ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL CONTROL AUTOMÁTICO DEL ENCENDIDO Y APAGADO DE LAS LUMINARIAS DE LA OFICINA 8 DE LA ESFOT A TRAVÉS DE UNA PLATAFORMA WEB, APLICACIÓN EN ANDROID Y POR MEDIO DE TELEGRAM

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

QUISHPI LOACHAMIN GABRIEL ALEJANDRO

DIRECTOR: LEANDRO ANTONIO PAZMIÑO ORTIZ

DMQ, JULIO 2024

CERTIFICACIONES

Yo, Gabriel Alejandro Quishpi Loachamin 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.

Gabriel Alejandro Quishpi Loachamin

gabriel.quishpi@epn.edu.ec

gabo.alb11@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Gabriel Alejandro Quishpi Loachamin, 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.

Gabriel Alejandro Quishpi Loachamin
CI: 175141308-7

DEDICATORIA

A mis padres, Francisco y Guadalupe, quienes me han brindado su amor incondicional, su apoyo constante y su sabiduría a lo largo de mi vida. Su ejemplo de trabajo arduo y perseverancia ha sido mi guía en cada paso de este camino académico. Sin su sacrificio y comprensión, este logro no habría sido posible.

A mi hermana, por su paciencia, sus palabras de ánimo y por creer en mí incluso en los momentos más difíciles. Gracias por ser mi refugio, por compartir mis alegrías y aliviar mis preocupaciones.

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a Dios, cuya infinita sabiduría, amor y gracia me han guiado y sostenido a lo largo de este arduo camino académico. Su presencia constante me ha brindado fortaleza en los momentos de dificultad, claridad en los momentos de duda y alegría en los momentos de éxito. Sin Su bendición y Su guía, este logro no habría sido posible.

Quiero extender mi gratitud a todos mis profesores, por proporcionarme una formación académica sólida y por fomentar un ambiente de aprendizaje estimulante. Cada uno de ustedes ha contribuido a mi crecimiento profesional y personal.

A mis compañeros de estudio y amigos, gracias por su compañía, por las innumerables horas compartidas, y por ser una fuente constante de apoyo y motivación.

Un especial agradecimiento a mi familia, por su amor incondicional y su confianza en mí.

Finalmente, quiero agradecer a todas aquellas personas que, de una u otra manera, han contribuido a la realización de esta tesis. Su apoyo, aunque no mencionado individualmente, es profundamente valorado y apreciado.

ÍNDICE DE CONTENIDOS

RESUMEN

La implementación de un prototipo para el control automático del encendido y apagado de las luminarias de la oficina 8 de la ESFOT, a través de una plataforma *web*, aplicación en *Android* y *Telegram*, busca desarrollar un sistema de control de luminarias utilizando un *System-on-a-Chip*. El objetivo es mejorar la eficiencia energética y facilitar el manejo de la iluminación. El prototipo se centra en la automatización de luminarias mediante la programación de horarios y el control manual remoto utilizando comunicaciones inalámbricas, integrándose a través de una conexión *Wi-Fi* para gestiones remotas y uso de aplicaciones.

La primera sección describe el componente desarrollado, estableciendo objetivos generales y específicos, que incluyen definir requisitos técnicos, seleccionar *hardware* y *software* adecuados, diseñar el prototipo y realizar pruebas de funcionamiento.

La segunda sección desglosa la metodología en varias etapas, desde la definición de requisitos técnicos hasta la implementación y pruebas del prototipo. Cada etapa incluye pasos específicos como la selección del *hardware*, abarcando la elección del microcontrolador, sensores, relés y la determinación de la fuente de alimentación.

La tercera sección presenta los resultados obtenidos, detallando las pruebas de funcionamiento realizadas. Estas pruebas demuestran la efectividad del sistema en condiciones reales y su capacidad para cumplir con los requisitos establecidos.

Las conclusiones de la cuarta sección resumen los logros del proyecto, destacando la implementación del prototipo y su impacto potencial en la eficiencia energética y la gestión de iluminación en entornos similares.

PALABRAS CLAVE: Control automático, eficiencia energética, Internet de las Cosas (*IoT*), *ESP32*, luminarias, *System-on-a-Chip* (SoC).

ABSTRACT

The implementation of a prototype for the automatic control of the on and off of the luminaires of office 8 of the ESFOT, through a web platform, application on Android and Telegram, seeks to develop a luminaire control system using a System-on-a-Chip. The goal is to improve energy efficiency and facilitate the management of lighting. The prototype focuses on the automation of luminaires through scheduling and remote manual control using wireless communications, integrating through a Wi-Fi connection for remote management and use of applications.

The first section describes the developed component, establishing general and specific objectives, which include defining technical requirements, selecting appropriate hardware and software, designing the prototype and performing operational tests.

The second section breaks down the methodology into several stages, from the definition of technical requirements to the implementation and testing of the prototype. Each stage includes specific steps such as the selection of the hardware, covering the choice of the microcontroller, sensors, relays and the determination of the power supply.

The third section presents the results obtained, detailing the functional tests carried out. These tests demonstrate the effectiveness of the system in real conditions and its ability to meet the established requirements.

The conclusions of the fourth section summarize the achievements of the project, highlighting the implementation of the prototype and its potential impact on energy efficiency and lighting management in similar environments.

KEYWORDS: *Automatic control, energy efficiency, Internet of Things (IoT), ESP32, luminaires, System-on-a-Chip (SoC).*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El proyecto desarrollado consiste en un prototipo destinado al control automático del encendido y apagado de las luminarias de la oficina 8 de la Escuela de Formación de Tecnólogos. Este sistema aprovecha las ventajas de la tecnología del Internet de las Cosas y está diseñado para optimizar el uso de energía eléctrica, incrementando así la eficiencia energética del espacio.

El objetivo principal del proyecto es la implementación de un prototipo basado en un *System-on-a-Chip* para el control automático de las luminarias. Este sistema no solo permite el encendido y apagado automático, sino que también ofrece funcionalidades avanzadas como la configuración de alertas y la gestión a través de diferentes plataformas, incluyendo una aplicación *Android*, una página *web* y *Telegram*. La elección de estas plataformas permite una gran flexibilidad y accesibilidad, facilitando el control del sistema desde cualquier ubicación con acceso a internet.

Para alcanzar el objetivo general, se plantearon varios objetivos específicos. El primero fue la definición de los requisitos técnicos necesarios para el correcto control del encendido y apagado de las luminarias. Esta etapa inicial fue crucial para asegurar que todos los componentes y funcionalidades del sistema estuvieran alineados con las necesidades del proyecto. Luego, se procedió a la selección del *hardware* y el *software* más adecuados. Esta selección incluyó la identificación de componentes como el microcontrolador *ESP32*, el sensor movimiento, y los actuadores necesarios para controlar las luminarias.

El diseño del prototipo fue la siguiente fase crítica del proyecto. En esta etapa, se desarrollaron los esquemas de circuito y los diagramas de flujo para asegurar que todos los componentes trabajaran de manera integrada. Una vez completado el diseño, se pasó a la implementación del prototipo. Este proceso involucró el montaje y la configuración de los componentes en una placa de circuito impreso (*PCB*), asegurando conexiones fiables y duraderas.

El núcleo del sistema es el microcontrolador *ESP32*, conocido por su potencia y versatilidad en aplicaciones de *IoT*. Este microcontrolador es responsable de coordinar las operaciones de los diversos periféricos, incluyendo sensores y relés. La conectividad inalámbrica es un aspecto clave del proyecto, permitiendo el control remoto del sistema mediante *Wi-Fi* 802.11 b/g/n. Esta capacidad de conectividad facilita la integración del

sistema con diversas plataformas de *IoT*, ofreciendo un control flexible y accesible desde cualquier dispositivo con conexión a internet.

El sistema incluye varios componentes esenciales para su correcto funcionamiento. Entre ellos, destaca el sensor *PIR*, utilizado para detectar la presencia de personas en la oficina. Cuando se detecta movimiento, el sensor envía una señal al *ESP32* para activar o desactivar las luminarias según sea necesario. Este sensor es crucial para el funcionamiento automático del sistema. Además, el sistema cuenta con un relé, responsable de la activación física de las luminarias. Este componente se conecta al *ESP32* y actúa como un interruptor, controlando el flujo de corriente hacia las luminarias.

La implementación del prototipo incluye varias etapas clave. En primer lugar, se desarrolló el circuito eléctrico que conecta el *ESP32* con el sensor *PIR* y el relé. Este circuito se ensambló en una *PCB* para asegurar conexiones fiables y duraderas.

A continuación, se desarrolló el software necesario para el funcionamiento del sistema. Esto incluyó la programación del *ESP32* para que pudiera recibir datos del sensor *PIR* y controlar el relé. Se creó un algoritmo que gestiona las condiciones de encendido y apagado de las luminarias, asegurando que el sistema responda de manera eficiente a los cambios en la ocupación de la oficina.

El desarrollo de interfaces de usuario fue otro aspecto importante del proyecto. Se implementaron una aplicación *Android* y una interfaz *web* que permiten a los usuarios configurar y controlar el sistema de manera remota. Además, se desarrolló un *bot* de *Telegram* para ofrecer otra opción de control. Estas interfaces permiten a los usuarios seleccionar entre el control automático y el control manual de las luminarias, ofreciendo una gran flexibilidad en la gestión de la iluminación.

Las pruebas del sistema fueron esenciales para asegurar su funcionamiento correcto. Se realizaron pruebas para cada una de las funciones principales del sistema. En primer lugar, se verificó que las luminarias pudieran ser controladas manualmente a través de la aplicación *Android*, la interfaz *web* y el *bot* de *Telegram*. Luego, se probaron las capacidades de control automático del sistema. Estas pruebas incluyeron la evaluación del sistema de detección de movimiento, esto al momento de realizar una programación de horarios de tal forma que el sensor solo pueda detectar y encender las luminarias dentro de los horarios previamente programados y si se detecta la presencia de personas.

1.1 Objetivo general

Implementar un prototipo con System-on-a-*Chip* (*SoC)* para el control automático del encendido y apagado de las luminarias de la oficina 8 de la Escuela de Formación de Tecnólogos (*ESFOT*).

1.2 Objetivos específicos

- Definir los requisitos técnicos para el control del encendido y apagado de las luminarias.
- Seleccionar el *hardware* y *software* acorde a los requerimientos establecidos.
- Diseñar el prototipo de control del encendido y apagado de las luminarias.
- Implementar el prototipo de control del encendido y apagado de las luminarias.
- Realizar pruebas de funcionamiento del prototipo.

1.3 Alcance

El alcance del proyecto comprende la implementación de un prototipo funcional automático del encendido y apagado de las luminarias de la oficina 8 de la ESFOT que incluye las siguientes funcionalidades:

- Control automático de encendido y apagado de las luminarias en función de al menos dos condiciones.
- Configuración de alertas en función de una condición.
- Control del encendido y apagado por medio de una página *web*, de una aplicación en *Android* y por medio de *Telegram*.

1.4 Marco Teórico

Eficiencia energética

La capacidad de este sistema de incrementar el ahorro energético al permitir el encendido automatizado de las luminarias en función de periodos planificados o condiciones ambientales es una de sus principales ventajas. Esto puede resultar en un menor uso de energía y grandes ahorros a largo plazo. El sistema puede reducir el desperdicio de energía y disminuir su efecto ambiental al tiempo que brinda a los usuarios un ambiente cómodo y bien iluminado al ajustar dinámicamente el consumo de iluminación a las condiciones circundantes [1].

Internet de las cosas (*IoT*)

La red de elementos físicos que utilizan software, sensores y otras tecnologías para conectarse e intercambiar datos con otros sistemas y dispositivos a través de Internet se conoce como Internet de las cosas (IoT). Estos artículos, a veces denominados dispositivos inteligentes, pueden ser desde electrodomésticos estándar hasta grandes máquinas industriales y sensores ambientales. Estos dispositivos recopilan datos, que se transmiten a una plataforma central a través de redes cableadas o inalámbricas, donde se procesan y analizan para proporcionar información útil.

Las aplicaciones de *IoT* son tan variadas como la imaginación humana. Están presentes en muchas industrias diferentes, incluidas las ciudades inteligentes, el transporte, la atención médica, la agricultura y la manufactura. Los dispositivos de *IoT* pueden, por ejemplo, reducir el uso de energía en las ciudades, gestionar el tráfico de manera inteligente, mejorar la eficiencia de las líneas de producción, optimizar el riego agrícola y monitorear la salud de los pacientes desde cualquier lugar [2].

ESP32

Debido a su asequibilidad, potencia y adaptabilidad, el microcontrolador *ESP32* ha ganado mucha tracción en la industria de *IoT* Con una *CPU Xtensa LX106* de doble núcleo con frecuencia de hasta 240 (MHz), este microcontrolador de 32 *bits* puede manejar de manera eficiente aplicaciones complicadas. Es perfecto para almacenar datos y ejecutar programas ya que contiene un buen porcentaje de *RAM* y almacenamiento *flash*.

La conexión integrada, que incorpora *Bluetooth v4.2 BR/EDR* y *BLE* además de *Wi-Fi 802.11 b/g/n*, es su principal característica. Esto simplifica la construcción de aplicaciones de Internet de las cosas para *ESP32* al permitir conexiones inalámbricas a Internet, otros dispositivos y sensores. También es compatible con una amplia gama de sensores y actuadores gracias a sus numerosos periféricos, que incluyen *GPIO*, *ADC*, *DAC*, *PWM*, *I2C*, *SPI* y *UART* [3].

Arduino IoT Cloud

Arduino Cloud se presenta como una plataforma en línea que revoluciona la forma en la que interactúa con los dispositivos electrónicos basados en *Arduino*. Con la ayuda de esta plataforma, los usuarios pueden monitorear y administrar sus proyectos *Arduino* desde cualquier lugar con conexión a Internet, actuando como un vínculo entre el mundo real y el virtual.

Con la ayuda de la interfaz gráfica de usuario (*GUI*) [4], sencilla y fácil de usar de *Arduino Cloud*, los usuarios pueden verificar el estado del dispositivo en tiempo real, controlar el encendido y apagado del dispositivo de forma remota, configurar programaciones automatizadas, recopilar y analizar datos e incluso desarrollar aplicaciones. Diseñado para involucrarse con sus proyectos de una manera más sofisticada.

Además, la plataforma cuenta con una interfaz de programación de aplicaciones (*API*) que permite crear aplicaciones más complejas y personalizadas, brindando a los consumidores la posibilidad de personalizar el control de sus dispositivos para satisfacer sus propios requisitos. Además, *Arduino Cloud* proporciona una comunidad dinámica de usuarios y desarrolladores que promueve la cooperación y el aprendizaje permanente mediante el intercambio de proyectos, guías y recursos [5].

Sensores

Los dispositivos electrónicos conocidos como sensores *Passive Infra Red* (*PIR*) miden las variaciones en la radiación infrarroja liberada por objetos calientes, como humanos y animales. La idea detrás de estos sensores es que todo lo que tenga una temperatura superior a cero irradie luz infrarroja. La cantidad de radiación infrarroja que llega al sensor varía cuando un humano o un animal se mueve dentro de su campo de detección.

Esto da como resultado la producción de una señal eléctrica que indica la presencia de movimiento. Numerosas aplicaciones, incluidos sistemas de seguridad, alarmas contra incendios, puertas automatizadas, iluminación automática y gestión de presencia, emplean sensores *PIR*. Dado que sólo utiliza energía cuando detecta movimiento, su principal beneficio es que utiliza menos energía. También tienen un amplio rango de detección, tienen un precio razonable y son fáciles de instalar [6].

Actuadores

Los componentes electrónicos llamados actuadores de relé permiten impulsos de baja potencia para operar equipos eléctricos de mayor potencia. Dependiendo de la señal de control recibida, funcionan como interruptores gestionados por un electroimán, permitiendo el encendido/apagado de un circuito eléctrico.

Su principal beneficio es que pueden regular corrientes altas con señales de control de baja potencia, lo cual es perfecto para situaciones en las que es necesario controlar equipos eléctricos de mayor potencia, incluidos calentadores, luces, motores y electrodomésticos. Los relés también son piezas resistentes y confiables que se pueden usar con seguridad bajo cargas de corriente elevada [7].

2 METODOLOGÍA

Para el diseño del prototipo, se llevó a cabo una exhaustiva identificación de los requerimientos funcionales y no funcionales. Se priorizó la selección de un hardware compatible con las especificaciones técnicas del proyecto, considerando aspectos como capacidad de procesamiento, memoria disponible, opciones de conectividad (Wi-Fi, Bluetooth) y la disponibilidad de puertos de entrada/salida para integrar sensores y actuadores. Además, se definió la necesidad de desarrollar un software robusto que permitiera la comunicación fluida con una plataforma IoT y la implementación de algoritmos de control automático de las luminarias.

Posteriormente, se realizó una evaluación minuciosa de las diversas plataformas de desarrollo IoT disponibles en el mercado, seleccionando aquella que mejor se ajustaba a los requerimientos del proyecto. Una vez definida la plataforma IoT, se procedió con el desarrollo del firmware, el cual fue cuidadosamente diseñado para garantizar la compatibilidad con el hardware seleccionado y los protocolos de comunicación establecidos. Se implementaron algoritmos de control que permitieran gestionar el encendido y apagado de las luminarias de manera eficiente y confiable, considerando variables como la intensidad lumínica y horarios preestablecidos.

En la fase de implementación, se ensambló el prototipo físico, integrando los componentes electrónicos seleccionados, como el microcontrolador ESP32, sensores de luz y relés. A continuación, se cargó el firmware desarrollado en el microcontrolador y se configuró la plataforma IoT para establecer la comunicación con el dispositivo. Se realizaron pruebas exhaustivas en un entorno controlado para verificar el correcto funcionamiento del prototipo y ajustar los parámetros de configuración según fuera necesario.

Finalmente, se llevó a cabo una evaluación del prototipo en un entorno real, simulando las condiciones de uso previstas. Durante esta fase, se monitoreó el desempeño del sistema en términos de confiabilidad, eficiencia energética y respuesta a los comandos de control. Los resultados obtenidos permitieron validar la factibilidad técnica del prototipo y sentar las bases para su futura implementación a mayor escala.

3 RESULTADOS

En la etapa de ejecución del proyecto, se prevé que habrá una serie de resultados ventajosos tras la implementación del prototipo para encender y apagar las luminarias.

Se prevé que los sensores del prototipo permitan una detección exitosa. Además, se estableció una conexión confiable a la plataforma *Arduino Cloud*, lo que permitió el control remoto de las luminarias a través de la aplicación móvil y la interfaz en línea. Se supone que el prototipo funciona de manera confiable en pruebas del mundo real, reaccionando adecuadamente a las modificaciones y ejecutando rápidamente órdenes de encendido y apagado.

Los resultados del proyecto se presentarán de tal forma que se sigan con lo cinco puntos establecidos en la metodología. A continuación, se describe de forma detallada cada una de las cinco etapas:

3.1 Definición de los requisitos técnicos para el control del encendido y apagado de las luminarias

A la hora de definir los requisitos para el diseño del prototipo se analizan los siguientes aspectos:

Controlador

El motor de este proyecto de control de iluminación inteligente reside en un microcontrolador. Estos microcontroladores no son *chips* cualesquiera, sino *chips* especialmente seleccionados para características únicas que se convierten en componentes clave para el éxito del proyecto.

Equipado con conectividad inalámbrica, este microcontrolador se conecta fácilmente al mundo digital, permitiéndole acceder a Internet y comunicarse con otros dispositivos. Esto lo convierte en un puente ideal entre el mundo físico de los sensores y actuadores y el mundo digital de la información y el control remoto.

Un microcontrolador envía comandos precisos a otros controladores a través de sus pines de salida, controlando el comportamiento de los actuadores y apagando las luces cuando es necesario. La capacidad de control cubre todo el sistema y proporciona un control de la iluminación eficiente y preciso.

Para facilitar la programación y configuración del sistema, el microcontrolador se ha seleccionado cuidadosamente para ser compatible con *C++*, el lenguaje estándar en *Arduino IDE*. Esto permite a los desarrolladores utilizar herramientas familiares y ampliamente conocidas para crear el software que da vida al sistema [8].

Conexión inalámbrica

El sistema requiere de una conexión inalámbrica, la libertad de movimiento es fundamental. Gracias a la conexión inalámbrica, se podrá controlar y monitorear el sistema desde cualquier lugar del mundo, sin importar la distancia. La conexión inalámbrica actúa como un puente invisible que conecta el sistema con el mundo digital. A través de *Wi-Fi*, el sistema se conecta directamente al *módem*, eliminando la necesidad de cables complejos y costosos. Utilizando el estándar *802.11b/g/n*, el sistema opera en la banda no licenciada de 2.4 (GHz), una frecuencia ampliamente utilizada y compatible con la mayoría de los dispositivos [9].

Esto garantiza una conexión estable y confiable, sin interferencias ni interrupciones. La conexión inalámbrica permite un intercambio fluido de información entre el sistema, el sensor y el usuario. Los datos recopilados por el sensor se transmiten de forma segura a través de Internet. Al mismo tiempo, el usuario puede enviar comandos desde su dispositivo móvil o computadora para controlar el movimiento de las cortinas.

Sistema capaz de encender y apagar las luminarias automáticamente según una programación definida

El sistema de control de iluminación inteligente no solo permitirá encender y apagar las luces de forma manual o remota, sino que también te ofrece la posibilidad de programar su funcionamiento automático según tus necesidades. Con el sistema, se podrá establecer horarios personalizados para cada día de la semana, adaptando la iluminación a las rutinas y preferencias.

Interfaz *web*, aplicación Android y *Telegram*

El sistema de control de iluminación inteligente ofrece la libertad de controlar las luces desde cualquier lugar y en cualquier momento, utilizando la interfaz *web*, la aplicación *Android* o *Telegram*.

La plataforma *web* seleccionada deberá ser capaz de entregar un ambiente intuitivo para el usuario. En esta plataforma, el desarrollo y creación del entorno deberá ser fácil y eficiente. Esta página, deberá contar con controles para la programación de horarios y su control de manera o automática.

Diseño y construcción del prototipo

La planificación de un circuito resulta esencial, pues representa de manera gráfica las conexiones entre los elementos electrónicos del sistema. La ejecución del montaje del circuito precisa de meticulosidad para garantizar una conexión segura entre todos los

componentes electrónicos. Es importante dedicar especial cuidado a la calidad de las uniones, reduciendo al mínimo la probabilidad de cortocircuitos o desconexiones accidentales.

Este diseño se llevará a cabo mediante la grabación en una *PCB*, es esencial elaborar un diseño que sea compacto y adaptable al entorno físico en donde será instalado. Este diseño debe representar de forma precisa tanto la disposición física como la lógica de cada componente, garantizando así eficacia al momento de realizar las conexiones necesarias.

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

Selección del hardware

Actualmente hay muchas opciones diferentes de microcontroladores y sensores disponibles. Por lo tanto, para asegurar un proceso adecuado para este prototipo, se realizó un análisis técnico comparativo de cada componente que conforma el diseño.

Selección del microcontrolador

Actualmente existe una variedad de microcontroladores en el mercado con diferentes funcionalidades que pueden usarse para construir adecuadamente aplicaciones basadas en el Internet de las cosas. Examinar características como la velocidad del procesador, la conexión inalámbrica, la conectividad del sensor y el tamaño es el primer paso para seleccionar el microcontrolador para este proyecto. Además, se proporciona la siguiente comparación técnica de los siguientes dispositivos: *ESP32 DEVKIT V1*, *NodeMCU ESP8266* y *Arduino UNO*.

El *ESP32 DEVKIT V1* es un módulo muy adaptable y robusto que combina redes *Bluetooth* y *Wi-Fi* con un microcontrolador. Tiene una *CPU Tensilica LX6* de doble núcleo que puede funcionar hasta 240 (MHz), lo que le brinda un rendimiento sólido para tareas exigentes. El diseño de bajo consumo de energía del *ESP32* lo hace además perfecto para proyectos que requieran eficiencia energética. Una de sus características destacadas es la conectividad inalámbrica; Es compatible con *Bluetooth* 4.2 y *Wi-Fi 802.11 b/g/n*, lo que facilita la integración de la red y la comunicación del dispositivo.

La amplia gama de interfaces y pines de *E/S* del *ESP32 DEVKIT V1* es otra característica digna de mención. Los 30 pines *GPIO* que se muestran en la *Figura 3.1* de este módulo son versátiles y se pueden configurar para realizar una variedad de funciones, incluidas *PWM*, *ADC*, *DAC* e interfaces como *SPI*, *I2C* y *UART*. Debido a su

adaptabilidad, el *ESP32* se puede utilizar en una variedad de contextos, incluidos dispositivos de Internet de las cosas y sistemas de automatización del hogar. La placa de desarrollo también tiene un regulador de voltaje para proporcionar un funcionamiento estable y un convertidor de USB a serie para una programación y depuración sencillas [10].

