;
```

```

z = true;

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("ALARMA ENCENDIDA");      // Imprime el mensaje de encendido en el
LCD

EncenderLCD();                      // Llamó la función para encender el LCD

} else if (text == "Alarmaoff")     // Si el mensaje es Alarmaoff ejecuto acciones de
Apagado
{
    estadoLed12 = 0;                // Guardo el valor para ejecutar el estado de la
alarma en telegram

    onoff = LOW;

    bot.sendMessage(chat_id, "Alarma APAGADA", "");

    cambioDesdeTelegram = true;

    alarmaActivada = false;

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("ALARMA APAGADA");

    EncenderLCD();

} else if (text == "Estado")        // Si el mensaje es Estado
{
    if (estadoLed12)                //si el estado de la alarma es verdadero o encendido
    {

        String ayuda = "Alarma ENCENDIDA, " ".\n\n"; // Envió la notificación con el estado
de la alarma y de los sensores

        ayuda += "Sala " + ac + ".\n";

        ayuda += "Ventana " + vn + ".\n";

        ayuda += "Puerta " + pu + ".\n";

```

```

    bot.sendMessage(chat_id, ayuda, "");
}
else //Si el estado es falso o apagado
{
    String ayuda = "Alarma APAGADA, " ".\n\n";
    ayuda += "Sala " + ac + ".\n";
    ayuda += "Ventana " + vn + ".\n";
    ayuda += "Puerta " + pu + ".\n";
    bot.sendMessage(chat_id, ayuda, "");
}
} else if (text == "Ayuda") //Si el mensaje es Ayuda desde Telegram, responde
con los comandos aceptados en el bot
{
    String ayuda = "Bienvenido al sistema de Alarma con Esp32, " ".\n";
    ayuda += "Estas son tus opciones.\n\n";
    ayuda += "Alarmaon: Para encender la Alarma \n";
    ayuda += "Alarmaoff: Para apagar la Alarma \n";
    ayuda += "Estado : Devuelve el estado actual de la Alarma\n";
    ayuda += "Ayuda: Imprime este menú \n";
    ayuda += "Recuerda el sistema distingue entre mayúsculas y minúsculas \n";
    bot.sendMessage(chat_id, ayuda, "");
} else if (text == "Pass"){
    bot.sendMessage(chat_id,nueva_clave,"");
}else {
    bot.sendMessage(chat_id, "Escribe Ayuda para ver las opciones", "");// Si se
ingresa un comando erroneo se envia una respuesta
}

```

```

    }

}

}

//----Funcion para leer el teclado -----
char readKeypad() {

    // Realizar el escaneo del teclado matricial

    for (int col = 0; col < COLS; col++) {

        pinMode(colPins[col], OUTPUT);

        digitalWrite(colPins[col], LOW);

        for (int row = 0; row < ROWS; row++) {

            pinMode(rowPins[row], INPUT_PULLUP);

        }

        delayMicroseconds(10);

        for (int row = 0; row < ROWS; row++) {

            if (!digitalRead(rowPins[row])) {

                delay(50); // Debounce

                // Determinar el carácter correspondiente a la tecla presionada

                char key = keys[row][col];

                // Esperar a que se libere la tecla

                while (!digitalRead(rowPins[row])) {

                    delay(10);

                }

                // Restaurar los pines del teclado a sus configuraciones originales

                for (int row = 0; row < ROWS; row++) {

                    pinMode(rowPins[row], OUTPUT);

```

```

    digitalWrite(rowPins[row], LOW);
}

for (int col = 0; col < COLS; col++) {
    pinMode(colPins[col], INPUT_PULLUP);
}

return key;
}
}

// Restaurar los pines del teclado a sus configuraciones originales
for (int row = 0; row < ROWS; row++) {
    pinMode(rowPins[row], OUTPUT);
    digitalWrite(rowPins[row], LOW);
}

pinMode(colPins[col], INPUT_PULLUP);
}

return '\0';    // Si no se presionó ninguna tecla, devolver '\0'
}

//-----Función para ejecutar acciones de los sensores cuando esta la alarma ON -----
void alarmaOn() {
    int sensorValue = digitalRead(ven);    //Lee el valor del sensor de ventana
    int sensorValuep = digitalRead(pue);    //Lee el valor del sensor de puerta

    motionStatePrevious = motionStateCurrent; //Asigno el valor del estado actual del PIR
a la variable de estado previo

    motionStateCurrent = digitalRead(MOTION_SENSOR_PIN); // Asigno el valor del
sensor PIR a la variable de estado actual

    digitalWrite(ledOn, HIGH);    // Enciendo el LED para mostrar que la alarma esta
encendida

```