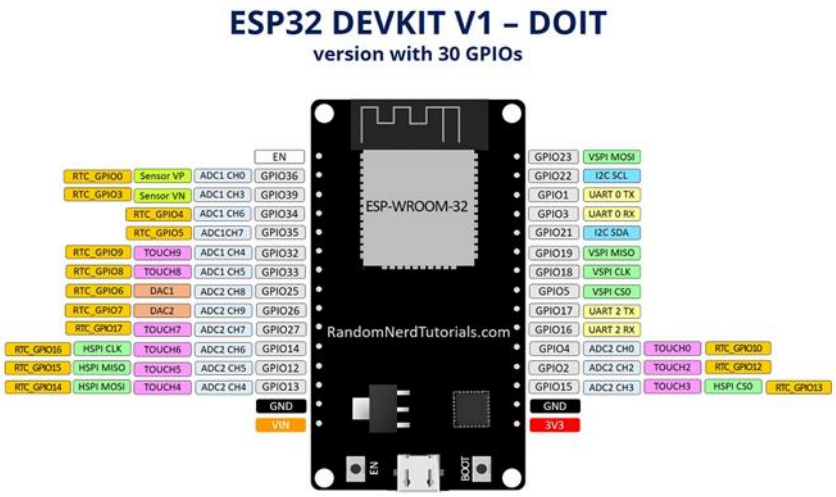

Figura 3.1 *Devkit V1* [11]

Una opción popular para aplicaciones de Internet de las cosas es el *NodeMCU ESP8266*, un módulo *SoC* como se muestra en la *Figura 3.2,* que destaca por sus capacidades de conexión en red *Wi-Fi*. Este módulo tiene un procesador de 32 bits que equilibra el rendimiento y la economía energética. Se basa en la arquitectura *Tensilica L106* y funciona a una frecuencia de 80 (MHz). Una conexión inalámbrica sólida y confiable es posible gracias a su compatibilidad con redes *Wi-Fi 802.11 b/g/n*, lo cual es crucial para las aplicaciones de Internet de las cosas que necesitan transmitir datos en tiempo real. El *ESP8266* también cuenta con una memoria *flash* de 4 (MB), lo que facilita el almacenamiento y ejecución de los datos y programas necesarios para una variedad de aplicaciones.

Se pueden configurar varios pines *GPIO* en el módulo para realizar diferentes tareas, incluidas *PWM, I2C, SPI y UART*, lo que le brinda la libertad de conectar una amplia gama de sensores y actuadores. El entorno de desarrollo integrado de *NodeMCU*, que está basado en *Lua* y permite una programación rápida y sencilla. La placa también tiene incorporado un convertidor de *USB* a serie, lo que facilita la programación y depuración del dispositivo. Debido a su pequeño tamaño y bajo consumo de energía, es perfecto para proyectos y aplicaciones portátiles que deben implementarse de manera rápida y efectiva [12].

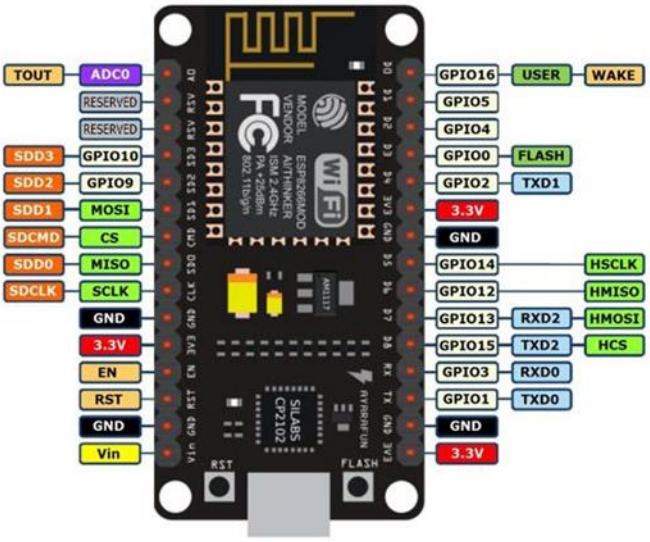

Figura 3.2 *NodeMCU ESP8266* [13]

Una de las placas de desarrollo más populares y adaptables en los campos de la electrónica y la programación es *Arduino UNO,* que se muestra en la *Figura 3.3*. Está construido alrededor del microcontrolador *ATmega328P*, que tiene una frecuencia de 16 (MHz) y un rendimiento apropiado para una variedad de proyectos educativos y de pasatiempos. La placa permite la conexión y control de una variedad de sensores y actuadores gracias a sus 14 pines de entrada/salidas digitales, seis de los cuales pueden usarse como salidas *PWM*, y sus seis entradas analógicas. Sus 2 (KB) *SRAM*, 1 (KB) *EEPROM* y 32 (KB) de memoria *flash* son suficientes para manejar datos en tiempo real y almacenar programas moderadamente grandes.

La accesibilidad de *Arduino UNO* para principiantes y su facilidad de uso son dos de sus ventajas más notables. El puerto *USB* de la placa, que se puede utilizar para alimentación y programación, facilita su integración con cualquier computadora. Escribir y cargar código se simplifica gracias al entorno de desarrollo *Arduino IDE*, que es fácil de usar y admite varios lenguajes de programación, incluidos *C y C++.* Además, un vasto ecosistema de bibliotecas y recursos brindan asistencia y ejemplos para casi cualquier tipo de proyecto. Además, *Arduino UNO* se puede utilizar con una amplia gama de módulos complementarios, lo que aumenta su funcionalidad y permite a los usuarios personalizar la placa para satisfacer necesidades particulares sin la necesidad de conocimientos electrónicos sofisticados [14].

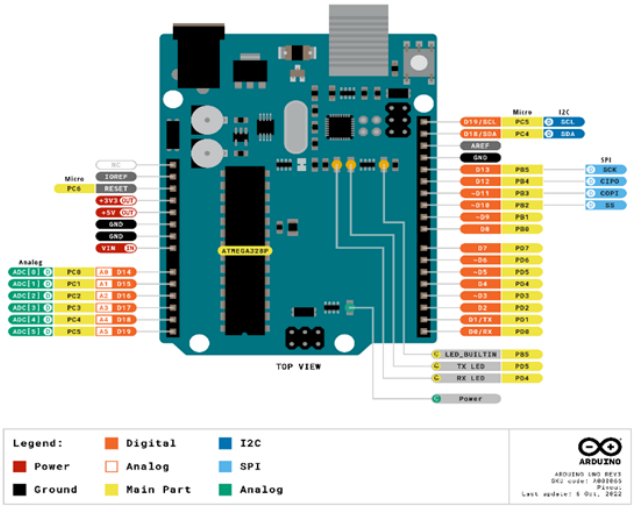

Figura 3.3 *Arduino UNO* [15]

El estudio y comparación de las características de la placa se realiza en base a los requerimientos específicos del proyecto como se presenta en la *Tabla 3.1*, teniendo en cuenta los requisitos básicos para la elección del microcontrolador necesario para el correcto montaje del prototipo.

Tabla 3.1 Comparación de los microcontroladores [16] [17] [18]

Componente	***ESP32 DEVKIT V1***	***NodeMCU ESP8266***	***Arduino UNO***
Wi-Fi	Integrado 802.11 b/g/n/ac	Integrado 802.11 b/g/n	No integrado
Bluetooth	Integrado (4.2 BLE)	No integrado	No integrado
Velocidad del Procesador (MHz)	240	80 - 160	16
Memoria *Flash*	4 (MB)	4 (MB)	32 (KB)
Memoria *RAM*	520 (KB)	80 (KB)	2 (KB)
Pines Analógicos	18	1	6
Pines Digitales	16	10	14
Interfaz *I2C*	Sí	Sí	Sí
Interfaz *SPI*	Sí	Sí	Sí
Interfaz *UART*	Sí	Sí	Sí
Dimensiones (mm)	55 x 28	57 x 33	73 x 53

Con base en el análisis de la *Tabla 3.1*, se concluye que la *ESP32* se diferencia de las demás placas al contar con características tecnológicas superiores que serán de utilidad para el desarrollo del proyecto. Estas características son las siguientes:

- En comparación con el *ESP8266 NodeMCU*, que solo admite *802.11 b/g/n*, el *ESP32 DEVKIT V1* tiene conectividad *Wi-Fi* incorporada que admite los estándares *802.11 b/g/n/ac*, lo que ofrece una velocidad y un alcance de conexión mejorados.
- *ESP8266 NodeMCU y Arduino UNO* carecen de *Bluetooth 4.2 BLE*, funcionalidad que tiene el *ESP32*. Debido a esto, el *ESP32* es la opción ideal para aplicaciones que necesitan multifuncionalidad y conectividad inalámbrica avanzada.
- Operando a 240 (MHz), el *ESP32* es mucho más rápido que el Arduino UNO (que sólo alcanza los 16 (MHz)) y el *NodeMCU ESP8266* (que abarca de (80 a 160 MHz)). El *ESP32* es perfecto para aplicaciones que exigen un alto rendimiento computacional debido a su procesador más rápido, que le permite realizar trabajos más complicados y procesar datos más rápido.
- El *ESP32 y el NodeMCU ESP8266* tienen 4 (MB) de memoria *flash* cada uno, pero el *Arduino UNO* sólo tiene 32 (KB). Con 520 (KB) de *RAM*, el *ESP32* funciona mejor que el *NodeMCU ESP8266*, que tiene 80 (KB), y el *Arduino UNO*, que tiene 2 (KB). La mayor capacidad de memoria del *ESP32* le brinda más versatilidad al crear proyectos complicados, ya que le permite ejecutar aplicaciones más grandes y manejar más datos en tiempo real.
- La conexión de una amplia gama de sensores y actuadores se simplifica gracias a los 18 pines analógicos y los 16 pines digitales del *ESP32*. Esto es significativamente más que el 1 pin analógico y los 10 pines digitales del *NodeMCU ESP8266*, así como los 6 pines analógicos y los 14 pines digitales del *Arduino UNO*. Con pines adicionales, el *ESP32* se puede ampliar para utilizarlo en proyectos que necesitan muchas entradas y salidas, lo que aumenta su versatilidad para una variedad de aplicaciones.
- El *ESP32* es compatible con interfaces *I2C, SPI y UART*, al igual que *ESP8266 NodeMCU y Arduino UNO*. Sin embargo, tiene una gran ventaja en términos de flexibilidad y funcionalidad debido a su mayor número de pines, su capacidad superior de procesamiento y memoria y su capacidad para manejar estas interfaces. La interfaz *I2C*, que proporciona el *ESP32*, es perfecta para facilitar la comunicación con memoria, sensores y otros dispositivos de baja velocidad.

Además, el *ESP32* es compatible con *SPI*, una interfaz de comunicación síncrona que utilizan principalmente periféricos de alta velocidad como tarjetas *SD*, monitores y otros dispositivos que necesitan una transferencia rápida de datos.

- El *ESP32* es apropiado para situaciones donde el espacio es una consideración crítica debido a su pequeño tamaño de 55 x 28 (mm). El *NodeMCU ESP8266 y ESP32* tienen un tamaño comparable (57 x 33 (mm)), sin embargo, el *ESP32* funciona mejor. El *Arduino UNO*, por el contrario, mide 73 x 53 (mm), lo que puede resultar problemático para proyectos con espacio limitado. Además, considerando sus características y capacidades sofisticadas, el *ESP32* tiene un precio atractivo de alrededor de $12, lo que proporciona un valor excepcional en comparación con los $8 del *NodeMCU ESP8266* y los $40 del *Arduino UNO*.

Selección de sensor

Se seleccionó el sensor *PIR* por encima de otros sensores alternativos debido a sus importantes beneficios para la gestión automatizada de la iluminación. En primer lugar, el sensor *PIR* utiliza muy poca energía, ya que puede identificar el movimiento analizando las variaciones en la radiación infrarroja que liberan los objetos en movimiento, lo que elimina la necesidad de una producción constante de señales. Esto lo hace perfecto para situaciones donde preservar la energía es esencial, como en dispositivos que funcionan con baterías o entornos donde el objetivo es minimizar el uso de electricidad.

Debido a la compatibilidad directa del *ESP32* y su facilidad de integración, se seleccionó el sensor *PIR* que se muestra en la *Figura 3.4*, para trabajar. Los sensores *PIR* son fáciles de conectar al *ESP32* y su interfaz digital facilita la lectura de datos de movimiento sin requerir *hardware* adicional complejo. Combinando la simplicidad y efectividad del sensor *PIR* con la capacidad de procesamiento del *ESP32*, puede construir de manera rápida y efectiva un sistema de control de iluminación automático [19].

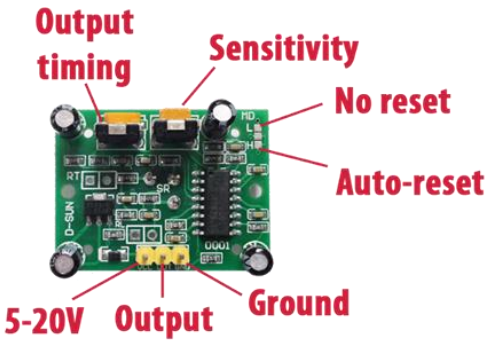

Figura 3.4 Sensor *PIR* [20]

Selección del *relé*

Debido a que el módulo de relé puede regular de forma segura y confiable cargas de alta potencia, fue elegido para este proyecto. Dentro de las aplicaciones de control de luminarias, un relé ofrece una interfaz confiable entre el microcontrolador y la carga, particularmente cuando se utilizan lámparas u otros equipos eléctricos con requisitos precisos de voltaje y corriente. El *ESP32* no puede controlar directamente cargas de alta potencia porque es un *SoC* con bajo voltaje y capacidades de corriente restringida en sus pines *GPIO*.

Como resultado, un módulo de relé que se muestra en la *Figura 3.5* sirve como un interruptor operado electrónicamente que puede encender o apagar las luces de forma segura, protegiendo el *SoC* y cualquier dispositivo asociado.

Además, el relé separa eléctricamente de forma efectiva la carga de la luminaria de la lógica de control *ESP32*. Al evitar interferencias electromagnéticas y proteger el microcontrolador de posibles sobrecargas, se promueve un funcionamiento más seguro y confiable [21].

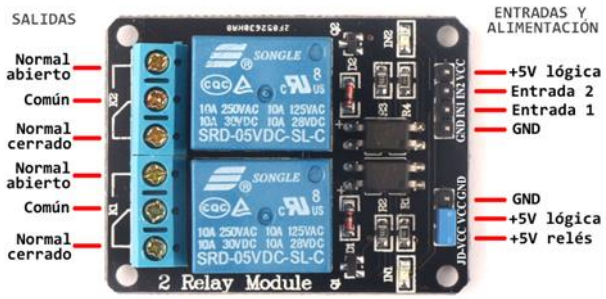

Figura 3.5 Modulo relé [22]

Determinación de la alimentación

El método óptimo para diversos escenarios y aplicaciones dependerá de una serie de parámetros, incluido si se debe alimentar el *SoC ESP32* a través del conector micro *USB* o los pines *VIN y GND*.

Es importante tener en cuenta que el microcontrolador *ESP32* está diseñado para funcionar normalmente con un voltaje de suministro de 3,3 (V) mientras se examinan los voltajes necesarios para alimentarlo. Sus partes internas, incluido el procesador y los periféricos asociados, dependen de este voltaje para funcionar correctamente. Se debe utilizar el regulador de voltaje interno del *ESP32* o un regulador externo para reducir el voltaje a niveles seguros y adecuados si se suministra un voltaje superior a 3,3 (V). Sin embargo, usar un voltaje inferior a 3,3 (V) para alimentar el *ESP32* podría provocar un

funcionamiento inestable o posiblemente impedir que el dispositivo se inicie correctamente.

La *Tabla 3.2* describe el voltaje y la corriente recomendadas por el fabricante para cada elemento del circuito.

Tabla 3.2 Voltajes y corrientes recomendadas por el fabricante [16] [17] [18]

Dispositivo	Voltaje recomendado	Corriente recomendada
***ESP32* Dev Kit V1**	7-12 (V)	500 (mA)
Sensor *PIR*	5-12 (V)	100 (mA)
Módulo de relé (estándar)	5V o 3.3 (V)	100 (mA) (por canal)

Selección del software

Se realizó un análisis exhaustivo para determinar qué plataformas serían mejores para implementar un proyecto que utiliza el microcontrolador *ESP32*. Durante este procedimiento, se evaluaron una serie de opciones de acuerdo con estándares que incluyen soporte técnico, escalabilidad, seguridad y facilidad de integración como se muestra en la *Tabla 3.3*. El objetivo principal fue encontrar la solución que mejor se ajuste a los requisitos únicos del proyecto y garantice una implementación confiable y efectiva. El análisis técnico se realizó entre las siguientes aplicaciones: *Arduino IoT Cloud y Arduino IDE.*

Tanto *Arduino IoT Cloud* como *Arduino IDE* son esenciales, pero tienen diferentes propósitos cuando se trata de desarrollar proyectos de *IoT*. Una plataforma en línea llamada *Arduino IoT Cloud* [23], permite crear, configurar y administrar dispositivos conectados a Internet de manera efectiva. Su principal ventaja es lo sencillo que es integrarlo con el entorno *Arduino*; proporciona una interfaz gráfica que facilita la configuración y el monitoreo de dispositivos. *Arduino IoT Cloud* se posiciona como una potente herramienta para proyectos que necesitan conectividad y monitoreo remoto continuo gracias a características como paneles de control configurables, notificaciones en tiempo real y la capacidad de administrar numerosos dispositivos desde una única plataforma.

El entorno de desarrollo integrado *Arduino* convencional, por otro lado, se llama *Arduino IDE* [24], y se utiliza para escribir, compilar y cargar código para la familia de microcontroladores *Arduino*. Con acceso directo a bibliotecas de *hardware y software*, los desarrolladores pueden escribir código *C/C++* en este entorno increíblemente

ampliable y flexible. La principal ventaja del *IDE de Arduino* es su capacidad para proporcionar un control completo del *hardware*, lo que permite una programación compleja y una optimización específica de los componentes del sistema. El *IDE* de *Arduino* también es esencial para crear prototipos complejos y únicos debido a su amplia selección de bibliotecas y su interoperabilidad con una amplia gama de sensores y actuadores.

A diferencia de la otra herramienta, *Arduino IoT Cloud* es más adecuada para aplicaciones que requieren monitoreo y control remotos debido a su énfasis en la conectividad y una administración más sencilla de dispositivos *IoT*. El *IDE* de *Arduino*, por otro lado, proporciona un mayor grado de control y personalización del código, lo cual es crucial para los desarrolladores que deben maximizar la optimización y el ajuste del *hardware*. La decisión entre estas dos aplicaciones dependerá de los requisitos particulares del proyecto: *Arduino IDE* es indispensable para un control complejo y una optimización sofisticada del código, mientras que *Arduino IoT Cloud* es la mejor opción para una implementación rápida y una gestión sencilla de los dispositivos conectados.

Tabla 3.3 Comparación entre *Arduino IoT Cloud y Arduino IDE [23] [24]*

Características	***Arduino IoT Cloud***	***Arduino IDE***
Facilidad de Uso	Alta, interfaz gráfica y configuraciones guiadas	Moderada, requiere conocimientos de programación en *C/C++*
Integración con *Hardware*	Excelente, especialmente con dispositivos *Arduino y ESP32*	Alta, compatible con una amplia gama de *hardware* Arduino
Interfaz de Usuario	Interfaz *web* amigable, paneles de control visuales	Interfaz de desarrollo de código, consola y editor de texto
Capacidades de Programación	Limitadas a configuraciones y automatizaciones predefinidas	Completa, permite programación detallada y control total sobre el *hardware*
Compatibilidad con *IoT*	Alta, diseñado específicamente para proyectos *IoT*	Moderada, requiere programación y configuración adicional para proyectos *IoT*

Características	*Arduino IoT Cloud*	*Arduino IDE*
Actualizaciones *OTA*	Sí, soporta actualizaciones Over-The-Air	No, requiere conexión física para cargar código

La elección de *Arduino IoT Cloud* sobre Arduino IDE se debe a su superior facilidad de uso, mejor integración con *hardware* específico, interfaz amigable para usuarios, mayor compatibilidad con proyectos *IoT*, y capacidad de realizar actualizaciones remotas. Estas características hacen que *Arduino IoT Cloud* sea la opción más adecuada para proyectos que requieren una implementación rápida y eficiente, así como una administración sencilla de dispositivos conectados.

3.3 Diseño del prototipo de control del encendido y apagado de las luminarias

Selección de la plataforma

Se realizó un análisis técnico para determinar qué plataformas de *IoT* serían mejores para implementar un proyecto que utiliza el microcontrolador *ESP32*. Durante este procedimiento, se evaluaron una serie de opciones de acuerdo con estándares que incluyen soporte técnico, escalabilidad, seguridad y facilidad de integración como se muestra en la *Tabla 3.4*. El objetivo principal fue encontrar la solución que mejor se ajuste a los requisitos únicos del proyecto y garantice una implementación confiable y efectiva. Se espera que la investigación brinde a los desarrolladores e implementadores del sistema *IoT* basado en *ESP32* una base sólida sobre la cual tomar decisiones durante el desarrollo y la implementación del sistema. El análisis técnico se realizó entre las siguientes aplicaciones: *Arduino IoT Cloud* [23] *y ThingSpeak* [25].

ThingSpeak y Arduino IoT Cloud son dos plataformas conocidas en el campo del Internet de las cosas que brindan sólidas capacidades para organizar y analizar datos de dispositivos vinculados. La integración de *Arduino IoT Cloud* con *hardware* de la familia *Arduino* y su interfaz fácil de usar para desarrolladores de todos los niveles son sus características primordiales. A través de un diseño basado en la nube, la plataforma facilita la configuración y el monitoreo del dispositivo, lo que permite a los usuarios crear paneles personalizados y recibir notificaciones en tiempo real. Su gran red de soporte y compatibilidad con el lenguaje de programación *C++* lo convierten en una opción flexible y asequible para proyectos que van desde la producción hasta la creación de prototipos.

Sin embargo, *ThingSpeak* destaca por tener funciones sofisticadas de análisis de datos y estar integrado con *MATLAB*, lo que lo hace especialmente atractivo para aplicaciones que necesitan procesamiento de datos detallado y análisis predictivo. *ThingSpeak* hace posible el almacenamiento y la visualización de datos en tiempo real, y se puede interactuar con una amplia gama de dispositivos y aplicaciones gracias a su *API RESTful* flexible. Además, la plataforma proporciona herramientas analíticas sólidas que permiten a los usuarios crear algoritmos personalizados y modelos de aprendizaje automático, lo que permite una toma de decisiones bien informada basada en los datos recopilados.

Al comparar las dos plataformas, *Arduino IoT Cloud* es más fácil de usar y se configura más rápidamente, lo que la convierte en la mejor opción para proyectos que buscan una implementación sencilla y eficiente. Por otro lado, *ThingSpeak* se posiciona como una opción más potente para tareas que requieren predicción sofisticada y procesamiento de datos complejo. Los requisitos únicos del proyecto determinarán cuál de estas plataformas es mejor, teniendo en cuenta aspectos como el nivel de análisis de datos necesario, lo sencillo que es integrarlo con el *hardware* y la capacidad de monitorear datos en tiempo real. Aunque ambas plataformas ofrecen buenas soluciones, son más apropiadas para ciertos tipos de aplicaciones de Internet de las cosas debido a sus respectivas ventajas.

Tabla 3.4 Comparación entre *Arduino IoT Cloud y ThingSpeak* [23] [25]

Características	***Arduino IoT Cloud***	***ThingSpeak***
Facilidad de Uso	Muy alta, interfaz intuitiva y amigable	Alta, pero requiere conocimientos de *MATLAB* para análisis avanzado
Integración con *Hardware*	Excelente, especialmente con dispositivos *Arduino y ESP32*	Buena, compatible con una amplia variedad de dispositivos *IoT*
Interfaz de Usuario	Interfaz *web* y aplicación móvil amigables	Interfaz *web* robusta, sin aplicación móvil oficial
Notificaciones en Tiempo Real	Sí, configurable desde la plataforma	Limitadas, depende de *scripts* personalizados

Características	***Arduino IoT Cloud***	***ThingSpeak***
Compatibilidad con *Telegram*	Integración sencilla mediante *API*	Requiere más configuración, posible mediante *scripts* personalizados
Compatibilidad con *Android*	Aplicación oficial de Arduino *IoT*	No tiene aplicación oficial, integración mediante *API*

Se seleccionó *Arduino IoT Cloud* para el desarrollo del prototipo debido a su excepcional usabilidad y facilidad de integración, particularmente cuando se combina con *hardware ESP32*. Una interfaz fácil de usar como la que ofrece *Arduino IoT Cloud* como se muestra en la *Figura 3.6,* que permite configurar y administrar dispositivos rápidamente, lo cual es esencial para la creación de prototipos, que requiere una ejecución fluida y efectiva. También es sencillo monitorear e interactuar con el sistema desde cualquier dispositivo gracias a la capacidad de crear paneles personalizados y recibir notificaciones en tiempo real. Una curva de aprendizaje sencilla y acceso a soluciones para cualquier problema técnico imprevisto son garantías adicionales proporcionadas por la extensa comunidad de soporte y recursos disponibles para *Arduino*, lo que hace de *Arduino IoT Cloud* la opción perfecta para este complejo proyecto.

Figura 3.6 *Arduino IoT Cloud* [26]

Diseño esquemático del sistema

El diseño del prototipo del sistema de control automática de luminarias se basa en un circuito complejo que contiene muchos elementos diferentes como se muestra en la *Figura 3.7*. Los elementos que están incluidos en este esquema es la interfaz *web,* la aplicación *Android* y el *bot* de *Telegram,* que son los sistemas con los que el usuario tendrá interacción. La plataforma *Arduino IoT* es la plataforma encargada de recibir los datos que serán enviados desde los dispositivos. El *ESP32* es el orquestador principal, este se encarga de llevar a cabo las tareas y estará a cargo de recompilar la información.

El sensor *PIR* es el encargado de detectar la presencia para el encendido y apagado. El *relé* actuara como interruptor de forma remota para el encendido y apagado de las luminarias.

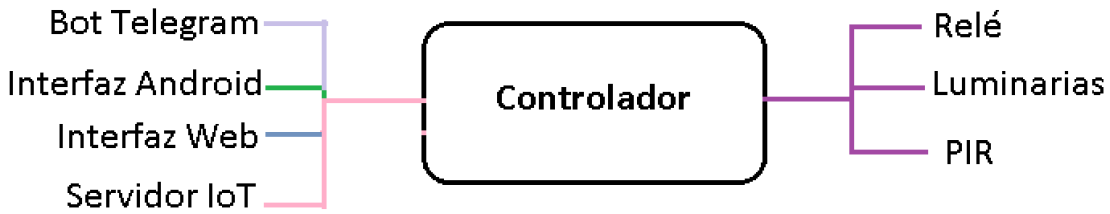

Figura 3.7 Esquema del sistema de control de luminarias

La *Figura 3.8* muestra un esquema detallado del prototipo de control de luminarias, acompañado de una lista de los componentes que conforman el sistema. El usuario puede interactuar con el sistema a través de una interfaz *web* accesible desde cualquier navegador o mediante una aplicación para teléfonos inteligentes *Android*. La administración y almacenamiento eficiente de los datos transferidos entre el usuario y el prototipo de control de iluminación se alojan en el servidor *Arduino IoT Cloud*.

Tanto el *ESP32* como el servidor están conectados a Internet a través del módem *Wi-Fi*. Un servidor en la nube que centraliza la gestión y el control del sistema; en este caso, utilizando la plataforma *Arduino IoT Cloud*. Recibe comandos de las distintas interfaces de usuario (*Android, web y Telegram*) y envía instrucciones al *hardware* conectado.

ESP32 Devkit V1 es una plataforma de desarrollo construida sobre el microcontrolador *ESP32* que utiliza un módem *Wi-Fi* para recibir instrucciones del servidor en la nube de *IoT*. Regule el *relé* y reciba señales del sensor *PIR*. El sensor *PIR* es un sensor de movimiento que utiliza tecnología de infrarrojos pasivos para identificar la presencia de personas. En función del movimiento observado, envía señales al *ESP32 Devkit V1* para encender o apagar las luminarias. Las Luminarias se pueden encender o apagar mediante el *relé*, un interruptor electromecánico que es gestionado por el *ESP32 Devkit V1*. El sistema regula las luminarias. Responden a las señales del *relé* encendiéndose o apagándose.

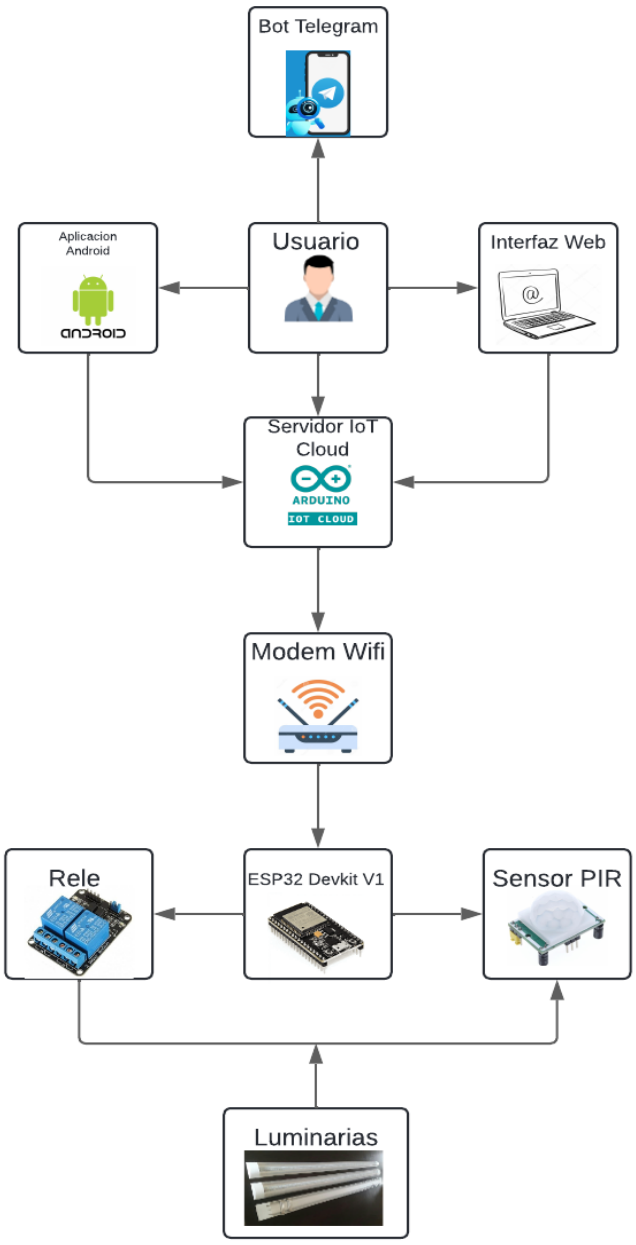

Figura 3.8 Elementos del sistema

Creación de códigos

Se accedió a la plataforma *Arduino IoT Cloud* ingresando el usuario y la contraseña antes de realizar cualquier programación. Luego se conectó el dispositivo *ESP32* y se registró en la sección Dispositivos de la plataforma. Los pasos que necesita para la vinculación y el registro se muestran a continuación.

Conexión del *ESP32* con la plataforma *Arduino IoT Cloud*

Primero se inició sesión en el sitio *web* de *Arduino IoT Cloud* utilizando las credenciales de inicio de sesión para conectar el *ESP32* a la plataforma. Luego de ingresar, se dirige al área de "Dispositivos" y se agrega allí un nuevo dispositivo como se muestra en la *Figura 3.9*. Luego de elegir la opción de agregar un *ESP32*, se sigue paso a paso el asistente para finalizar el emparejamiento. Este procedimiento implica utilizar un cable *USB* para conectar el *ESP32* a la computadora, elegir el modelo de placa preciso y dejar que la plataforma reconozca y configure el dispositivo automáticamente.

El proyecto o colección de variables y atributos que se desea controlar y monitorear está representado por una *"Thing*" que se produce en *Arduino IoT Cloud* una vez que el *ESP32* ha sido registrado y conectado a la plataforma. Ahora se pueden definir las variables que se sincronizarán entre el *ESP32* y la nube. Estas variables están configuradas para enviar y recibir datos entre el dispositivo y la plataforma. Pueden ser de varios tipos, incluidas cadenas de texto, números enteros y booleanos. Se debe asignar un nombre y una descripción del proyecto para que sea más fácil identificar la "*thing*" una vez configurada.

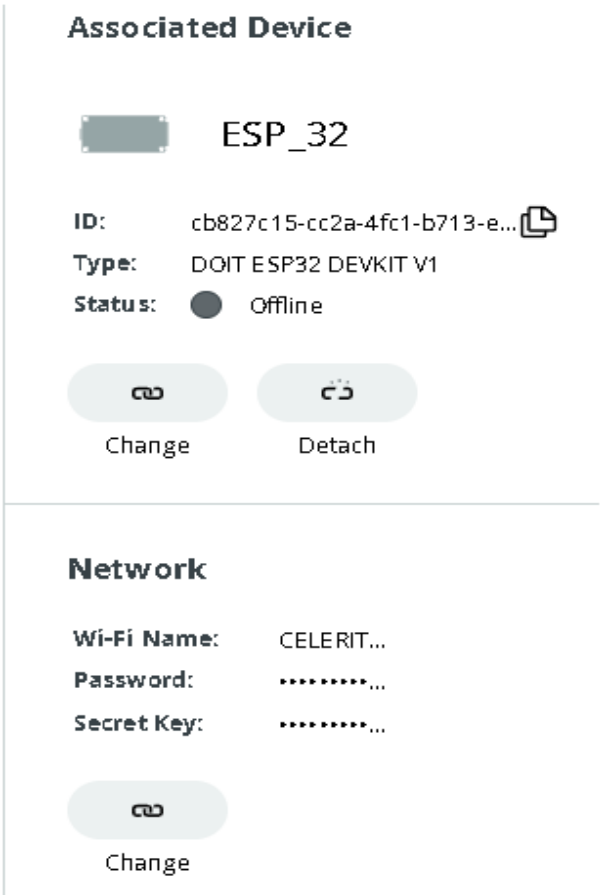

Figura 3.9 Integración del *ESP32* con la plataforma *Arduino IoT Cloud*

Implementación de variables

Al finalizar la integración del *ESP32* con la plataforma *Arduino IoT Cloud*, es necesaria la creación de las variables a utilizar en el desarrollo del código. Como paso anterior al desarrollo del código, es importante crear las variables dentro de la pestaña de *Cloud Variables* como se presenta en la *Figura 3.10.* Para el desarrollo del código, se crearon cinco variables.

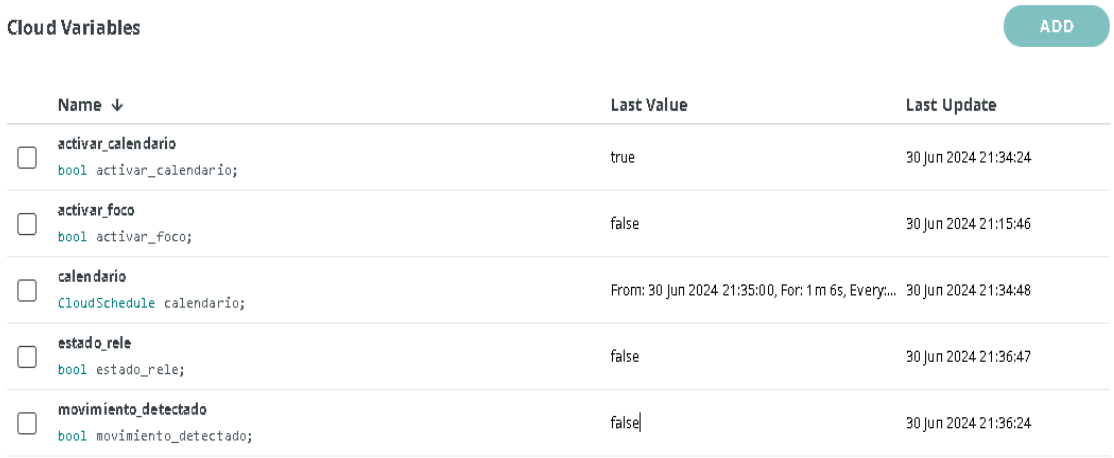

Figura 3.10 Variables integradas en la plataforma *Arduino IoT Cloud*

La segunda variable, denominada estado_rele, de tipo *booleana*, es aquella que se encarga de indicar el estado del relé*, esto* dependiendo de las señales receptadas por el sensor *PIR* y por activada o desactivado por el control manual. La tercera variable, llamada *activar_calendario*, es la encargada de activar la programación de horarios. La variable *calendario*, que es de tipo *CloudSchedule*, es una variable propia del entorno del *Arduino IoT Cloud*, esta variable es la encargada de programar horarios dependiendo de las condiciones que programa el usuario. La variable *activar_foco*, de tipo *booleano*, es la encargada de la activación manual del *relé*.

Acoplar el *SoC ESP32*, realizar la conexión del dispositivo a la red *Wi-Fi* y la creación de las variables, representan un punto fundamental para poder proceder con la creación del código en el *Sketch* que se encuentra en la plataforma. El desarrollo del código se realizó utilizando el lenguaje *C++* que se encuentra integrado en la plataforma *Arduino IoT Cloud*, este código está dividido en 4 secciones. La primera sección es la encargada de declarar las librerías necesarias para el funcionamiento correcto del código, están declarados los pines con su respectivo identificador, diferentes variables para el control y variables para el tiempo, cuenta también con variables para controlar los horarios de la plataforma *Telegram* y la configuración del cliente *NTP*.

La segunda parte está compuesta por la función *void setup (),* antes de que el programa principal comience a ejecutarse, se utiliza en el entorno de programación *Arduino* para inicializar las configuraciones requeridas, llamado solo una vez al iniciar el programa, es la ubicación adecuada para configurar temporizadores, inicializar la comunicación en serie, configurar los pines de entrada y salida y realizar cualquier otra configuración necesaria antes de ingresar al ciclo principal del programa. En la tercera parte, se ubica la función *void loop (),* esta función es importante porque tiene un código que, una vez finalizada la configuración inicial en *void setup ()*, se ejecuta repetida y continuamente.

Es donde se pone en práctica la lógica principal (como el control de actuadores, el manejo de eventos y las lecturas de sensores) que reacciona a los cambios ambientales. En la sección final del código, es donde se encuentra todas las líneas de código definidas para cada función. *Cloud Variables*, crea de forma automática diferentes funciones con los nombres respectivos a cada variable que se han creado en la sección.

Librerías y variables

Se utilizan varias bibliotecas cruciales en el proyecto que se muestran en la *Figura 3.11* para controlar automáticamente luminarias con un *ESP32* en *Arduino IoT Cloud*. Para el control remoto y la comunicación, el *ESP32* debe poder conectarse a una red *Wi-Fi*, lo cual es posible gracias a la biblioteca <*WiFi.h*>, <*WiFiClientSecure.h*> que agrega una capa adicional de seguridad a las conexiones de red, garantizando la seguridad de los datos transferidos. La biblioteca <*UniversalTelegramBot*La biblioteca <*UniversalTelegramBot.h*>, facilita la integración con *Telegram*, por lo que el sistema puede comunicarse con esta plataforma para enviar y recibir mensajes. Además, para obtener la hora actual de un servidor de protocolo de tiempo de red (NTP) y programar eventos y actividades basados en el tiempo, utiliza <NTPClient.h> y <WiFiUdp.h>.

En cuanto a las variables que se presentan en la *Figura 3.12*, puede planificar acciones y eventos desde el panel de la nube utilizando la instancia <*CloudSchedule calendario*>. El estado del sensor *PIR,* que detecta el movimiento, se mantiene en <movimiento_detectado>. Las luminarias están controladas por el *relé*, cuyo estado se guarda mediante <estado_rele>. Los estados de los conmutadores operados manualmente y controlados en la nube se almacenan en las variables <activar_foco> respectivamente. Estos factores facilitan la comunicación entre el *hardware* y el *software*, lo que permite un control eficaz y automatizado del sistema de iluminación.

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
```

Figura 3.11 Bibliotecas

```
CloudSchedule calendario;
bool activar_calendario;
bool activar_foco;
bool estado_rele;
bool movimiento_detectado;
```

Figura 3.12 Variables

Void setup ()

La función *void setup ()* se encarga de inicializar y configurar el entorno requerido para que el *ESP32* funcione correctamente. Al utilizar *Serial.begin*(9600), primero se establece la conectividad en serie, lo que permite la depuración mediante la interacción con el monitor en serie. A continuación, se utiliza *delay* (1500), para definir un retraso de 1,5 segundos para esperar a que se conecte el monitor serie. Para inicializar las propiedades definidas en el archivo *thingProperties.h*, se invoca el método *initProperties ()*. Luego, *ArduinoCloud.begin(ArduinoIoTPreferredConnection)* se utiliza para conectarse a *Arduino IoT Cloud*, lo que permite la administración remota de dispositivos basada en la nube. Usando *setDebugMessageLevel (2)*, se puede ajustar el nivel de depuración y obtener información completa sobre la condición de la red y la conexión a la nube de *IoT*. En última instancia, *ArduinoCloud.printDebugInfo()* se utiliza para imprimir mensajes de depuración.

Estas líneas definen las variables necesarias para la conexión del *bot* de *Telegram*. *bot*_token es una cadena de caracteres constante que contiene el token de autenticación del *bot*, necesario para interactuar con la API de *Telegram*. Este token no debe modificarse, ya que es específico para el *bot* en cuestión. chat_id_aux es una cadena auxiliar que guarda un ID de chat, mientras que chat_id es una cadena que probablemente será utilizada más adelante en el código para almacenar dinámicamente el ID de chat con el que se comunicará el *bot*.

WiFiClientSecure client; declara un cliente seguro de *Wi-Fi*, que permite la comunicación segura a través de HTTPS. UniversalBot bot (bot_token, client); instancia un objeto de

tipo UniversalTelegramBot, que utiliza el _token y el client para realizar solicitudes a la API de *Telegram*. El código se detalla en la *Figura 3.13.*

```
// Variables para el bot de telegram
const char* bot_token = "7417029630:AAHAQ2KAApvnVSReLn8s9abaf-UYfZD2gzs";  // NO modificar
const String chat_id_aux = "1324656447";
String chat_id;

//Variables conexion a bot de telegram
WiFiClientSecure client;
UniversalTelegramBot bot(bot_token, client);
```

Figura 3.13 Conexión a *Telegram*

La configuración de los pines del *ESP32* que se utilizaran en el proyecto corresponden a la siguiente distribución: relé está conectado al pin 14 y se utiliza para controlar un relé, botón está conectado al pin 35 y se utilizará para detectar pulsaciones de un botón que permite el control manual, el *PIR* está conectado al pin 27 y se utilizará para detectar movimiento a través de un sensor *PIR*. La definición de estos pines se detalla en la *Figura 3.14.*

```
// Pines del ESP32
const int rele = 14;
const int boton = 35;
const int pir = 27;
```

Figura 3.14 Definición de los pines

Estas líneas definen variables relacionadas con el tiempo de ejecución del *bot*, botRequestDelay se establece en 1000 milisegundos, lo que indica que el *bot* verificará nuevas actualizaciones cada segundo, lastTimeBotRan es una variable sin signo de tipo unsigned long que se utilizará para almacenar el momento en que el *bot* se ejecutó por última vez. Las variables de tiempo utilizadas se expresan en la *Figura 3.15*.

```
// Variables de tiempo
int botRequestDelay = 1000;  //verificar bot cada 1000ms
unsigned long lastTimeBotRan;
```

Figura 3.15 Variables de tiempo

Se declaran varias variables booleanas que se utilizan para controlar diferentes estados del programa, aux y aux2 son variables auxiliares, scheduleActive indica si el calendario de *Telegram* está activo, detecto_movimiento y boton_presionado rastrean si se ha detectado movimiento o si el botón ha sido presionado, respectivamente, mov es una

variable auxiliar para el control del sensor *PIR*, txmsg_state indica si los mensajes están listos para ser enviados, cumple_hora y porHorario se utilizan para gestionar el cumplimiento de los horarios establecidos. En la *Figura 3.16* se indica la declaración de las variables.

```
// Variables booleanas
bool aux = false;
bool aux2 = true;
bool scheduleActive = false;  //verificar si calendario esta activo o no en Telegram
bool detecto_movimiento = false;
bool boton_presionado = false;
bool mov = false;         //aux para control de pir
bool txmsg_state = true;  //aux enviar mensajes

bool cumple_hora = false;
bool porHorario = false;
```

Figura 3.16 Variables *booleanas*

Estas líneas configuran el cliente *NTP* (Network Time Protocol) para sincronizar la hora. WiFiUDP ntpUDP; crea un objeto *UDP* para la comunicación *NTP*. NTPClient timeClient(ntpUDP, "europe.pool.ntp.org", -18000, 60000); crea un cliente *NTP* que se sincroniza con el servidor "europe.pool.ntp.org" cada 60 segundos. El desplazamiento de tiempo se establece en -18000 segundos para ajustar a la zona horaria UTC-5. La *Figura 3.17* indica la configuración del cliente *NTP*.

```
// Configuración del cliente NTP
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "europe.pool.ntp.org", -18000, 60000);  // Sincronización cada 60 segundos UTC-5
```

Figura 3.17 Configuración cliente *NTP*

Las siguientes variables permiten almacenar los horarios de inicio y fin para la operación programada del *bot* de *Telegram*, startHour y startMinute almacenan la hora y el minuto de inicio, mientras que endHour y endMinute almacenan la hora y el minuto de finalización. La *Figura 3.18* indica las variables de tiempo para *Telegram*.

```
// Variables de horario de telegram
int startHour = 0;
int startMinute = 0;
int endHour = 0;
int endMinute = 0;
```

Figura 3.18 Variables de horario para *Telegram*

La función setup() se ejecuta una vez al inicio del programa y se utiliza para inicializar configuraciones y conexiones. Serial.begin(115200) inicia la comunicación serial a

115200 baudios, útil para la depuración. initProperties() se encargar de inicializar propiedades definidas en otro archivo (thingProperties.h) que es un archivo propio que crea *Arduino IoT Cloud*. ArduinoCloud.begin(ArduinoIoTPreferredConnection) conecta el dispositivo a la nube de Arduino IoT, configurando el nivel de mensajes de depuración y mostrando información de depuración. client.setCACert(TELEGRAM_CERTIFICATE_ROOT) se encarga de establecer el certificado raíz necesario para la comunicación segura con Telegram.

Finalmente, se configuran los pines físicos del *ESP32*: pinMode(rele, OUTPUT), establece el pin del relé como salida, pinMode(boton, INPUT), establece el pin del botón como entrada, y pinMode(*PIR*, INPUT), establece el pin del sensor *PIR* como entrada. digitalWrite(rele, LOW), asegura que el relé esté inicialmente apagado. El detalle del código se encuentra en la *Figura 3.19.*

```
// configuracion
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(115200);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  //delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  client.setCACert(TELEGRAM_CERTIFICATE_ROOT);

  // configuración física
  pinMode(rele, OUTPUT);
  pinMode(boton, INPUT);
  pinMode(pir, INPUT);
  digitalWrite(rele, LOW);
}
```

Figura 3.19 *Void setup()*

Void loop()

Una función importante que se ejecuta constantemente en un bucle infinito en los proyectos *Arduino* es la función `*void loop()*`. Para mantener la sincronización del estado del dispositivo con *Arduino IoT Cloud*, la función en este ejemplo se inicia con una llamada a *ArduinoCloud.update()*. Esta actualización podría leer datos de sensores o realizar operaciones en respuesta a instrucciones de la nube. Si el dispositivo está conectado correctamente (ArduinoCloud.connected() devuelve verdadero), la variable aux se establece en true, lo que indica que se ha establecido la primera conexión. La inicialización de la función se detalla en la *Figura 3.20*.

```
// bucle principal
void loop() {

  //primera conexion
  ArduinoCloud.update();
  if (ArduinoCloud.connected()) {
    aux = true;
  }
```

Figura 3.20 *Void loop()*

Dentro del bucle while (aux), se verifica si aux2 es verdadero y si el *Wi-Fi* está conectado (WiFi.status() == WL_CONNECTED). Si ambas condiciones se cumplen, se inicia el cliente NTP (timeClient.begin()) para sincronizar la hora y se establece aux2 a false para evitar iniciar el cliente *NTP* nuevamente. Se actualiza la conexión a la nube de Arduino con ArduinoCloud.update(). Además, se lee el estado del sensor *PIR* (digitalRead(*PIR*)) para detectar movimiento y se establece la variable detecto_movimiento en consecuencia. El detalle del código se indica en la *Figura 3.21*.