```

//sla = digitalRead(MOTION_SENSOR_PIN);

if (motionStatePrevious == LOW && motionStateCurrent == HIGH) { // Evaluó si el
estado del PIR cambio de bajo a alto

    if (ArduinoCloud.connected()) { //Evaluó si existe conexión a una de las
plataformas, lo que significa conexión

        pir = HIGH; //Si hay conexión informo del estado del PIR a cloud y
Telegram

        bot.sendMessage(ID_Chat, "MOVIMIENTO EN LA SALA !!!", "");

    }

    digitalWrite(ledPin, HIGH); //Se enciende el led rojo con cada movimiento

    ac = "CON MOVIMIENTO"; // Asigno una frase para utilizarla a la hora de
ver el Estado en Telegram

    } else if (motionStatePrevious == HIGH && motionStateCurrent == LOW) { // Evaluó
si el estado del PIR paso de alto a bajo

        digitalWrite(ledPin, LOW); //Apago el led

    }

if (sensorValue == HIGH) { //Evaluó si el estado del sensor de ventana es alto

    if (ArduinoCloud.connected()) { //Evaluó la conexión

        ventana = HIGH; //Muestro el estado de activo en cloud

        bot.sendMessage(ID_Chat, "VENTANA ABIERTA !!!", ""); //Notifico en Telegram

        vn = "ABIERTA"; //Asigno la frase para el Estado en Telegram

    }

}

if (sensorValuep == HIGH) { // Evaluó Si el estado del sensor de Puerta es alto

    if (ArduinoCloud.connected()) { //Evaluó si hay conexión

        puerta = HIGH; //Muestro el estado en cloud

```

```

bot.sendMessage(ID_Chat, "PUERTA ABIERTA !!!", ""); //Notifico en Telegram

pu = "ABIERTA";          //Asigno la frase para el Estado de Telegram
}
}

if (motionStateCurrent || sensorValuep || sensorValue) { //Si alguno de los sensores se
activo

digitalWrite(BUZZER_PIN, HIGH);    // El buzzer emite un sonido constante

if (ArduinoCloud.connected()) {    //Evaluó la conexión

mov = 1;          //Asigno estado de 1 al chart de cloud

}

lcd.clear();

lcd.setCursor(0, 0);

lcd.print(" INTRUSO !!!");        // Proyecto en el LCD un mensaje de ALERA con el
sensor activado

if(motionStateCurrent){

lcd.setCursor(0, 1);

lcd.print("MOVIMIENTO:SALA");

}else if(sensorValuep){

lcd.setCursor(0, 1);

lcd.print("PUERTA ABIERTA");

}else if(sensorValue){

lcd.setCursor(0, 1);

lcd.print("VENTANA ABIERTA");

}

EncenderLCD();//Enciendo el LCD

}

```



```
}
```

```
//-----Funcion para el Apagado de la Alarma-----
```

```
void alarmaOff() {
```

```
    digitalWrite(ledOn, LOW);          //Apago el led que muestra el estado encendido o  
    apagado
```

```
    digitalWrite(BUZZER_PIN, LOW);    //Silencio el Buzzer
```

```
    if (ArduinoCloud.connected()) {   //Verifico la conexión
```

```
        mov = 0;
```

```
        pir = digitalRead(MOTION_SENSOR_PIN);
```

```
    }
```

```
    digitalWrite(ledPin, digitalRead(MOTION_SENSOR_PIN)); //Enciendo el led acorde al  
    estado del PIR
```

```
    sla = digitalRead(MOTION_SENSOR_PIN);
```

```
    if(sla){
```

```
        ac = "CON MOVIMIENTO";
```

```
        digitalWrite(ledPin, HIGH);
```

```
    }else{
```

```
        ac = "SIN MOVIMIENTO";
```

```
        digitalWrite(ledPin, LOW);
```

```
    }
```

```
    ventana = digitalRead(ven);       //Asigno el estado del sensor de ventana a la  
    variable en cloud
```