```
while (aux) {
  if (aux2 && WiFi.status() == WL_CONNECTED) {
    timeClient.begin();
    aux2 = false;
  }
  ArduinoCloud.update();
  //Serial.println("conectado");


  if (digitalRead(pir)) {
    detecto_movimiento = true;
  } else {
    detecto_movimiento = false;
  }
```

Figura 3.21 Verificación de conexión *Wi-Fi*

Funciones con códigos

Se actualiza el cliente *NTP* (timeClient.update()) y se llama a la función handleSchedule(), que gestiona un calendario configurado a través de *Telegram*. Si el calendario está activado (activar_calendario) y no está activo en *Telegram* (!scheduleActive), se verifica el estado del calendario (calendario.isActive()). Si el calendario está activo, se establecen las variables cumple_hora y porHorario en true. Si porHorario es true, pero cumple_hora es false y el relé está encendido (digitalRead(rele)), se envía un mensaje a través de *Telegram* notificando que la luz está encendida fuera del horario, se apaga el relé, y se establece porHorario en false. El manejo del horario se detalla en la *Figura 3.22*.

```
// manejo del scheduler con telegram
timeClient.update();
handleSchedule();  //verificar calendario configurado en Telegram

// manejo del scheduler con el widget arduino cloud
if (activar_calendario && !scheduleActive) {  // solo se activa si el switch esta en ON y el scheduler del telegram está en /stopschedule
  if (calendario.isActive()) {
    //digitalWrite(rele, HIGH);  // enciende las luminarias si está en dentro del horario
    cumple_hora = true;
    porHorario = true;
  } else {
    // if (!detecto_movimiento) {  // apaga las luminarias fuera del horario solo si el pir no ha detectado movimiento
    //   digitalWrite(rele, LOW);
    // }
    cumple_hora = false;
  }
  if (porHorario && !cumple_hora && digitalRead(rele)) {
    bot.sendMessage(chat_id_aux, "Luz encendida fuera de horario, apagando…", "");
    delay(2000);
    digitalWrite(rele, LOW);
    porHorario = false;
  }
}
```

Figura 3.22 Manejo del horario por *Telegram*

La función control_*PIR*() se llama para manejar la lógica del sensor *PIR*. La función handleTelegramBot() gestiona las interacciones con el *bot* de *Telegram*. Luego, se maneja el estado del botón físico conectado al *ESP32*. Si se detecta que el botón está presionado (digitalRead(boton)) y el relé está apagado (!digitalRead(rele)), se enciende el relé (digitalWrite(rele, HIGH)). Si el botón está presionado y el relé está encendido (digitalRead(rele)), se apaga el relé (digitalWrite(rele, LOW)). El código en la *Figura 3.23* expresa a detalle el funcionamiento.

```
control_pir();

// función para el manejo del bot de telegram
handleTelegramBot();

// manejo del toggle del button fisico
if (digitalRead(boton) && !digitalRead(rele)) {
  digitalWrite(rele, HIGH);
  //Serial.println("fisico");
} else if (digitalRead(boton) && digitalRead(rele)) {
  digitalWrite(rele, LOW);
}
```

Figura 3.23 Manejo del *toggle* del botón físico

El estado del botón en la nube (activar_foco) también se verifica. Si está activado y el relé está apagado, se enciende el relé. Si está activado y el relé está encendido, se apaga el relé. Los indicadores para el tablero de control se actualizan con el estado del relé (estado_rele) y la detección de movimiento (movimiento_detectado). En la *Figura 3.24* se indica el código a detalle.

```
// manejo del toggle del button cloud
if (activar_foco && !digitalRead(rele)) {
  digitalWrite(rele, HIGH);
} else if (activar_foco && digitalRead(rele)) {
  digitalWrite(rele, LOW);
}

// manejo de indicadores en el dashboard
estado_rele = digitalRead(rele);
movimiento_detectado = digitalRead(pir);
```

Figura 3.24 Manejo del *toggle* del botón *cloud* e indicadores del *dashboard*

Finalmente, se manejan las notificaciones de *Telegram*. Si el relé está encendido (digitalRead(rele)) y txmsg_state es true, se envía un mensaje indicando que las luminarias están encendidas y se establece txmsg_state en false. Si el relé está apagado y txmsg_state es false, se envía un mensaje indicando que las luminarias están apagadas y se establece txmsg_state en true. Se añade un pequeño retraso (delay(10)) para evitar sobrecargar el bucle y luego se cierra el bucle while (aux) con otro retraso de 10 milisegundos. La *Figura 3.25* expresa a detalle el código.

```
  // manejo de notificaciones por telegram
  if (digitalRead(rele) && txmsg_state) {  // envia un mensaje solo una vez por cada cambio de estado de las luminarias
    bot.sendMessage(chat_id_aux, "Luminarias Encendidas", "");
    txmsg_state = false;
  } else if (!digitalRead(rele) && !txmsg_state) {
    bot.sendMessage(chat_id_aux, "Luminarias Apagadas", "");
    txmsg_state = true;
  }
  delay(10);
}  //cierra while
delay(10);
```

Figura 3.25 Manejo de las notificaciones por *Telegram*

La función `control_*PIR*()` se encarga de gestionar el comportamiento del relé en función del estado del sensor *PIR* y la variable `cumple_hora`. Esta variable indica si el horario configurado permite el funcionamiento del sistema. Dentro de la función, se evalúa primero si `cumple_hora` es verdadero, lo que permite que el sistema interactúe con el sensor *PIR*.

Si se detecta movimiento (`digitalRead(*PIR*)`) y el relé está apagado (`!digitalRead(rele)`) y la variable `mov` es falsa, se enciende el relé (`digitalWrite(rele, HIGH) `). Esto sugiere que, al detectar movimiento, se activa el relé, para encender una luminaria. Si no se detecta movimiento (`!digitalRead(*PIR*)`) y el relé está encendido, se

establece `mov` en verdadero, lo que parece ser una señal de que se ha registrado un cambio en el estado del sensor *PIR*.

Luego, si se detecta movimiento y el relé está encendido, y además `mov` es verdadero, se apaga el relé (`digitalWrite(rele, LOW) `), lo que implica un mecanismo de temporización o control para evitar que la luz permanezca encendida continuamente. Finalmente, si no se detecta movimiento y el relé está apagado, se restablece `mov` a falso, reiniciando el ciclo de control del sensor *PIR*. Esta lógica permite una gestión eficiente del relé en función de la detección de movimiento y el horario permitido. La *Figura 3.26* indica el código correspondiente al funcionamiento del sensor *PIR*.

```
void control_pir() {
  if (cumple_hora) {
    // manejo del sensor PIR
    if (digitalRead(pir) && !digitalRead(rele) && !mov) {
      digitalWrite(rele, HIGH);
    }
    if (!digitalRead(pir) && digitalRead(rele)) {
      mov = true;
    }
    if (digitalRead(pir) && digitalRead(rele) && mov) {
      digitalWrite(rele, LOW);
    }
    if (!digitalRead(pir) && !digitalRead(rele)) {
      mov = false;
    }
  }
}
```

Figura 3.26 Manejo del sensor *PIR*

La función `handleTelegramBot()` gestiona los mensajes entrantes del *bot* de *Telegram*. Primero, verifica si ha transcurrido suficiente tiempo desde la última ejecución utilizando `millis()`, que devuelve el tiempo en milisegundos desde que el programa comenzó a ejecutarse. Si el tiempo actual (`millis()`) es mayor que la suma de `lastTimeBotRan` y `botRequestDelay`, procede a buscar nuevos mensajes. La variable `lastTimeBotRan` se utiliza para almacenar el tiempo en que se realizó la última verificación de mensajes, y `botRequestDelay` define el intervalo en milisegundos entre cada verificación.

A continuación, `int numNewMessages = bot.getUpdates(bot.last_message_received + 1); ` llama al método `getUpdates()` del objeto `bot` para obtener nuevos mensajes a partir del último mensaje recibido, almacenando la cantidad de nuevos mensajes en `numNewMessages`. Si hay nuevos mensajes, el programa entra en un bucle `while` que continúa mientras `numNewMessages` sea distinto de cero. Dentro de este bucle, se imprime un mensaje en el monitor serial indicando que se ha recibido un mensaje.

La función `handleNewMessage(numNewMessages); ` se llama para manejar los nuevos mensajes recibidos, ejecutando la lógica correspondiente para cada mensaje o comando de *Telegram*. Después de procesar los mensajes, se vuelve a llamar a `.getUpdates(.last_message_received + 1); ` para actualizar el número de nuevos mensajes pendientes de recibir. Una vez que no quedan más mensajes nuevos, se actualiza `lastTimeBotRan` con el valor actual de `millis()`, marcando el momento en que se realizó la última verificación de mensajes. Esta función asegura que el *bot* de *Telegram* maneje los mensajes entrantes de manera eficiente y periódica. La *Figura 3.27* describe el código para el manejo de mensajes por *Telegram*.

```
// función para el manejo de mensajes entrantes del bot de telegram
void handleTelegramBot() {
  if (millis() > lastTimeBotRan + botRequestDelay) {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    while (numNewMessages) {
      Serial.println("se recibe un mensaje");

      handleNewMessage(numNewMessages);  // funcion para manejar los comandos de telegram
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
  }
}
```

Figura 3.27 Manejo de los mensajes de *Telegram*

La función `handleNewMessage(int messageIndex)` se encarga de procesar los comandos recibidos a través del *bot* de *Telegram*. Esta función itera sobre cada mensaje recibido hasta el índice especificado por `messageIndex`, utilizando un bucle `for`. En cada iteración, primero extrae el `chat_id` y el texto del mensaje (`text`) de la lista de mensajes del *bot*.

Cuando se recibe el comando `/iniciar`, se construye un mensaje (`sms`) que contiene una descripción del sistema y un menú con las opciones disponibles. Este mensaje se envía al usuario a través de `*bot*.sendMessage(chat_id, sms, "") `. El código de la *Figura 3.28* indica a detalle el funcionamiento.

```
// funcion para manejar comandos de telegram
void handleNewMessage(int messageIndex) {

  for (int i = 0; i < messageIndex; i++) {
    chat_id = String(bot.messages[i].chat_id);
    String text = bot.messages[i].text;

    if (text == "/iniciar") {
      String sms = "Control luminaria mediante PIR y horarios\n";
      sms += "==== MENU DE OPCIONES ====\n\n";
      sms += "/proginiciar HH:MM, hora de encendido de luminarias \n";
      sms += "/progfinal HH:MM, hora de apagado de luminarias \n";
      sms += "/iniciarhorario: para activar horario configurado en telegram\n";
      sms += "/pararhorario: para desactivar horario configurado en telegram\n";
      sms += "/encender, encender luminaria\n";
      sms += "/apagar, apagar luminaria\n";

      bot.sendMessage(chat_id, sms, "");
    }
```

Figura 3.28 Función para indicar las opciones en *Telegram*

Para los comandos que comienzan con `/proginiciar `, la función extrae la hora de inicio especificada por el usuario. Busca el índice del carácter `: ` para separar la hora y los minutos. Luego, convierte estos valores a enteros y los almacena en las variables `startHour` y `startMinute`. Un mensaje de confirmación con la hora de inicio establecida se envía al usuario.

De manera similar, para los comandos que comienzan con `/progfinal `, la función extrae la hora de fin, convierte los valores a enteros y los almacena en `endHour` y `endMinute`. Se envía un mensaje de confirmación con la hora de fin establecida al usuario. En la *Figura 3.29* se expresa el código correspondiente.

```
// scheduler - setear una hora de inicio
if (text.startsWith("/proginiciar ")) {
  int colonIndex = text.indexOf(':');
  if (colonIndex > 0) {
    startHour = text.substring(13, colonIndex).toInt();
    startMinute = text.substring(colonIndex + 1).toInt();
    bot.sendMessage(chat_id, "Hora de inicio establecida a " + String(startHour) + ":" + String(startMinute), "");
  }
}

// scheduler - setear una hora de fin
if (text.startsWith("/progfinal ")) {
  int colonIndex2 = text.indexOf(':');
  if (colonIndex2 > 0) {
    endHour = text.substring(11, colonIndex2).toInt();
    endMinute = text.substring(colonIndex2 + 1).toInt();
    bot.sendMessage(chat_id, "Hora de fin establecida a " + String(endHour) + ":" + String(endMinute), "");
  }
}
```

Figura 3.29 Función para programar un horario de inicio y final

Para el comando `/iniciarhorario`, la función activa el horario configurado si `activar_calendario` es falso y envía un mensaje de confirmación. Si el horario ya está activo en *Arduino Cloud*, se notifica al usuario. El comando `/pararhorario` desactiva el horario y envía una confirmación al usuario. La *Figura 3.30* expresa a detalle el código correspondiente.

```
// scheduler - iniciar el scheduler
if (text == "/iniciarhorario" && !activar_calendario) {
  scheduleActive = true;
  bot.sendMessage(chat_id, "Horario por telegram activado", "");
} else if (text == "/iniciarhorario" && activar_calendario) {
  bot.sendMessage(chat_id, "El horario de encendido ya se encuentra activo en Arduino Cloud.", "");
}

// scheduler - para el scheduler
if (text == "/pararhorario") {
  scheduleActive = false;
  bot.sendMessage(chat_id, "Horario desactivado", "");
}
```

Figura 3.30 Función para iniciar y parar el horario

Finalmente, los comandos `/encender` y `/apagar` controlan directamente el relé, encendiendo o apagando las luminarias respectivamente mediante `digitalWrite(rele, HIGH) ` y `digitalWrite(rele,LOW) `````LOW) `. Esta función asegura que los comandos de *Telegram* se manejen adecuadamente y se ejecuten las acciones correspondientes en el sistema. La *Figura 3.31* indica el código aplicado.

```
  // encender luminarias
  if (text == "/encender") {
    digitalWrite(rele, HIGH);
  }  // apagar luminarias
  if (text == "/apagar") {
    digitalWrite(rele, LOW);
  }
}
```

Figura 3.31 Función encender y apagar las luminarias por mensaje

La función `handleSchedule()` maneja la lógica de activación y desactivación de un relé en función de un horario preestablecido. La función primero verifica si el horario (`scheduleActive`) está activo. Si lo está, obtiene la hora y los minutos actuales del cliente NTP mediante `timeClient.getHours()' y `timeClient.getMinutes()`, respectivamente.

Luego, la función compara la hora y los minutos actuales con la hora de inicio (`startHour` y `startMinute`). Si coinciden, se establece la variable `cumple_hora` en `true` y `porHorario` en `true`, indicando que se debe cumplir con el horario y se enciende el relé (aunque la línea de código para encender el relé está comentada).

A continuación, la función compara la hora y los minutos actuales con la hora de fin (`endHour` y `endMinute`). Si coinciden o si los minutos actuales son mayores que los minutos de fin, se establece `cumple_hora` en `false`, indicando que se ha alcanzado o superado la hora de fin y que el relé debería apagarse (aunque la línea de código para apagar el relé está comentada, excepto cuando no se detecta movimiento).

Finalmente, si `porHorario` es `true` y `cumple_hora` es `false` y el relé está encendido, se envía un mensaje a través del *bot* de *Telegram* notificando que la luz está encendida fuera del horario permitido y luego se apaga el relé. Se espera 2 segundos (`delay(2000)`) antes de apagar el relé y se establece `porHorario` en `false` para evitar envíos repetidos del mismo mensaje. Esta función garantiza que el relé solo esté encendido dentro del horario permitido y maneja las notificaciones correspondientes cuando se incumple el horario. La *Figura 3.32* expresa a detalle el código para el control del relé por medio de horarios.

```
void handleSchedule() {
  if (scheduleActive) {
    int currentHour = timeClient.getHours();
    int currentMinute = timeClient.getMinutes();

    if (currentHour == startHour && currentMinute == startMinute) {
      //digitalWrite(rele, HIGH);
      cumple_hora = true;
      porHorario = true;
    }

    if (currentHour == endHour && currentMinute >= endMinute) {
      // if (!detecto_movimiento) {
      //   digitalWrite(rele, LOW);
      // }
      cumple_hora = false;
    }
    if (porHorario && !cumple_hora && digitalRead(rele)) {
      bot.sendMessage(chat_id_aux, "Luz encendida fuera de horario, apagando…", "");
      delay(2000);
      digitalWrite(rele, LOW);
      porHorario = false;
    }
  }
}
```

Figura 3.32 Función para encender o apagar el relé según horarios

Desarrollo de la interfaz *web*

La interfaz *web* desarrollada, se ha creado con el objetivo de ofrecer una solución automatizada y controlada para la gestión de luminarias, utilizando tanto sensores de movimiento (*PIR*) como un calendario programado a través de Telegram y *Arduino Cloud*. Cada *widget* en la interfaz tiene una función específica destinada a facilitar la

administración y supervisión de las luminarias, garantizando una operación eficiente y conveniente, como se muestra en la *Figura 3.33.*

El *widget* "Activar luminaria" permite al usuario encender o apagar las luminarias manualmente a través de la interfaz. Este control directo es esencial para situaciones donde se necesita intervención inmediata sin depender de sensores o horarios programados. El estado del relé, representado por el *widget* "Estado relé", muestra visualmente si las luminarias están actualmente encendidas o apagadas, proporcionando una retroalimentación inmediata al usuario sobre el estado del sistema.

El *widget* "Activar calendario" permite al usuario habilitar o deshabilitar el uso de horarios programados para el control de las luminarias. Cuando el calendario está activado, el sistema sigue los horarios establecidos para encender y apagar las luces automáticamente. Esto es particularmente útil para asegurar que las luces se enciendan y apaguen a horas específicas, optimizando el uso de energía y mejorando la seguridad. El *widget* "Movimiento detectado" indica si el sensor *PIR* ha detectado movimiento, lo cual es crucial para la funcionalidad del sistema que apaga las luces cuando no se detecta movimiento, siempre y cuando no esté dentro del horario programado.

El apartado de "Calendario" y "Scheduler" proporciona una visión clara de los horarios establecidos para la operación automática de las luminarias. Estos horarios pueden ser configurados para que las luces se enciendan y apaguen en momentos específicos del día, repitiendo esta acción según la frecuencia definida. Esta característica es fundamental para automatizar el control de iluminación en espacios donde la presencia humana puede ser esporádica, asegurando que las luces solo estén encendidas cuando sean necesarias, lo cual contribuye a la eficiencia energética y a la reducción de costos operativos.

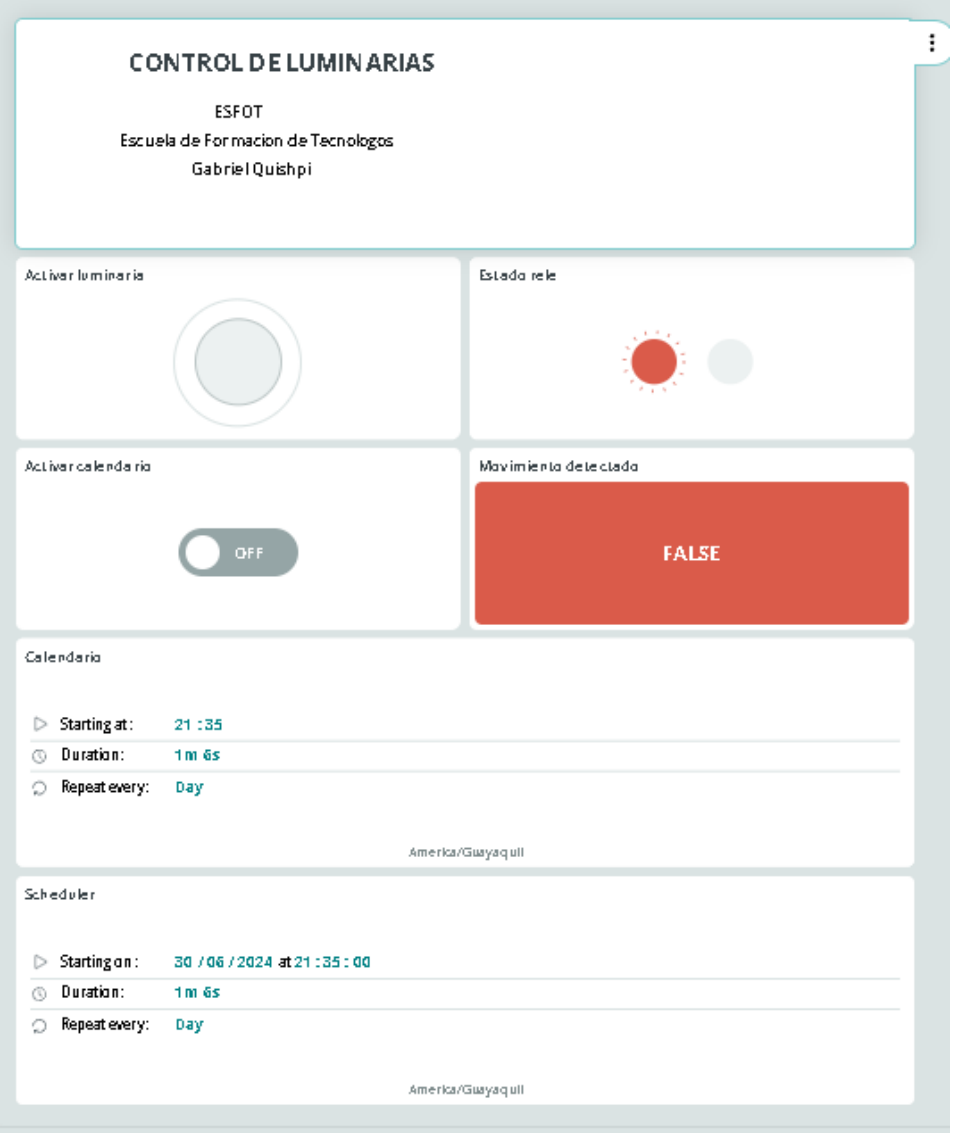

Figura 3.33 Diseño de la interfaz *web*

Desarrollo de la aplicación *Android*

La configuración del diseño de la aplicación para *Android* se realizó utilizando la plataforma *Arduino IoT Cloud* como se muestra en la *Figura 3.34*. La interfaz *web* se sincroniza con la aplicación para *Android*, lo que permite que el diseño de la página *web* se replique en la aplicación móvil. Para utilizar la aplicación, el usuario debe instalar la aplicación *IoT* Remote en su dispositivo *Android* desde Play Store o App Store e ingresar sus credenciales.

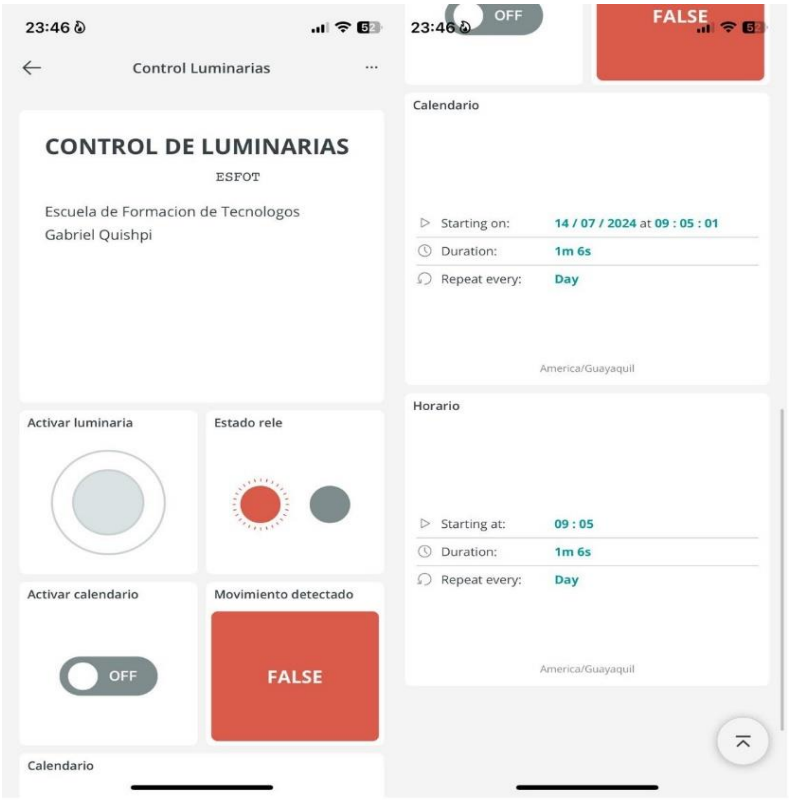

Figura 3.34 Aplicación *Android*

Desarrollo del *bot* de *Telegram*

El desarrollo de un *bot* en *Telegram* se justifica por la necesidad de ofrecer una interfaz accesible y eficiente para el control remoto y la automatización de dispositivos *IoT*. *Telegram*, siendo una plataforma de mensajería ampliamente utilizada y segura, proporciona una manera conveniente para que los usuarios interactúen con sus dispositivos sin necesidad de interfaces adicionales o *hardware* especializado. Al aprovechar la familiaridad de los usuarios con la aplicación de mensajería, se minimiza la curva de aprendizaje y se facilita una adopción más rápida y efectiva del sistema automatizado.

Este *bot* en particular permite a los usuarios controlar luminarias y recibir notificaciones sobre el estado del sistema a través de comandos simples en *Telegram*. Puede encender y apagar luces, configurar horarios de operación y detectar movimiento para activar o desactivar dispositivos automáticamente. La importancia de este *bot* radica en su capacidad para proporcionar control y monitoreo en tiempo real, mejorar la eficiencia energética mediante la automatización basada en horarios y sensores, y ofrecer una capa adicional de seguridad al alertar a los usuarios sobre eventos específicos, como el encendido de luces fuera del horario programado.

El *bot* oficial de *Telegram* para administrar otros *bots*, llamado *BotFather* como se muestra en la *Figura 3.35*, es donde primero se debe construir el *bot*. Se inicia *Telegram*, luego se busca @BotFather. Para iniciar un diálogo con BotFather, se usa el comando /start. A continuación, se crea un nuevo *bot* utilizando el comando /newbot. BotFather pedirá que le dé a su *bot* un nombre y un nombre de usuario especial. Luego se obtendrá un token API, que se necesitará para integrar el *bot* con la aplicación que se cree [30].

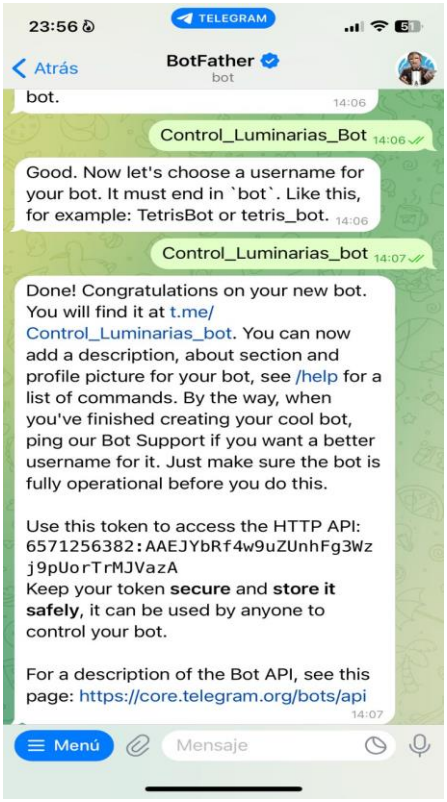

Figura 3.35 Creación del *bot Telegram*

3.4 Implementar el prototipo de control del encendido y apagado de las luminarias

Compilación del código en el SoC

La compilación del código presentado en el *Arduino IoT Cloud* que se presenta en la *Figura 3.36* se realiza a través de un proceso automatizado que facilita la integración y el despliegue de proyectos *IoT*. Primero, es necesario definir las propiedades del dispositivo y las variables en la plataforma *Arduino IoT Cloud*, lo cual se hace en el archivo `thingProperties.h`. Esta configuración permite que el código interactúe con la nube, habilitando el monitoreo y control remoto de los dispositivos conectados. Una vez que las propiedades están configuradas, el código principal, que incluye la lógica para

manejar las conexiones *Wi-Fi*, la comunicación con *Telegram* y el control de los sensores y actuadores, se sube al dispositivo.

Para compilar el código, el usuario debe asegurarse de tener el entorno de desarrollo *Arduino IDE* instalado y configurado correctamente, con las librerías necesarias como `WiFiClientSecure`, `Universal*Telegram*Bot`, `ArduinoJson` y `NTPClient`. Con el dispositivo conectado a la computadora, el usuario selecciona la placa adecuada (por ejemplo, *ESP32*) y el puerto COM correspondiente en el *Arduino IDE*. Luego, al hacer clic en el botón de verificación y posteriormente en el botón de carga, el *Arduino IDE* compila el código y lo sube al dispositivo. Durante este proceso, el IDE traduce el código fuente en binarios que el microcontrolador puede ejecutar, permitiendo así que el dispositivo se conecte a la red, interactúe con el *bot* de *Telegram* y realice las funciones programadas.

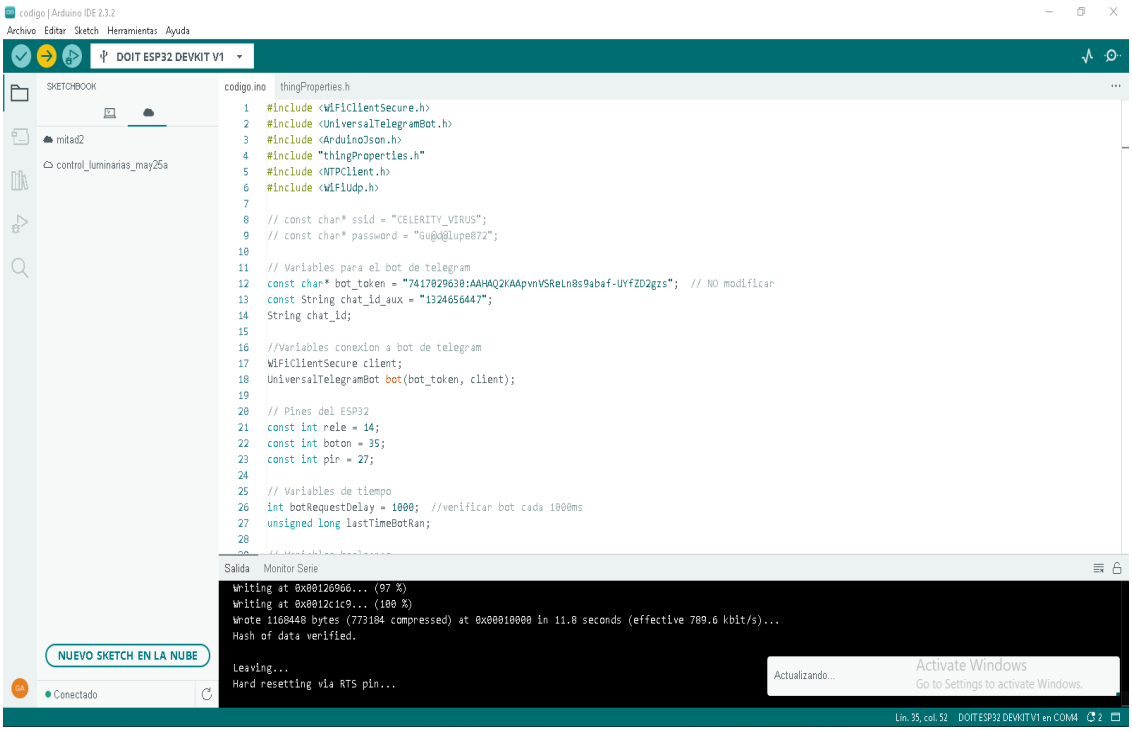

Figura 3.36 Compilación del código en el *ESP32*

Desarrollo del circuito eléctrico

Para poder fusionar el microcontrolador, el sensor *PIR* y el relé, es necesario realizar la creación de un circuito eléctrico que permita integrar cada elemento y que cumpla con su respectiva función. El desarrollo de este circuito eléctrico tuvo como objetivo principal, el buscar la mejor optimización de tal forma que el circuito no ocupe demasiado espacio y que sea totalmente funcional. La *Figura 3.37* presenta el esquema de conexiones

diseñado. En el esquema de conexiones, se utilizaron conectores macho para la respectiva conexión del sensor *PIR* y del relé.

Se implemento conectores macho y hembras en el diseño del circuito eléctrica, para que en algún futuro el reemplazo de los elementos no signifique ningún problema mayor. Para la alimentación del microcontrolador, se mantiene utilizando el conector propio de este.

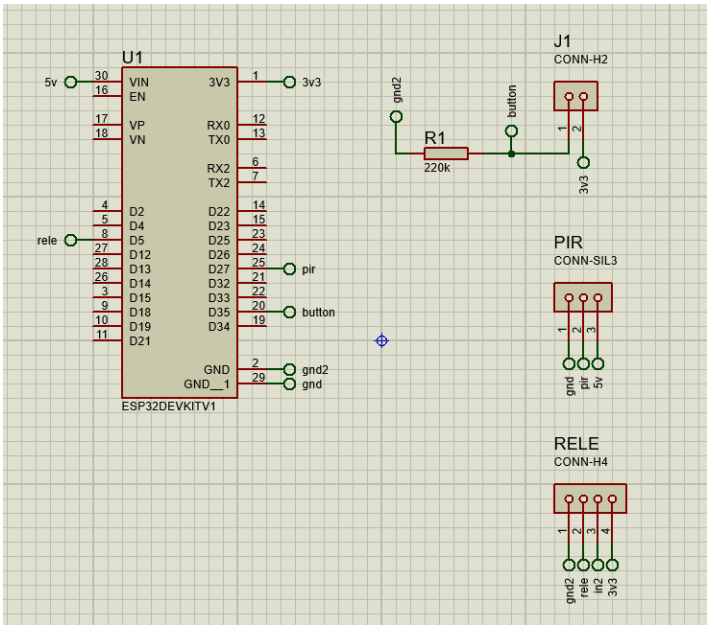

Figura 3.37 Diseño del circuito eléctrico

La asignación de pines del *ESP32* se realizó tal manera que pueda integrarse eficientemente con los pines del relé, el botón y el sensor *PIR*, además de cumplir con los criterios técnicos. Antes de implementar la *PCB*, la planificación permite optimizar el sistema tanto en rendimiento como en funcionalidad, garantizando una integración fluida y exitosa de todos los componentes. La *Figura 3.38* muestra la distribución de la *PCB*.

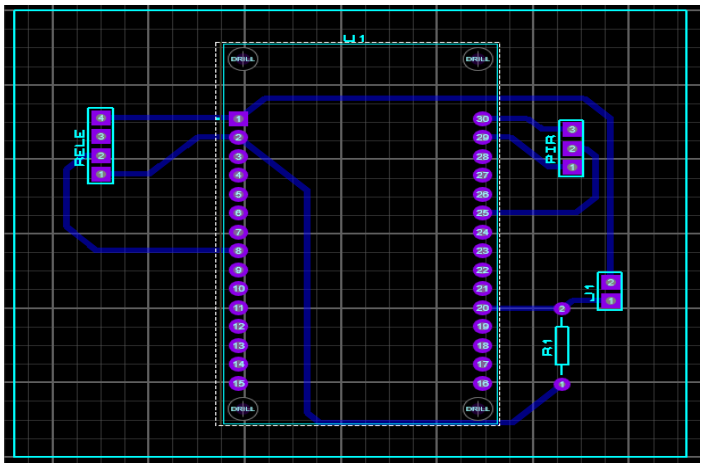

Figura 3.38 Diseño de la *PCB*

Después de realizar el diseño en el *software Proteus*, es necesario continuar con la implementación del diseño en la *PCB*. Para la implementación del diseño se optó por utilizar baquelita fenólica [27], y la técnica del planchado. Debido a su robustez y fuertes características eléctricas, este material se emplea con frecuencia. Por lo general, uno o ambos lados del tablero tienen una capa de cobre que los cubre. Para garantizar una transferencia perfecta del patrón, la superficie de cobre se limpia meticulosamente utilizando un abrasivo fino, como lana de acero, para eliminar cualquier residuo o aceite.

Una técnica común es la transferencia por planchado [28]. Se imprime el diseño del circuito en una hoja especial utilizando una impresora láser. Luego, se coloca la hoja sobre la placa y se aplica calor con una plancha caliente. El calor y la presión transfieren la tinta del tóner a la superficie de cobre de la placa. Una vez enfriada, se retira la hoja y la tinta del tóner queda adherida al cobre, formando una máscara protectora. Las dimensiones de la baquelita se los adquirieron tomando en cuenta los componentes que van a estar conectados en esta.

Después de la transferencia del diseño, cualquier material que no esté cubierto por la máscara protectora se retira de la placa sumergiéndolo en una solución de revelado, a menudo hidróxido de sodio. Una vez producida la placa, se sumerge en una solución ácida, como persulfato de amonio o cloruro férrico. El cobre expuesto es disuelto por este ácido, protegiendo las partes cubiertas por la máscara. Después del grabado, la placa se lava para eliminar cualquier resto de ácido y se utiliza acetona o alcohol para quitar la máscara protectora.

Después de la transferencia del diseño, cualquier material que no esté cubierto por la máscara protectora se retira de la placa sumergiéndolo en una solución de revelado, a menudo hidróxido de sodio. Una vez producida la placa, se sumerge en una solución ácida, como persulfato de amonio o cloruro férrico. El cobre expuesto es disuelto por este ácido, protegiendo las partes cubiertas por la máscara. Después del grabado, la placa se lava para eliminar cualquier resto de ácido y se utiliza acetona o alcohol para quitar la máscara protectora.

Los agujeros para los componentes se perforan una vez que el circuito está grabado en la placa. Usando un taladro preciso, esto se logra yendo a los lugares mencionados en el diseño original. Las piezas electrónicas se perforan, colocan y sueldan en la placa. Para asegurarse de que no haya cortocircuitos y de que todos los componentes estén conectados correctamente, es fundamental realizar una prueba eléctrica y un examen visual [29]. Tras realizar todos pasos para crear la *PCB*, se tiene la *Figura 3.39*, que resulta de este.

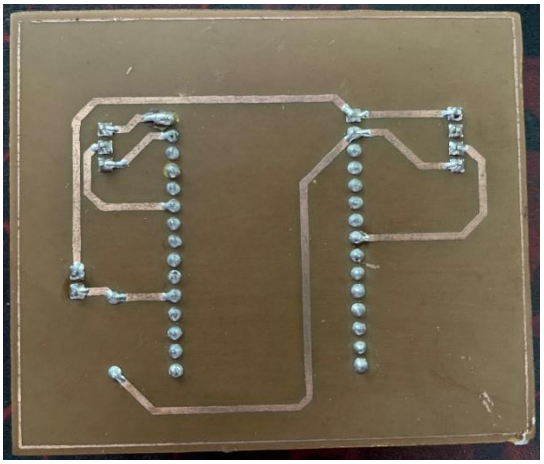

Figura 3.39 *PCB*

Para poder mantener la *PCB* segura de la humedad y otros elementos externos, se aplicó una capa de laca o barniz. Este recubrimiento contribuye a prolongar la vida útil de la *PCB* y a un rendimiento confiable en diversas circunstancias. Aparte, como se muestra en la *Figura 3.40* Fijación de la PCB en la caja de protección

, la *PCB* se coloca en una caja de manera de *MDF* para garantizar aún más su seguridad ante agentes externos.

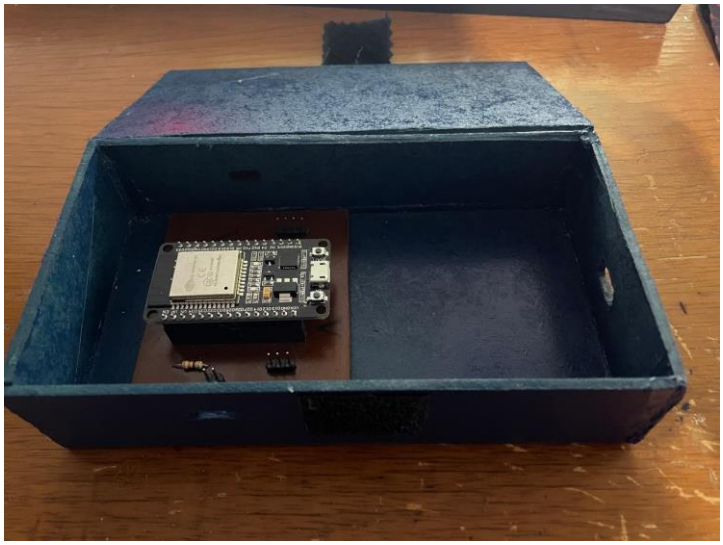

Figura 3.40 Fijación de la *PCB* en la caja de protección

Implementación del sensor *PIR* y el relé en el *PCB*

Para esta parte de implementación, como se visualiza en la *Figura 3.41*, se realizaron las conexiones de la *PCB* con el sensor *PIR* y el relé que son los elementos principales y únicos de este sistema. Para esto, se realizó la conexión mediante los pines que se soldaron a la *PCB* con cables hembra-hembra. Estas conexiones con cables hembra-

hembra se realizaron para poder conectar cables largos que permitan llevar las conexiones a lugares más distantes.

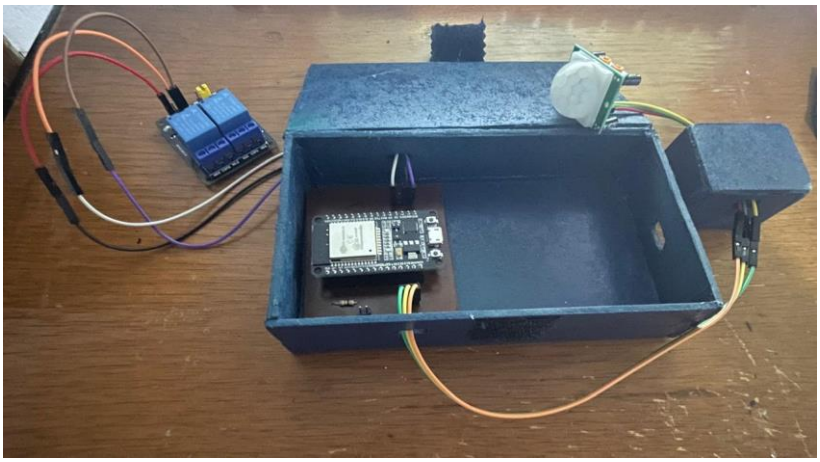

Figura 3.41 Implementación de los elementos a la *PCB*

En este proceso, para poder proteger y limitar el campo de visión del sensor *PIR*, se creó una caja de protección como se muestra en la *Figura 3.42* Implementación del sensor PIR en su caja de protección

. El sensor *PIR* con su caja de protección, será implementada en un lugar estratégico de tal forma que no interrumpa el comportamiento esperado para el sistema.

Figura 3.42 Implementación del sensor *PIR* en su caja de protección

Implementación del prototipo

Para la implementación del prototipo como se observa en la *Figura 3.43*, se utilizó la *PCB* que se creó y se conectaron los elementos principales, que son el relé y el sensor *PIR*. Para llevarlo a un caso más cercano al proceso de instalación, la cual se llevará a cabo en la oficina 8 de la ESFOT, al relé se le conecto una luminaria la cual se conectó al puerto normalmente cerrado y al punto común de las salidas del relé.

La luminaria está conectada de tal forma que el normalmente cerrado permita el encendido y apagado de la luminaria, el puerto común sale hacia el neutro de un enchufe el cual fue conectado a la bombilla de la luminaria para poder conectarlo a un tomacorriente energizado a 110 (v).

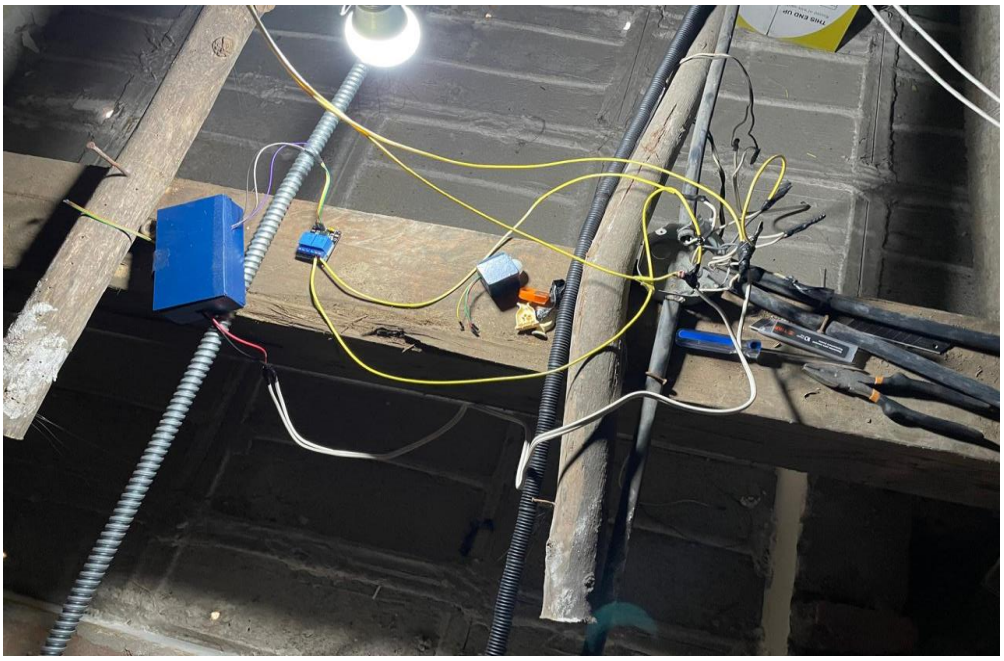

Figura 3.43 Implementación del prototipo

Implementación de la aplicación *Android*

Para poder controlar el sistema mediante la aplicación, es necesario descargar la aplicación *Arduino IoT Cloud* Remote desde la tienda oficial que es la Play Store o la App Store. Es necesario iniciar sesión con las credenciales de la cuenta en donde fue desarrollado el código. Dentro de la aplicación previamente instalada, se procede a buscar la pestaña de los *dashboard*. En esta pestaña de los *dashboard* se encuentra el diseño previamente creado en la plataforma *web*. La *Figura 3.44* muestra la implementación de la aplicación.

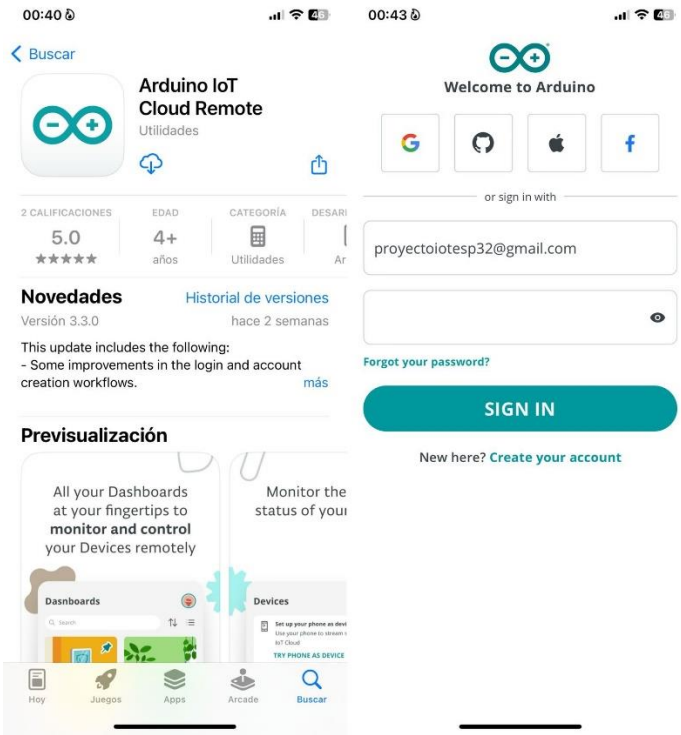

Figura 3.44 Instalación de la aplicación

Implementación de la página *web*

Desde un navegador escogido a preferencia, se procede a buscar desde cualquier motor de búsqueda la página oficial de *Arduino IoT Cloud*. Se inicia sesión con las credenciales de la cuenta en donde se creó el código y se dirige a la pestaña de *dashboard* como se presenta en la *Figura 3.45.* Una vez en la pestaña de los *dashboard*, se procede a seleccionar la que se creó para el proyecto.

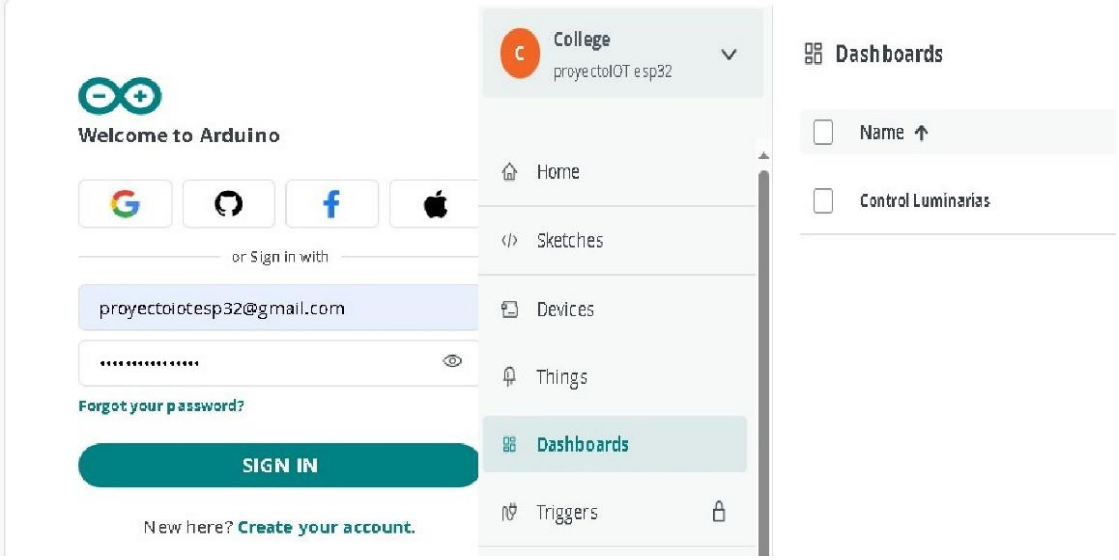

Figura 3.45 Inicio de sesión y selección del *dashboard*

Ya dentro de nuestro *dashboard*, se encuentra el diseño que se creó como se muestra en la *Figura 3.46*, este *dashboard* se puede utilizar desde cualquier lugar de donde se conecte, siempre y cuando se tenga las credenciales de donde se desarrolló el código.

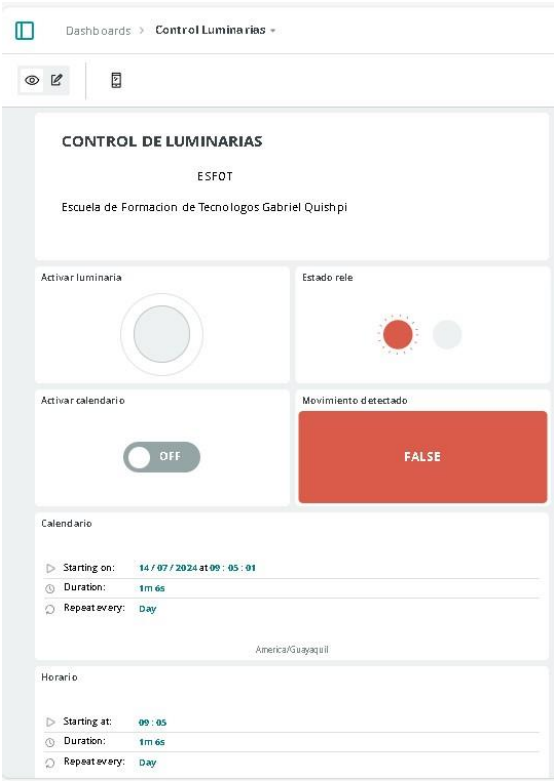

Figura 3.46 Interfaz *web*

3.5 Pruebas de funcionamiento del prototipo

Al concluir la etapa de definir los requisitos técnicos para el control del encendido y apagado de las luminarias, seleccionar el *hardware* y *software* acorde a los requerimientos establecidos, diseñar el prototipo de control del encendido y apagado de las luminarias e implementar el prototipo de control del encendido y apagado de las luminarias, se procedió a completar las pruebas del prototipo en un ambiente que simula el destino final del prototipo. La realización de estas pruebas tiene como objetivo principal, confirmar el cumplimiento correcto de los requisitos establecidos.

Inicio del sistema