```
    vtana = digitalRead(ven);
```

```
    if (vtana){
```

```
        vn = "ABIERTA";
```

```
    }else{
```

```

    vn = "CERRADA";
}

puerta = digitalRead(pue);          //Asigno el estado del sensor de Puerta a la variable
en cloud

prta = digitalRead(pue);

if (prta){
    pu = "ABIERTA";
}else{
    pu = "CERRADA";
}
}
}

```

//-----Funcion para cambiar la clave -----

```

void cambiar_clave() {
    lcd.clear();
    lcd.backlight();
    lcd.print("INGR CLAVE ACTUAL");    //Despliego en el LCD el mensaje

    char clave_ingresada[7];
    int indice_clave = 0;

    while (true) {
        char key = readKeypad();

        if (key) {

```

```

lcd.setCursor(indice_clave, 1);

lcd.print('*');          //Despliego un * en cada pulsación

clave_ingresada[indice_clave] = key;//Almaceno los dígitos ingresados 6 veces

indice_clave++;

}

if (indice_clave >= 6) {

    clave_ingresada[indice_clave] = '\0';

    break;

}

}

if (strcmp(clave_ingresada, clave_anterior) == 0) { //Comparo la clave guardada con la
clave ingresada

    lcd.clear();

    lcd.print("INGR NUEVA CLAVE");

while (true) {

    char key = readKeypad();

    if (key) {

        lcd.setCursor(indice, 1);

        lcd.print(key);

        nueva_clave[indice] = key;

        indice++;

    }
}

```

```

if (indice >= 6) {
    nueva_clave[indice] = '\0';
    break;
}
}

strcpy(clave_anterior, nueva_clave); //Copeo la nueva clave en la clave guardada

EEPROM.begin(512);          //Inicio la memoria para guardar la nueva clave
for (int i = 0; i < 7; i++) {
    EEPROM.write(i, clave_anterior[i]);
    EEPROM.write(i + 7, nueva_clave[i]);
}
EEPROM.commit();
EEPROM.end();

delay(500);
lcd.clear();
lcd.print("CLAVE CAMBIADA");
if (ArduinoCloud.connected()) {
    bot.sendMessage(ID_Chat, "Se cambio la clave, Envía Pass para revisar", "");
}
delay(500);

if (alarmaActivada) {      // Si el estado de la alarma es verdadero ejecuto acciones
de alaram On
    lcd.clear();
    lcd.setCursor(0, 0);

```

```

    lcd.print("ALARMA ENCENDIDA");

} else {          // Si es falso ejecuto acciones de alarma apagada

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("ALARMA APAGADA");

}

EncenderLCD();

delay(500);

} else {          // Si la clave ingresada no es igual a la clave guardada se despliega
clave incorrecta

    delay(500);

    lcd.clear();

    lcd.print("CLAVE INCORRECTA");

    delay(500);

}

indice = 0;

}

//----- Funcion para encender el LCD -----

void EncenderLCD() {

    t = millis();

    encender = true;

    lcd.backlight();

}

void buzzer(){

```

```

unsigned long startTime = millis(); // tiempo de inicio

unsigned long elapsedTime = 0; // tiempo transcurrido

while (elapsedTime < tiempo_ejecute * 1000) {

    float progress = (float)elapsedTime / (tiempo_ejecute * 1000); // progreso actual (0-
1)

    int currentFrequency = map(progress, 0, 1, frecuencia_inicio, frecuencia_final);

    tone(BUZZER_PIN, currentFrequency, 100); // reproduce el tono con la frecuencia
actual durante 100ms

    delay(300); // espera 200ms antes de cambiar a la siguiente frecuencia

    noTone(BUZZER_PIN); // detiene el sonido durante 200ms

    delay(300); // espera 200ms antes de continuar

    elapsedTime = millis() - startTime; // actualiza el tiempo transcurrido

}

// Último pitido más fuerte

tone(BUZZER_PIN, frecuencia_final, 100);

delay(300); // espera 300ms con el último tono fuerte

noTone(BUZZER_PIN); // detiene el sonido

//delay(1000); // espera 1 segundo antes de volver a empezar

}

```