Para poder iniciar con el funcionamiento del prototipo, es necesario que el usuario ya cuente instalado la aplicación de *Arduino IoT Cloud* e iniciado sesión, o por su lado que se encuentre logueado en la página de *Arduino IoT Cloud* desde cualquier navegador. Una vez ya iniciado sesión con las credenciales ya sea en la aplicación *Android* o en la interfaz *web*, se procede con la energización del microcontrolador. Una vez ya

energizado el microcontrolador, el *ESP32* se conecta a la red *Wi-Fi*, esto mediante las líneas de código que se configuraron previamente. Cuando microcontrolador ya se encuentra conectado a la red *Wi-Fi*, se puede visualizar que el sistema está en línea, ya sea en la página o mediante el monitor serie en el *Arduino IDE*. Tras seguir estos pasos, el sistema puede utilizarse.

Al momento de la iniciación del sistema, se puede observar en indicador visual cual es el estado del relé, el ultimo estado del sensor *PIR*, el switch del control del programador de los horarios y el botón de encendido y apagado. La *Figura 3.47* muestra este último estado de configuración en la que se quedó el sistema.

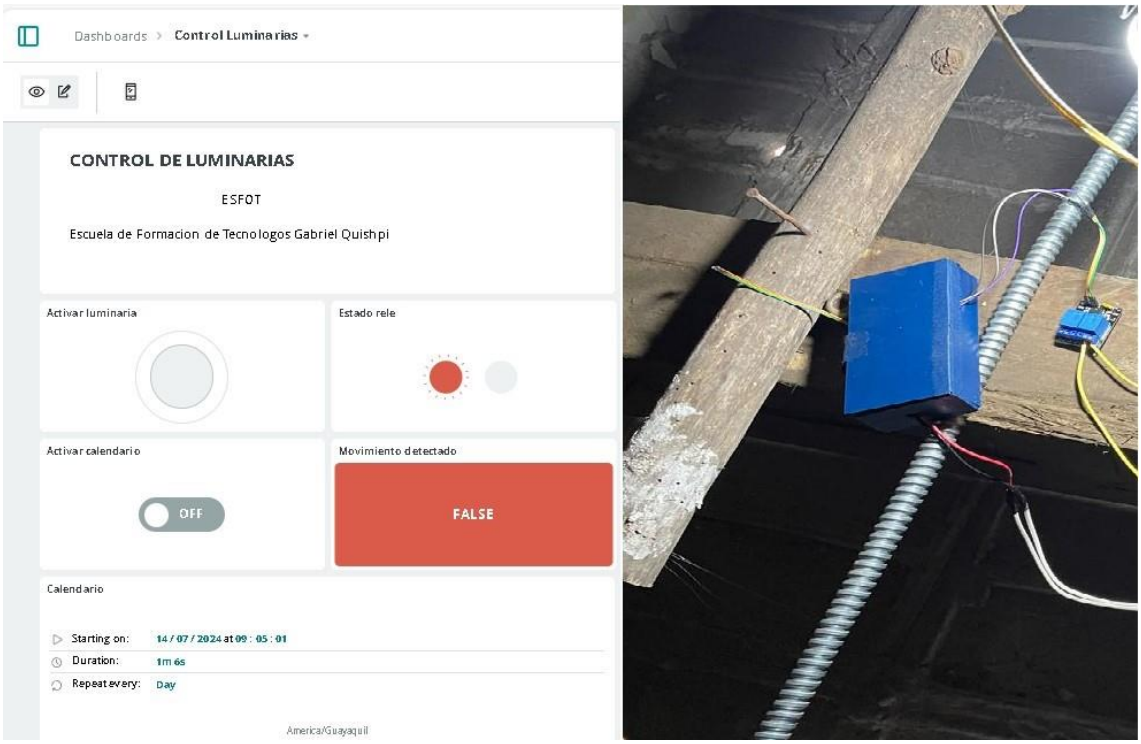

Figura 3.47 Inicio del sistema

Prueba de control manual de las luminarias mediante la aplicación *Android*

Para poder realizar las pruebas del control de las luminarias de forma manual, primero se ubica el botón, al iniciar el sistema este *widget* siempre va a iniciar en apagado. Este botón permite controlar el relé, que es el encargado de encender o apagar las luminarias. Para poder encender o apagar las luminarias, es necesario presionar el botón. Como se muestra en la *Figura 3.48,* para poder identificar el funcionamiento correcto de este, se tiene un *widget* con dos luminarias que indican en qué estado se encuentra el relé tras la pulsación.

Figura 3.48 Indicador visual del relé

Al presionar el *widget* del botón por un instante como se muestra en la *Figura 3.49*, se enciende el relé que en consecuencia activa nuestras luminarias, tras la pulsación de encendido de las luminarias, se procede a enviar un mensaje por medio del *bot* de *Telegram* el cual indica el mensaje “Luminarias encendidas” el cual es un indicativo del estado del sistema, este indicador cuenta con el mismo sentido del indicador visual.

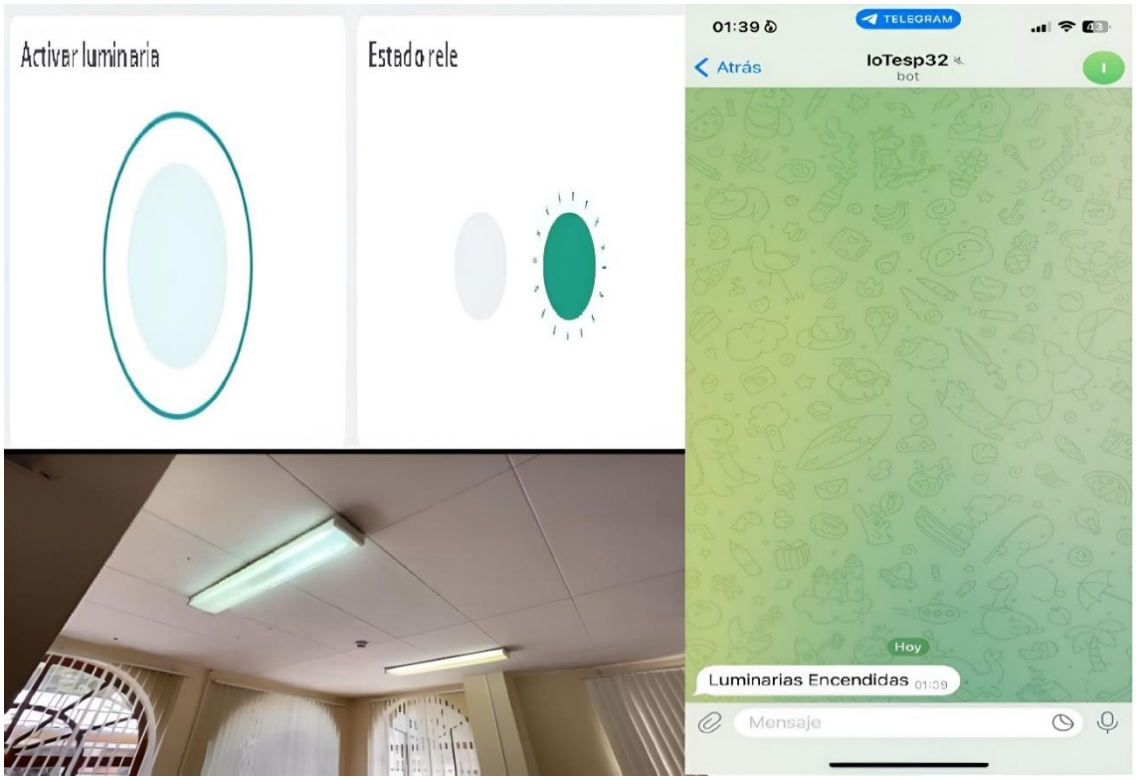

Figura 3.49 Encendido de las luminarias y visualización del indicador

Cuando se presiona por segunda vez el *widget* del botón, se procede con la desactivación del relé y por lo tanto las luminarias se desactiva, al momento del apagado del sistema, se envía de igual forma un mensaje que indica “Luminarias apagadas”, como se muestra en la *Figura 3.50* , permitiendo identificar el estado del sistema.

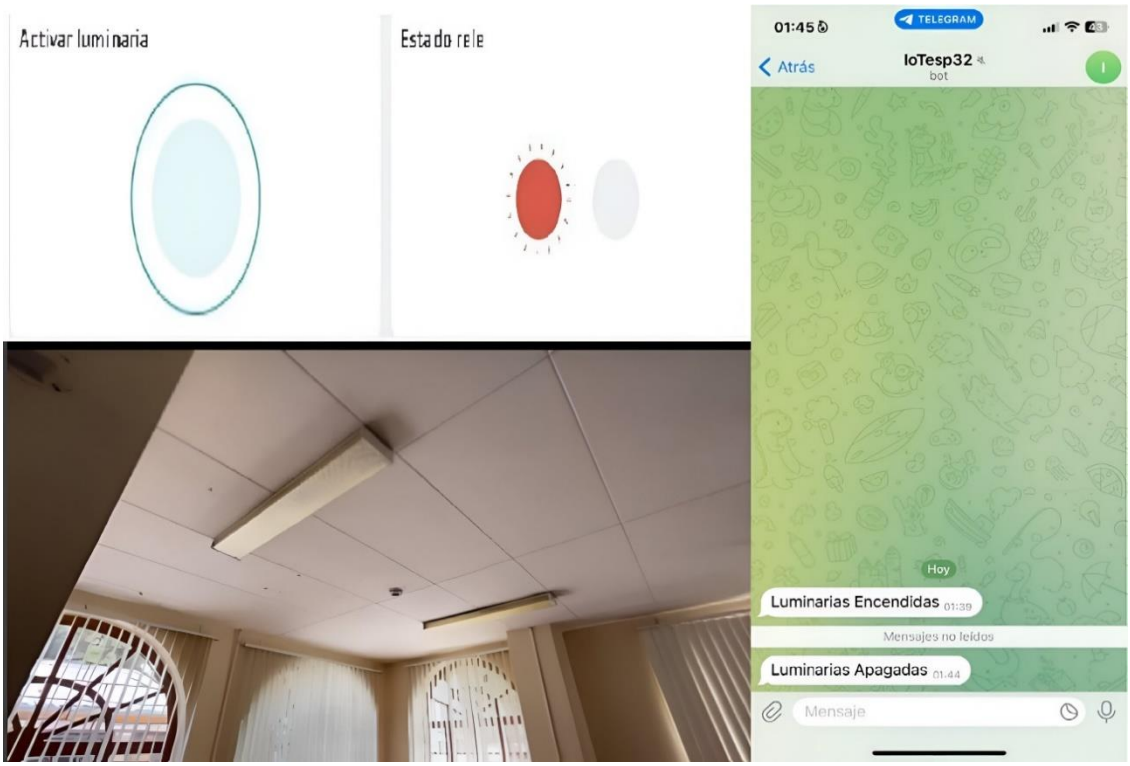

Figura 3.50 Apagado de las luminarias y visualización del indicador

Es importante mencionar que el sensor *PIR* si detecta señales, pero por la lógica de programación, estas detecciones no influyen en nada.

Prueba de control manual de las luminarias mediante la interfaz *web*

En la interfaz *web*, se espera que se cumpla las condiciones tal cual como se cumplen en la aplicación *Android*. El estado del relé siempre iniciara apagado, de tal forma que, a la primera pulsación del *widget* del botón, se encienda el relé y este encienda la luminaria, tal como se muestra en la *Figura 3.51,* de igual forma como en la aplicación *Android*, tras encender las luminarias este envía un mensaje por el *bot* de *Telegram* indicando el estado del sistema.

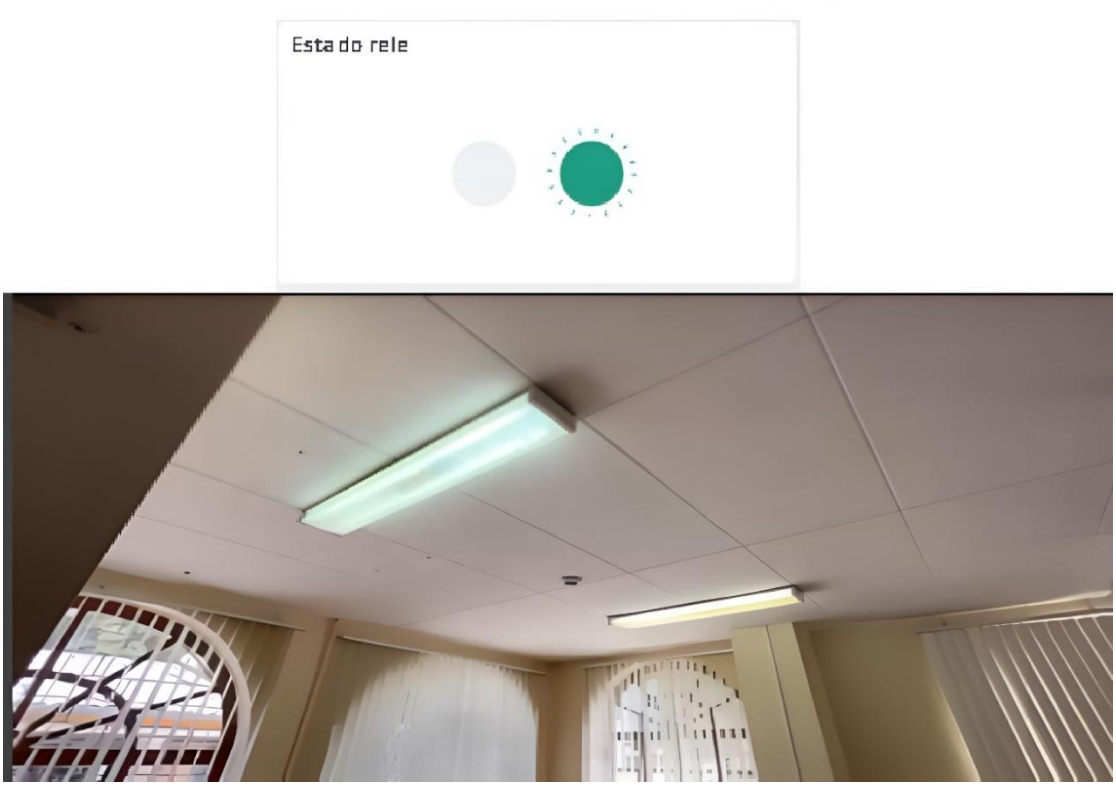

Figura 3.51 Encendido de la luminaria por medio de la interfaz *web*

Al realizar la segunda pulsación, el estado del relé cambia a apagado enviando un mensaje por medio de *Telegram* indicando que las luminarias están apagadas, como se muestra en la *Figura 3.52,* de tal forma que la luminaria se desactiva y el visualizador del estado del relé se actualiza.

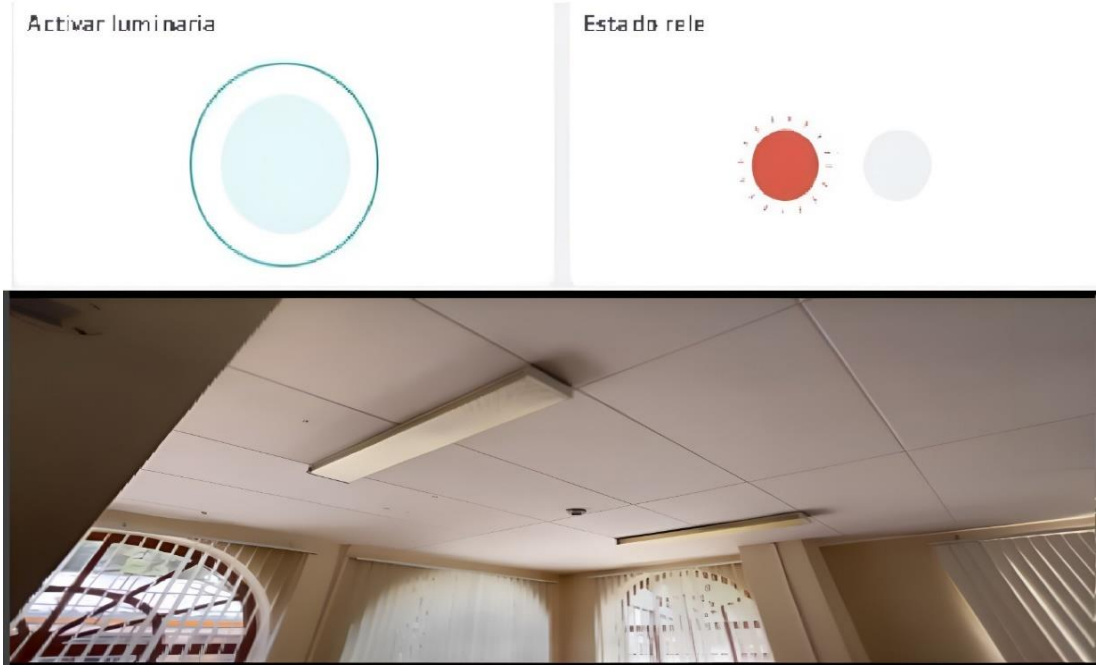

Figura 3.52 Apagado de la luminaria por medio de la interfaz *web*

Prueba de control manual de las luminarias mediante el *bot* de *Telegram*

Para poder realizar el control manual mediante el *bot* de *Telegram*, es necesario tener previamente descargada la aplicación e iniciar la sesión en esta. La aplicación de *Telegram* puede ser descargada desde la play store o la app store. Una vez ya instalada y luego de iniciar sesión en la aplicación, se procede a buscar en *bot* en la aplicación, la cual cuenta con el nombre de *IoTESP32*. Tal como se programó en el código, para poder iniciar el *bot* se manda un texto con el mensaje /iniciar. Al escribir el comando /iniciar, el *bot* despliega un menú con todas las opciones con las que cuenta este *bot*, tal como se muestra en la *Figura 3.53*.

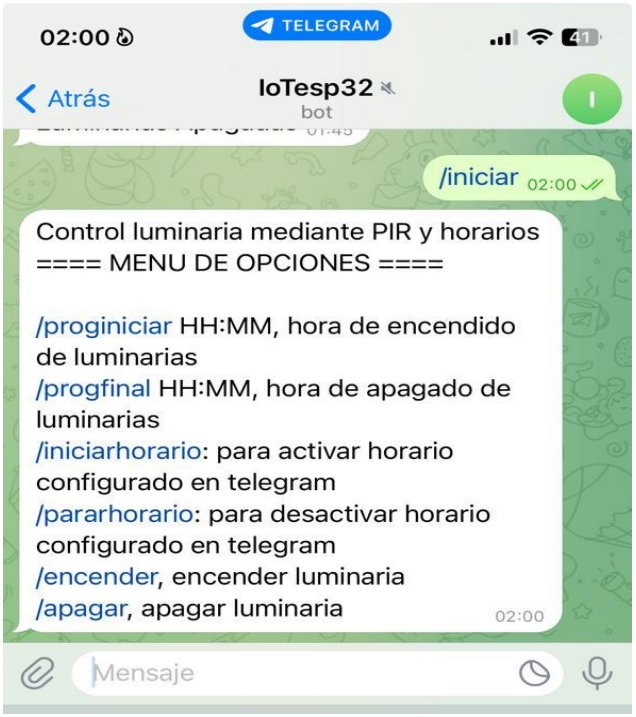

Figura 3.53 Menú *Telegram*

Para poder controlar de manera manual el sistema de luminarias el menú de *Telegram* presenta dos opciones como se muestra en la *Figura 3.54.*

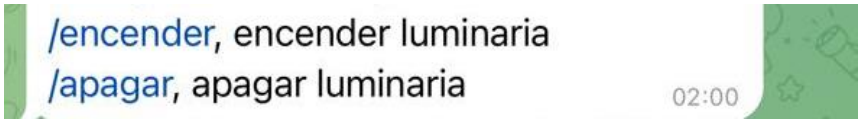

Figura 3.54 Opciones control manual por medio del *bot* de *Telegram*

La opción que permite encender las luminarias es la de /encender, al momento de enviar este mensaje por el *bot* de *Telegram*, el sistema enciende las luminarias y se muestre un mensaje en el *bot* el cual indica “Luminarias encendidas”, tal como se muestra en la

Figura 3.55, de igual forma en la aplicación del *Arduino IoT Cloud* , se actualiza en indicador visual del relé.

Figura 3.55 Encendido del sistema por medio del *bot* de *Telegram*

Para apagar el sistema de luminarias, se utiliza el mensaje /apagar, tras mandar este mensaje por medio del *bot*, las luminarias se desactivan y se recibe el mensaje “Luminarias apagadas” en el *bot*, el estado del relé se procede a actualizar en la aplicación de Arduino *IoT* tal como se muestra en la

*Figura **3.56**.*

Figura 3.56 Apagado del sistema por medio del *bot* de *Telegram*

Prueba de programación de horarios por medio de la aplicación *Android*

Para la programación de horarios mediante la aplicación de *Android*, se tiene dos *widget*s como se muestra en la *Figura 3.57.* El primer *widget* llamado “Calendario” permite programar horarios, con este *widget* se puede programar la fecha en la que se prevé inicie, el tiempo el cual va a estar encendido el calendario y la frecuencia con la que se va a repetir esta programación. Al programar el horario por medio de este calendario, se puede establecer el día, mes y el año en la que va a empezar, acompañado de su hora, minuto y segundo para empezar. Si se programa por medio del segundo *widget* llamado “Horario”, este solo permite programar la hora con minutos y segundos de cuando empezara el horario, así mismo permitiendo programar su frecuencia de repetición y el tiempo.

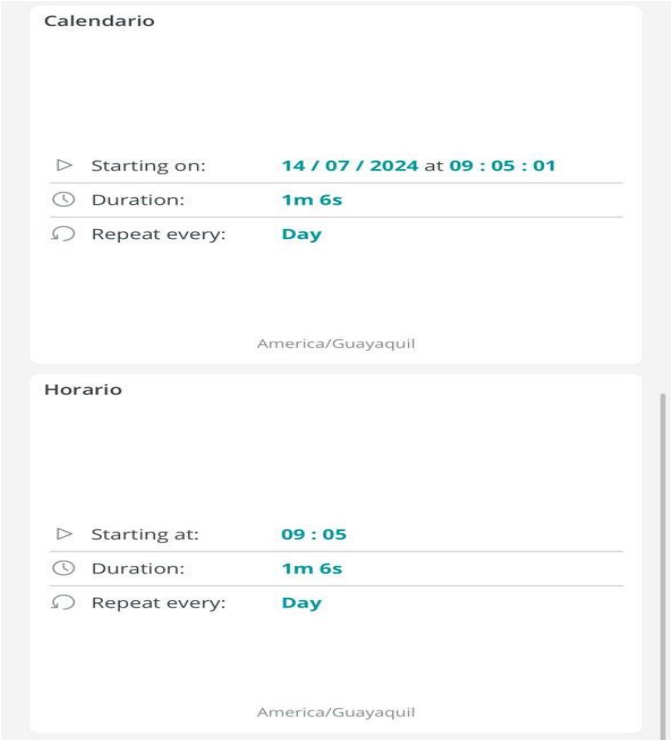

Figura 3.57 *Widgets* de programación de horarios

Para poder utilizar estos *widgets* de programación de horarios, es necesario tener desactivado la programación de horarios por medio del *bot* de *Telegram*. Para asegurarnos de que la programación de horarios este desactivada, se envía un mensaje por *Telegram* con el mensaje /iniciar para poder encontrar la opción que se necesite. La opción en este caso es la opción de “/pararhorario”, al enviar este mensaje por medio del *bot* de *Telegram*, el *bot* entregara un mensaje diciendo “Horario desactivado” como se muestra en la *Figura 3.58,* el cual asegura que no hay un horario establecido y se

pueda continuar con nuestra programación mediante los *widgets* de la aplicación de *Arduino IoT Cloud* .

Figura 3.58 Desactivación del horario de *Telegram*

Para la programación de horarios por medios de los *widgets*, el primer paso que se debe cumplir es cambiar el estado del switch llamado “Activar calendario”, de apagado a encendido como se muestra en la *Figura 3.59.*

Figura 3.59 Activación de la programación de *widgets*

Una vez activado la programación de horarios, se puede seleccionar el *widget* por el cual se desea empezar la programación. La configuración de los horarios se los establece de una manera muy sencilla, independientemente del *widget* seleccionado, tal como se muestra en la *Figura 3.60.*

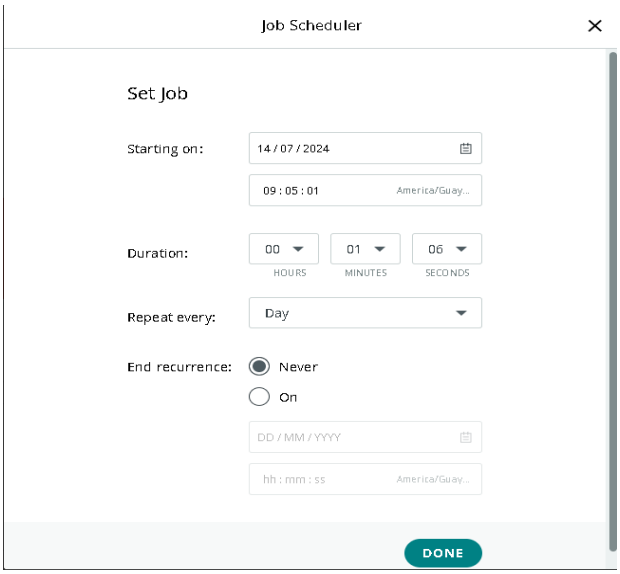

Figura 3.60 Programación de horarios en el *widget*

La lógica del código programado establece que cuando se cumpla la hora de inicio, el sensor *PIR* entrara a funcionar, permitiendo la activación o desactivación del relé. El sensor *PIR* a estar dentro del horario, funciona como un interruptor. El código establece que, una vez pasado este horario, el sensor *PIR* dejara de manejar al relé, permitiendo así que cuando no haya un horario establecido, el sensor no realice ninguna acción y permita al usuario activar las luminarias en cualquier momento mediante el botón del control manual. Para estas pruebas de funcionamiento, se configuro un horario que establece la hora de encendido como se muestra en la *Figura 3.61,* a las 2:35 am, con una duración de encendido de un minuto y una frecuencia de repetición de todos los días.

En este tiempo, en sensor actuara en conjunto con el relé, permitiendo la activación o desactivación de las luminarias. Esto viene acompañado del envió del mensaje por medio del *bot* de *Telegram*, indicando que las luminarias están encendidas o apagadas, a su vez también se actualiza el indicador del relé y el indicador del sensor *PIR* indicando si detecto movimiento.

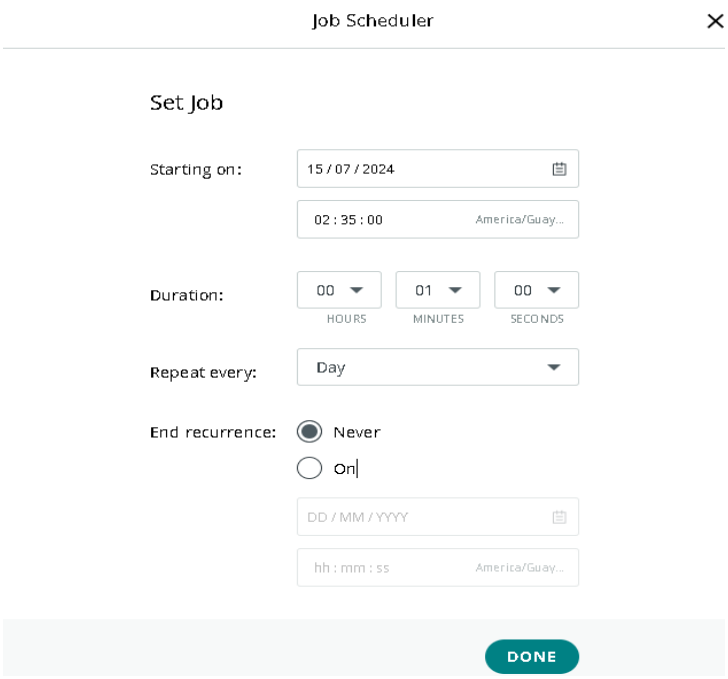

Figura 3.61 Configuración del horario

Es importante recalcar que también se puede configurar la fecha y la hora en la que se desea que para de repetirse esta secuencia de horarios.

Prueba de programación de horarios por medio de la aplicación *web*

La programación de horarios por medio de la aplicación *web*, se realiza de la misma manera por la cual se realiza la programación por medio de la interfaz *Android*. La

programación de horarios por medio de la aplicación *web* se presenta en la *Figura 3.62.* Para este caso, se realizó la programación estableciendo un horario de finalización, para poder tener una prueba de funcionamiento de esta funcionalidad que se puede configurar en los *widget*s seleccionados.

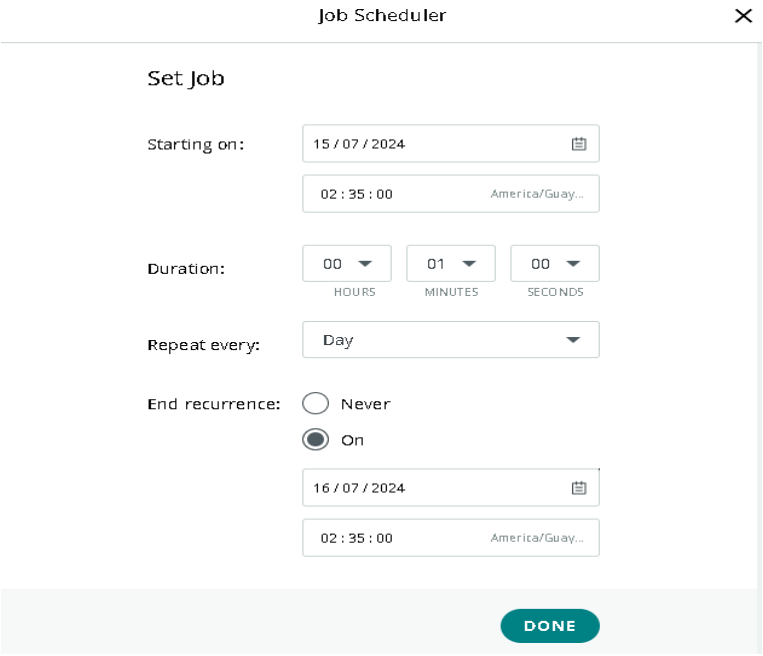

Figura 3.62 Configuración del horario por la aplicación *web*

Prueba de programación de horarios por medio del *bot* de *Telegram*

Para la programación de horarios por medio del *bot* de *Telegram*, primero es necesario desactivar el *widget* del switch llamado “Activar calendario”, para poder trabajar de forma correcta con la programación de horarios por medio del *bot* de *Telegram*, como se presenta en la *Figura 3.63*.

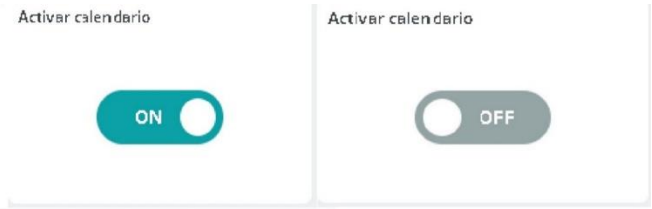

Figura 3.63 Desactivación de la programación de horarios por medio del *widget* en *Arduino IoT Cloud*

Primero, se despliega el menú del *bot* de *Telegram* mediante el envío del mensaje /iniciar. En este menú, para programar los horarios se cuenta con cuatro opciones como se muestra en la *Figura 3.64.*

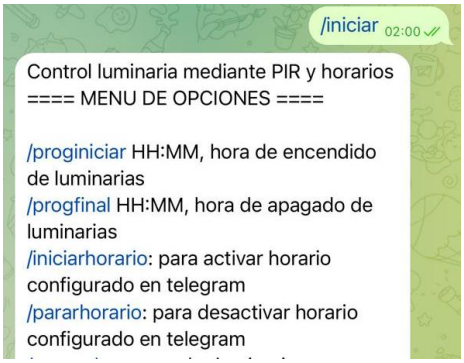

Figura 3.64 Menú para la programación de horarios por medio del *bot* de *Telegram*

La programación del horario de inicio se programa mediante el comando /proginiciar HH:MM, en la cual es necesario programar las horas y los minutos con el siguiente formato HH:MM, al mandar este mensaje por medio del *bot*, este devuelve un mensaje que dice “Hora de inicio establecida a HH:MM”. Para establecer el horario del final, se envía el mensaje “/progfinal HH:MM”, este mensaje va acompañado de la hora y los minutos en la cual el programa dejara de funcionar, al recibir esta instrucción el *bot* entrega un mensaje indicando “Hora de fin establecida a HH:MM”.Una vez establecido la hora y minutos del inicio y del final, para activar el horario se continua con él envió del mensaje “/iniciar horario” el cual para confirmar nuestra programación entrega el mensaje “Horario por telegram activado”. La configuración del horario por medio del *bot* de telegram se presenta en la *Figura 3.65.*

Figura 3.65 Programación de horario por medio de *Telegram*

Al igual que en la programación por medio de los *widget*s de *Arduino IoT Cloud* y el encendido y apagado por medio del control manual, cada vez que se encienda o apague el sistema de luminarias, se recibirán mensajes indicando el estado de estas con los mensajes “Luminarias encendidas” o “Luminarias apagadas” respectivamente y acompañados de la actualización del estado del relé. Este horario programado se repetirá todos los días a la hora programada, hasta que se envié el mensaje

“/pararhorario” en cual enviará un mensaje con el texto “Horario desactivado” indicando que el horario ya no se repetirá.

Prueba de utilización del botón manual y la programación de horarios

La lógica de la programación que se creó para este sistema permite que el botón del control manual funcione de forma independiente. De tal manera que, si está programado un horario ya sea por medio de los *widget*s de la página del *Arduino Cloud* o por medio del *bot* de *Telegram*, este *widget* del botón pueda trabajar. Esto quiere decir que si dentro del horario programado, el sensor *PIR* activo el relé, y que el usuario no activo en sensor nuevamente para apagarlo, el *widget* del botón pueda apagarlo o en su caso contrario, si el sensor *PIR* no ha activado el relé, este pueda ser activado por medio del *widget* del botón como se muestra en la *Figura 3.66*.

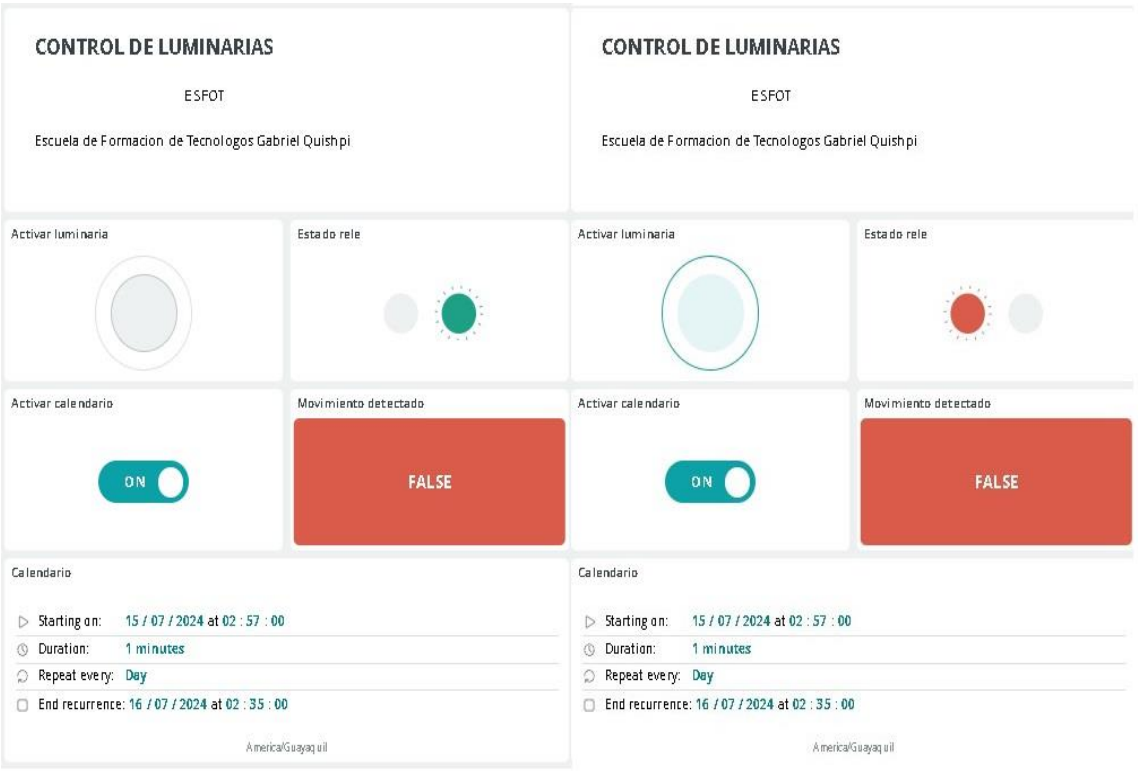

Figura 3.66 Integración del botón manual con la programación de horarios

Prueba de utilización de la programación de horarios por medio de *Telegram* y por medio de los *widgets* de *Arduino IoT Cloud*

En este caso, la lógica de la programación que se creó para este sistema no permite que actúen de forma simultánea ya que, el código intenta implementar dos sistemas de programación de horarios diferentes. Uno es controlado por comandos de *Telegram* y el otro por el *Arduino IoT Cloud*. Si ambos sistemas están activos simultáneamente, pueden surgir conflictos sobre quién tiene el control sobre el relé. Por ejemplo, si el

horario en *Telegram* activa la luz, pero el horario en *Arduino IoT Cloud* la desactiva, el relé recibirá comandos contradictorios, lo que puede llevar a comportamientos inesperados.

Ambos sistemas dependen de la sincronización del tiempo a través de NTP. Cualquier discrepancia en la sincronización del tiempo entre los dos sistemas puede llevar a fallos en la programación de los horarios. Si el tiempo no se actualiza correctamente, las acciones programadas pueden no ejecutarse en el momento esperado.

Cuando se recibe el comando "/iniciarhorario" a través del *bot* de*Telegram* y el control de horario de Arduino *IoT* Cloud "Activar calendario" está activo, el código envía un mensaje de respuesta a *Telegram* indicando que el "horario de encendido ya está activo en *Arduino Cloud*", tal como se muestra en la *Figura 3.67*.

Figura 3.67 Incompatibilidad del uso de la programación de horarios por medio del *widget* de *Arduino IoT Cloud* y por medio del *bot Telegram*

Resumen de las pruebas realizadas

Después de realizar las distintas pruebas, se presenta la *Tabla 3.5* en la cual se indica su cumplimiento o incumplimiento.

Tabla 3.5 Resumen de pruebas realizadas

Prueba de ejecución	Prueba pasada	Prueba no pasada
Inicio del sistema	*X*	

Prueba de ejecución	Prueba pasada	Prueba no pasada
Prueba de control manual de las luminarias mediante la aplicación *Android*	X	
Prueba de control manual de las luminarias mediante la interfaz *web*	X	
Prueba de control manual de las luminarias mediante el *bot* de *Telegram*	X	
Prueba de programación de horarios por medio de la aplicación *Android*	X	
Prueba de programación de horarios por medio de la aplicación *web*	X	
Prueba de programación de horarios por medio del *bot* de *Telegram*	X	
Prueba de utilización del botón manual y la programación de horarios	X	
Prueba de utilización de la programación de horarios por medio de *Telegram* y por medio de los *widgets* de Arduino *IoT* Cloud	X	

El prototipo demostró ser efectivo para el control automático de las luminarias, logrando optimizar el uso de energía eléctrica en la oficina 8 de la ESFOT. Las pruebas confirmaron que el sistema funciona correctamente tanto en modo automático como en manual, ofreciendo una solución robusta y flexible para el control de iluminación. La implementación exitosa de este sistema puede contribuir significativamente al ahorro de energía, reduciendo el consumo de electricidad y los costos asociados.

Implementación del prototipo en la oficina 8 de la ESFOT

Para verificar que el sistema pueda ser capaz de ser implementado en un ambiente real, se montó el sistema la oficina 8 de la ESFOT. Para la implementación del sistema dentro

de la oficina, no fue necesario realizar cambios ya sean en la programación o en los componentes. El funcionamiento del proyecto se puede apreciar con mayor detalle en el video proporcionado mediante el escaneo del código QR de la *Figura 3.68* o por medio del siguiente link:

https://epnecuador.sharepoint.com/:v:/s/TIC2024A_LPazmino/EZsJM6wmMy5DtGM3Z7hYRhABZgrkUhswdCimcbhr2z9VkQ?e=ZRtRGu

Figura 3.68 Código QR de la implementación y pruebas de funcionamiento del prototipo de encendido y apagado de luminarias.

Costo del prototipo

En la *Tabla 3.6* se puede encontrar el precio de los elementos utilizados. El precio de los elementos fueron tomados de la página de Marcado Libre.

Tabla 3.6 Costo de elementos [31] [32] [33] [34] [35] [36] [37]

Elemento	Cantidad	Precio unitario	Precio total
ESP32	1 (u)	$ 11.73	$ 11.73
Sensor *PIR*	1 (u)	$ 3.35	$ 3.35
Modulo Relé	1 (u)	$ 2.35	$ 2.35
Placa *PCB*	1 (u)	$ 12.50	$ 12.50
Caja de protección	2 (u)	$ 4	$ 8.00
Cables H - H	6 (u)	$ 0.20	$ 1.20
Cables M - H	6 (u)	$ 0.20	$ 1.20

Mano de obra	24 (h)	$ 10	$ 240
Cable UTP (m)	5 (m)	$ 0.80	$ 4.00
Cable flexible (m)	3 (m)	$ 0.45	$ 1.35
Fuente transformador/cargador	1 (u)	$ 5	$ 5.00
		Total	$ 290.68

4 CONCLUSIONES

- Se establecieron de manera efectiva los requisitos técnicos necesarios para el control del encendido y apagado de las luminarias, garantizando la correcta selección de hardware y software acorde a las necesidades del proyecto. Esta etapa fue fundamental para asegurar que todos los componentes y funcionalidades del sistema estuvieran alineados con las necesidades del proyecto.
- La elección del microcontrolador ESP32, junto con los sensores de movimiento y actuadores, demostró ser idónea para cumplir con los requisitos del sistema. Esta selección permitió una integración de todos los componentes involucrados. La evaluación cuidadosa de las especificaciones técnicas del hardware y software fue fundamental para satisfacer las necesidades específicas del proyecto. La capacidad del ESP32 para manejar múltiples tareas y su compatibilidad con diversos sensores y actuadores aseguraron que el prototipo final funcionara según los requerimientos establecidos. Además, la implementación de estos componentes facilitó futuras expansiones y mejoras del sistema.
- El diseño y montaje del prototipo en una *PCB* se llevaron a cabo de manera precisa, asegurando conexiones fiables y duraderas. El diseño esquemático del sistema, incluyendo el desarrollo de circuitos y algoritmos, permitió una integración eficiente de todos los componentes, lo que resultó en un sistema funcional y efectivo para el control de las luminarias.
- La integración de tecnologías IoT en el prototipo demostró ser exitosa, permitiendo la comunicación y el control remoto de las luminarias mediante una plataforma web, una aplicación Android y un bot de Telegram. Esta integración facilita la gestión y monitoreo del sistema desde múltiples dispositivos y ubicaciones, aumentando la versatilidad y accesibilidad del sistema.

- El sistema de automatización implementado permitió el encendido y apagado de las luminarias basado en la detección de movimiento, optimizando el uso de la energía y reduciendo el desperdicio. La automatización asegura que las luminarias solo se enciendan cuando sea necesario, contribuyendo a la eficiencia energética y a la sostenibilidad.
- La plataforma web y la aplicación Android desarrolladas proporcionaron una interfaz de usuario intuitiva y fácil de usar, permitiendo a los usuarios interactuar con el sistema de manera eficiente. La simplicidad y funcionalidad de la interfaz aseguraron una experiencia de usuario positiva, facilitando la adopción y el uso del sistema.
- Las pruebas realizadas demostraron la efectividad del sistema en condiciones reales, confirmando que el prototipo cumple con los requisitos establecidos. Las pruebas de control manual y automático, a través de la aplicación Android, la interfaz web y el bot de Telegram, validaron la funcionalidad del sistema, asegurando su capacidad para gestionar la iluminación de manera eficiente.
- La implementación del prototipo ha demostrado un impacto significativo en la eficiencia energética y la gestión de la iluminación en la oficina 8 de la ESFOT. Este sistema no solo mejora la eficiencia energética, sino que también ofrece flexibilidad y accesibilidad en el control de la iluminación, lo que lo hace aplicable a entornos similares que buscan optimizar el uso de energía y mejorar la gestión de recursos.
- El diseño del prototipo y la arquitectura del sistema permiten su escalabilidad y replicabilidad en otras áreas de la ESFOT o en diferentes entornos que requieran una solución similar. La flexibilidad del sistema facilita su adaptación a nuevas necesidades y la expansión del proyecto a otros contextos, demostrando su potencial para aplicaciones más amplias.
- Durante el desarrollo e implementación del prototipo, se identificaron y superaron varios desafíos técnicos y operativos. Las lecciones aprendidas en el proceso proporcionan una base sólida para futuros proyectos, mejorando la capacidad para enfrentar y resolver problemas similares en el desarrollo de sistemas IoT.

5 RECOMENDACIONES

- Es esencial establecer un plan de mantenimiento y actualización regular del sistema, incluyendo *hardware* y *software*. La incorporación de nuevas

tecnologías y la corrección de posibles fallos garantizarán el buen funcionamiento y la longevidad del sistema.

- Implementar un sistema de monitoreo y análisis de datos para evaluar el desempeño del prototipo y su impacto en el consumo de energía. Esto permitirá identificar patrones de uso, áreas de mejora y realizar ajustes necesarios para optimizar aún más el sistema.
- Evaluar las condiciones ambientales de la ubicación, para garantizar que el sensor movimiento funcione correctamente. Asegurarse de que no haya interferencias que puedan afectar el desempeño de los sensores.
- Confirmar la disponibilidad de una conexión estable a Internet, necesaria para el funcionamiento de la plataforma web, la aplicación *Android* y el *bot* de *Telegram*. Evaluar la calidad de la señal *Wi-Fi* o la posibilidad de instalar puntos de acceso adicionales si es necesario.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] La casa domótica: consumo eficiente de energía en el hogar. (2023, June 8). Retrieved May 13, 2024, from Telefónica website: https://www.telefonica.com/es/sala-comunicacion/blog/la-casa-domotica-consumo-eficiente-de-energia-en-el-hogar/

[2] Aplicación e importancia de la IoT en la actualidad. (n.d.). Retrieved May 13, 2024, from Usal.es website: https://bisite.usal.es/es/blog/formacion/22/08/17/aplicaci-n-e-importancia-de-la-iot-en-la-actualidad-bisite

[3] Carmenate, J. G. (2021, February 16). *ESP32* Wifi + Bluetooth en un solo lugar. Retrieved May 13, 2024, from Programarfacil Arduino y Home Assistant website: https://programarfacil.com/esp8266/*ESP32*/

[4] "¿Qué es una interfaz gráfica de usuario (GUI)?," IONOS Digital Guide, 14-Sep-2020. [Online]. Available: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/que-es-una-gui/. [Accessed: 10-Jun-2024].

[5] Yañez, C. (2020, November 3). Qué es Arduino IoT Cloud. Retrieved May 13, 2024, from CEAC website: https://www.ceac.es/blog/que-es-arduino-iot-cloud

[6] 330ohms, P. (2020, December 9). ¿Qué es un sensor *PIR*? Retrieved May 13, 2024, from 330ohms website: https://blog.330ohms.com/2020/12/09/que-es-un-sensor-*PIR*/

[7] Porto, J. P., & Merino, M. (2014, March 6). Relay. Retrieved May 13, 2024, from Definición.de website: https://definicion.de/relay/

[8] "1. ¿Qué es un microcontrolador?," Xbot.es. [Online]. Available: https://sherlin.xbot.es/microcontroladores/introduccion-a-los-microcontroladores/que-es-un-microcontrolador. [Accessed: 13-Jun-2024].

[9] "¿Qué es el wifi?," Proofpoint, 02-Jun-2023. [Online]. Available: https://www.proofpoint.com/es/threat-reference/wifi. [Accessed: 10-Jun-2024].

[10] MODULO NODEMCU *ESP32* DEVKIT V1 STARTER KIT. (n.d.). Retrieved June 17, 2024, from Puntoflotante.net website: https://www.puntoflotante.net/NODEMCU-*ESP32*-DEVKIT-V1-STARTER-KIT.htm

[11] (N.d.-b). Retrieved June 17, 2024, from Puntoflotante.net website: https://www.puntoflotante.net/*ESP32*-DOIT-DEVKIT-V1-Board-Pinout-30-GPIOs-2.jpg

[12] del Valle Hernández, L. (2017, June 20). NodeMCU y el IoT tutorial paso a paso desde cero. Retrieved June 17, 2024, from Programarfacil Arduino y Home Assistant website: https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/

[13] (N.d.-a). Retrieved June 17, 2024, from Components101.com website: https://components101.com/sites/default/files/component_pin/NodeMCU-ESP8266-Pinout.jpg

[14] Aguayo, P. (2019, January 14). Arduino UNO. Retrieved June 17, 2024, from Arduino.cl - Compra tu Arduino en Línea website: https://arduino.cl/arduino-uno/

[15] (N.d.-c). Retrieved June 17, 2024, from Arduino.cc website: https://content.arduino.cc/assets/A000066-pinout.png

[16] "*ESP32* DEVKITC V1 30 pines," AV Electronics. [Online]. Available: https://avelectronics.cc/producto/*ESP32*-devkitc-v1-30-pines/. [Accessed: 10-Jun-2024].

[17] "NodeMCU v2 ESP8266 WiFi," Naylamp Mechatronics - Perú. [Online]. Available: https://naylampmechatronics.com/espressif-esp/153-nodemcu-v2-esp8266-wifi.html. [Accessed: 10-Jun-2024].

[18] "ARDUINO," Blog de Tecnologías, 06-Jan-2020. [Online]. Available: https://www3.gobiernodecanarias.org/medusa/ecoblog/rsuagued/arduino/. [Accessed: 10-Jun-2024].

[19] J. Villegas, “¿Qué es un detector de movimiento pasivo o *PIR* y cómo funcionan los sensores de movimiento?,” Tecnoseguro.com, 02-Feb-2012. [Online]. Available: https://www.tecnoseguro.com/faqs/alarma/que-es-un-detector-de-movimiento-pasivo-o-*PIR*. [Accessed: 10-Jun-2024].

[20] “SENSOR *PIR*,” I0.wp.com. [Online]. Available: https://i0.wp.com/blog.330ohms.com/wp-content/uploads/2020/12/imagen_2020-12-09_135710.png?w=665&ssl=1. [Accessed: 10-Jun-2024].

[21] “¿Qué es un módulo de relé y qué hace?,” Geya.net, Available:https://www.geya.net/es/what-is-a-relay-module-and-what-does-it-do/ [Accessed: 10-Jun-2024].

[22] “PINES-RELE,” Com.ar. [Online]. Available: https://robots-argentina.com.ar/didactica/wp-content/uploads/006-contactos.png. [Accessed: 10-Jun-2024].

[23] Arduino IoT cloud. (n.d.). Retrieved June 18, 2024, from Catedu.es website: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/arduino-iot-cloud

[24] P. Jecrespom, “IDE Arduino,” Aprendiendo Arduino, 11-Dec-2016. [Online]. Available: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/. [Accessed: 20-Jul-2024].

[25] Introduccion A ThingSpeak. (n.d.). Retrieved June 18, 2024, from Scribd website: https://es.scribd.com/document/402371016/Introduccion-a-ThingSpeak-docx

[26] arduino IoT Cloud. (n.d.). Retrieved June 18, 2024, from Pinimg.com website: https://i.pinimg.com/736x/01/ac/d3/01acd39ba7c03f5ba4831e744fcd8865.jpg

[27] Baquelita (también llamada Bakelita) o Resina Fenólica. (2020, May 22). Retrieved July 10, 2024, from Metalplastics SAS website: https://metalplasticsas.com/product/baquelita-tambien-llamada-bakelita-o-resina-fenolica/

[28] López, P. (2019, May 22). Como fabricar tu propio *PCB*. Retrieved July 10, 2024, from Liberaturadio website: https://liberaturadio.org/como-fabricar-tu-propio-pcb/

[29] Perfil, V. (n.d.). Como hacer circuitos impresos con el método de planchado. Retrieved July 10, 2024, from Electronicaivanespinoza.com website:

http://www.electronicaivanespinoza.com/2017/09/como-hacer-circuitos-impresos-con-el.html

[30] Cómo crear y conectar un chatbot de Telegram. (n.d.). Retrieved July 11, 2024, from SendPulse website: https://sendpulse.com/latam/knowledge-base/chatbot/telegram/create-telegram-chatbot

[31] Mercado Libre, "Módulo ESP32 ESP-32 WiFi Bluetooth Arduino", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-547615288-modulo-esp32-esp-32-wifi-bluetooth-arduino-_JM#polycard_client=search-nordic&position=19&search_layout=stack&type=item&tracking_id=2d65660f-5f29-49fd-95b5-d64c2f6da834. [Accedido: 26-jul-2024].

[32] Mercado Libre, "Megatronica Módulo Sensor Movimiento PIR HC-SR501 Arduino/PIC", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-521859591-megatronica-modulo-sensor-movimiento-pir-hc-sr501-arduinopic-_JM#polycard_client=search-nordic&position=5&search_layout=stack&type=item&tracking_id=c39f8023-8302-4c31-8ac3-f9174a7b063a. [Accedido: 26-jul-2024].

[33] Mercado Libre, "MGSystem Módulo Relé Relay 5V Optoacoplador 2 Canales Arduino", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-514804575-mgsystem-modulo-rele-relay-5v-optoacoplador-2-canals-arduino-_JM#polycard_client=search-nordic&position=2&search_layout=stack&type=item&tracking_id=d294531d-68ae-401b-94d4-fdaaea661e17. [Accedido: 26-jul-2024].

[34] Mercado Libre, "Cable Jumper 40 Pines 20cm para Protoboard Arduino Dupont", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-517615926-cable-jumper-40-pines-20cm-para-protoboard-arduino-dupont-_JM#polycard_client=search-nordic&position=25&search_layout=stack&type=item&tracking_id=f67e88a3-00c2-4348-b2d9-c81770cca08b. [Accedido: 26-jul-2024].

[35] Mercado Libre, "Cable de Red UTP CAT5e 24AWG Color Gris RJ45 por Metro", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-518232968-cable-de-red-utp-cat5e-24awg-color-gris-rj45-por-metro-_JM#polycard_client=search-nordic&position=7&search_layout=stack&type=item&tracking_id=2514e86a-708e-46ab-9c57-3b17574c3917. [Accedido: 26-jul-2024].

[36] Mercado Libre, "Cable Flexible 100% Cobre", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-563487106-cable-flexible-100-

cobre-_JM#polycard_client=search-nordic&position=43&search_layout=stack&type=item&tracking_id=9b19440b-866a-4514-a093-bf3ed077be9d. [Accedido: 26-jul-2024].

[37] Mercado Libre, "Cargador de Pared USB 5V 1A para Celulares Tablets iPhone", 2024. [En línea]. Disponible en: https://articulo.mercadolibre.com.ec/MEC-517187164-cargador-de-pared-usb-5v-1a-para-celulares-tablets-iphone-_JM?searchVariation=variationID#polycard_client=search-nordic&position=32&search_layout=stack&type=item&tracking_id=15767e90-0538-4dcd-863a-ca9534b0b01a. [Accedido: 26-jul-2024].

7 ANEXOS

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Conjunto de datos extensos

ANEXO I: Certificado de Originalidad

F_AA_236

CERTIFICADO DE ORIGINALIDAD

TRABAJO DE INTEGRACIÓN CURRICULAR

Quito, D.M. 31 de julio de 2024

De mi consideración:

Yo, **LEANDRO ANTONIO PAZMIÑO ORTIZ**, en calidad de Director del Trabajo de Integración Curricular titulado **IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL CONTROL AUTOMÁTICO DEL ENCENDIDO Y APAGADO DE LAS LUMINARIAS DE LA OFICINA 8 DE LA ESFOT A TRAVÉS DE UNA PLATAFORMA WEB, APLICACIÓN EN ANDROID Y POR MEDIO DE TELEGRAM,** componente **IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL CONTROL AUTOMÁTICO DEL ENCENDIDO Y APAGADO DE LAS LUMINARIAS DE LA OFICINA 8 DE LA ESFOT A TRAVÉS DE UNA PLATAFORMA WEB, APLICACIÓN EN ANDROID Y POR MEDIO DE TELEGRAM** elaborado por el estudiante **GABRIEL ALEJANDRO QUISHPI LOACHAMIN** de la carrera en **TECNÓLOGÍA 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 11%.

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 informe generado por la herramienta Turnitin.

Atentamente,

Leandro Pazmiño Ortiz
Docente
Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

En el siguiente link, se encuentra el video de las pruebas realizadas del prototipo: https://epnecuador.sharepoint.com/:v:/s/TIC2024A_LPazmino/EZsJM6wmMy5DtGM3Z7hYRhABZgrkUhswdCimcbhr2z9VkQ?e=ZRtRGu

Anexo II.I Código QR de la implementación y pruebas de funcionamiento del prototipo de encendido y apagado de luminarias.

ANEXO III: Códigos Fuente

```
#include <WiFiClientSecure.h> // Biblioteca para cliente WiFi seguro

#include <UniversalTelegramBot.h> // Biblioteca para el bot de Telegram

#include <ArduinoJson.h> // Biblioteca para manejar JSON

#include "thingProperties.h" // Biblioteca para propiedades de Arduino IoT Cloud

#include <NTPClient.h> // Biblioteca para cliente NTP (Network Time Protocol)

#include <WiFiUdp.h> // Biblioteca para manejar conexiones UDP


// Variables para el bot de telegram

const char* bot_token = "7417029630:AAHAQ2KAApvnVSReLn8s9abaf-UYfZD2gzs"; // Token del bot de Telegram

const String chat_id_aux = "1324656447"; // ID de chat del usuario

String chat_id; // Variable para almacenar el chat ID


// Variables de conexión a bot de Telegram

WiFiClientSecure client; // Cliente seguro WiFi

UniversalTelegramBot bot(bot_token, client); // Bot de Telegram


// Pines del ESP32

const int rele = 5; // Pin del relé

const int boton = 35; // Pin del botón

const int PIR = 27; // Pin del sensor PIR


// Variables de tiempo

int botRequestDelay = 1000; // Verificar bot cada 1000ms

unsigned long lastTimeBotRan; // Última vez que se ejecutó el bot
```

```
// Variables booleanas
bool aux = false; // Variable auxiliar para control
bool aux2 = true; // Variable auxiliar para control
bool scheduleActive = false; // Indica si el calendario está activo
bool detecto_movimiento = false; // Indica si se detectó movimiento
bool boton_presionado = false; // Indica si el botón está presionado
bool mov = false; // Auxiliar para control del PIR
bool txmsg_state = true; // Auxiliar para enviar mensajes

bool cumple_hora = false; // Indica si se cumple la hora programada
bool porHorario = false; // Indica si está dentro del horario

// Configuración del cliente NTP
WiFiUDP ntpUDP; // Objeto UDP
NTPClient timeClient(ntpUDP, "europe.pool.ntp.org", -18000, 60000); // Cliente NTP configurado para UTC-5, sincroniza cada 60 segundos

// Variables de horario de Telegram
int startHour = 0; // Hora de inicio
int startMinute = 0; // Minuto de inicio
int endHour = 0; // Hora de fin
int endMinute = 0; // Minuto de fin

// Configuración inicial
void setup() {
  Serial.begin(115200); // Inicializa la comunicación serial a 115200 baudios
  initProperties(); // Inicializa propiedades definidas en thingProperties.h
```

```
  ArduinoCloud.begin(ArduinoIoTPreferredConnection); // Conecta a Arduino IoT Cloud
  setDebugMessageLevel(2); // Establece nivel de mensajes de depuración
  ArduinoCloud.printDebugInfo(); // Imprime información de depuración
  client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Configura el certificado raíz de Telegram

  // Configuración física de los pines
  pinMode(rele, OUTPUT); // Configura el pin del relé como salida
  pinMode(boton, INPUT); // Configura el pin del botón como entrada
  pinMode(PIR, INPUT); // Configura el pin del PIR como entrada
  digitalWrite(rele, LOW); // Establece el estado inicial del relé a LOW
}

// Bucle principal
void loop() {
  ArduinoCloud.update(); // Actualiza la conexión a Arduino Cloud
  if (ArduinoCloud.connected()) { // Si está conectado a Arduino Cloud
    aux = true; // Establece auxiliar a verdadero
  }

  while (aux) {
    if (aux2 && WiFi.status() == WL_CONNECTED) { // Si está conectado al WiFi
      timeClient.begin(); // Inicia el cliente NTP
      aux2 = false; // Establece auxiliar 2 a falso
    }
    ArduinoCloud.update(); // Actualiza la conexión a Arduino Cloud
```

```
  if (digitalRead(PIR)) { // Si el PIR detecta movimiento
    detecto_movimiento = true; // Establece detecto_movimiento a verdadero
  } else {
    detecto_movimiento = false; // Establece detecto_movimiento a falso
  }

  // Manejo del scheduler con Telegram
  timeClient.update(); // Actualiza el cliente NTP
  handleSchedule(); // Verifica el calendario configurado en Telegram

  // Manejo del scheduler con el widget de Arduino Cloud
  if (activar_calendario && !scheduleActive) { // Si el calendario está activado y no está activo el scheduler
    if (calendario.isActive()) { // Si el calendario está activo
      cumple_hora = true; // Establece cumple_hora a verdadero
      porHorario = true; // Establece porHorario a verdadero
    } else {
      cumple_hora = false; // Establece cumple_hora a falso
    }
    if (porHorario && !cumple_hora && digitalRead(rele)) { // Si está dentro del horario pero no cumple la hora y el relé está encendido
      bot.sendMessage(chat_id_aux, "Luz encendida fuera de horario, apagando…", ""); // Envía mensaje por Telegram
      delay(2000); // Espera 2 segundos
      digitalWrite(rele, LOW); // Apaga el relé
      porHorario = false; // Establece porHorario a falso
    }
  }
```

```
  control_PIR(); // Controla el sensor PIR

  // Función para el manejo del bot de Telegram
  handleTelegramBot(); // Maneja los comandos del bot de Telegram

  // Manejo del toggle del botón físico
  if (digitalRead(boton) && !digitalRead(rele)) { // Si el botón está presionado y el relé está apagado
    digitalWrite(rele, HIGH); // Enciende el relé
  } else if (digitalRead(boton) && digitalRead(rele)) { // Si el botón está presionado y el relé está encendido
    digitalWrite(rele, LOW); // Apaga el relé
  }

  // Manejo del toggle del botón de la nube
  if (activar_foco && !digitalRead(rele)) { // Si el foco está activado y el relé está apagado
    digitalWrite(rele, HIGH); // Enciende el relé
  } else if (activar_foco && digitalRead(rele)) { // Si el foco está activado y el relé está encendido
    digitalWrite(rele, LOW); // Apaga el relé
  }

  // Manejo de indicadores en el dashboard
  estado_rele = digitalRead(rele); // Actualiza el estado del relé en el dashboard
  movimiento_detectado = digitalRead(PIR); // Actualiza el estado del movimiento en el dashboard
```

```
    // Manejo de notificaciones por Telegram
    if (digitalRead(rele) && txmsg_state) { // Si el relé está encendido y el estado del mensaje es verdadero
      bot.sendMessage(chat_id_aux, "Luminarias Encendidas", ""); // Envía mensaje por Telegram
      txmsg_state = false; // Establece estado del mensaje a falso
    } else if (!digitalRead(rele) && !txmsg_state) { // Si el relé está apagado y el estado del mensaje es falso
      bot.sendMessage(chat_id_aux, "Luminarias Apagadas", ""); // Envía mensaje por Telegram
      txmsg_state = true; // Establece estado del mensaje a verdadero
    }
    delay(10); // Espera 10 ms
  }
  delay(10); // Espera 10 ms
}

// Función para controlar el sensor PIR
void control_PIR() {
  if (cumple_hora) { // Si cumple la hora
    if (digitalRead(PIR) && !digitalRead(rele) && !mov) { // Si el PIR detecta movimiento, el relé está apagado y mov es falso
      digitalWrite(rele, HIGH); // Enciende el relé
    }
    if (!digitalRead(PIR) && digitalRead(rele)) { // Si el PIR no detecta movimiento y el relé está encendido
      mov = true; // Establece mov a verdadero
    }
```

```
    if (digitalRead(PIR) && digitalRead(rele) && mov) { // Si el PIR detecta movimiento, el relé está encendido y mov es verdadero
      digitalWrite(rele, LOW); // Apaga el relé
    }
    if (!digitalRead(PIR) && !digitalRead(rele)) { // Si el PIR no detecta movimiento y el relé está apagado
      mov = false; // Establece mov a falso
    }
  }
}

// Función para el manejo del bot de Telegram
void handleTelegramBot() {
  if (millis() > lastTimeBotRan + botRequestDelay) { // Si ha pasado el tiempo de delay para el bot
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1); // Obtiene los nuevos mensajes
    while (numNewMessages) { // Mientras haya nuevos mensajes
      Serial.println("se recibe un mensaje"); // Imprime mensaje recibido

      handleNewMessage(numNewMessages); // Maneja los nuevos mensajes
      numNewMessages = bot.getUpdates(bot.last_message_received + 1); // Actualiza los nuevos mensajes
    }
    lastTimeBotRan = millis(); // Actualiza el tiempo de la última ejecución del bot
  }
}
```

```
// Función para manejar nuevos mensajes de Telegram
void handleNewMessage(int messageIndex) {
  for (int i = 0; i < messageIndex; i++) { // Itera sobre los mensajes
    chat_id = String(bot.messages[i].chat_id); // Obtiene el chat ID
    String text = bot.messages[i].text; // Obtiene el texto del mensaje

    if (text == "/iniciar") { // Si el texto es "/iniciar"
      String sms = "Control luminaria mediante PIR y horarios\n"; // Mensaje de bienvenida
      sms += "==== MENU DE OPCIONES ====\n\n";
      sms += "/proginiciar HH:MM, hora de encendido de luminarias \n";
      sms += "/progfinal HH:MM, hora de apagado de luminarias \n";
      sms += "/iniciarhorario: para activar horario configurado en telegram\n";
      sms += "/pararhorario: para desactivar horario configurado en telegram\n";
      sms += "/encender, encender luminaria\n";
      sms += "/apagar, apagar luminaria\n";

      bot.sendMessage(chat_id, sms, ""); // Envía el mensaje de bienvenida
    }
    // Scheduler - setear una hora de inicio
    if (text.startsWith("/proginiciar ")) { // Si el texto empieza con "/proginiciar "
      int colonIndex = text.indexOf(':'); // Encuentra el índice del caracter ':'
      if (colonIndex > 0) { // Si el índice es válido
        startHour = text.substring(13, colonIndex).toInt(); // Obtiene la hora de inicio
        startMinute = text.substring(colonIndex + 1).toInt(); // Obtiene el minuto de inicio
        bot.sendMessage(chat_id, "Hora de inicio establecida a " + String(startHour) + ":"
+ String(startMinute), ""); // Envía mensaje de confirmación
      }
```

```
  }

  // Scheduler - setear una hora de fin
  if (text.startsWith("/progfinal ")) { // Si el texto empieza con "/progfinal "
    int colonIndex2 = text.indexOf(':'); // Encuentra el índice del caracter ':'
    if (colonIndex2 > 0) { // Si el índice es válido
      endHour = text.substring(11, colonIndex2).toInt(); // Obtiene la hora de fin
      endMinute = text.substring(colonIndex2 + 1).toInt(); // Obtiene el minuto de fin
      bot.sendMessage(chat_id, "Hora de fin establecida a " + String(endHour) + ":" + String(endMinute), ""); // Envía mensaje de confirmación
    }
  }

  // Scheduler - iniciar el scheduler
  if (text == "/iniciarhorario" && !activar_calendario) { // Si el texto es "/iniciarhorario" y no está activado el calendario
    scheduleActive = true; // Activa el scheduler
    bot.sendMessage(chat_id, "Horario por telegram activado", ""); // Envía mensaje de confirmación
  } else if (text == "/iniciarhorario" && activar_calendario) { // Si el texto es "/iniciarhorario" y el calendario está activado
    bot.sendMessage(chat_id, "El horario de encendido ya se encuentra activo en Arduino Cloud.", ""); // Envía mensaje de aviso
  }

  // Scheduler - parar el scheduler
  if (text == "/pararhorario") { // Si el texto es "/pararhorario"
    scheduleActive = false; // Desactiva el scheduler
```

```
      bot.sendMessage(chat_id, "Horario desactivado", ""); // Envía mensaje de confirmación
    }

    // Encender luminarias
    if (text == "/encender") { // Si el texto es "/encender"
      digitalWrite(rele, HIGH); // Enciende el relé
    }
    // Apagar luminarias
    if (text == "/apagar") { // Si el texto es "/apagar"
      digitalWrite(rele, LOW); // Apaga el relé
    }
  }
}

// Función para manejar el calendario
void handleSchedule() {
  if (scheduleActive) {// Si el scheduler está activo
    int currentHour = timeClient.getHours(); // Obtiene la hora actual
    int currentMinute = timeClient.getMinutes(); // Obtiene el minuto actual

    if (currentHour == startHour && currentMinute == startMinute) {// Si es la hora de inicio
      cumple_hora = true; // Establece cumple_hora a verdadero
      porHorario = true; // Establece porHorario a verdadero
    }

    if (currentHour == endHour && currentMinute >= endMinute) {// Si es la hora de fin
```

```
      cumple_hora = false; // Establece cumple_hora a falso
    }
    if (porHorario && !cumple_hora && digitalRead(rele)) {// Si está dentro del horario, pero no cumple la hora y el relé está encendido
      bot.sendMessage(chat_id_aux, "Luz encendida fuera de horario, apagando…", ""); // Envía mensaje por Telegram
      delay(2000); // Espera 2 segundos
      digitalWrite(rele, LOW); // Apaga el relé
      porHorario = false; // Establece porHorario a falso
    }
  }
}

// Función para manejar cambios en MovimientoDetectado
void onMovimientoDetectadoChange() {
  // Añade tu código aquí para actuar cuando MovimientoDetectado cambie
}

// Función para manejar cambios en EstadoRele
void onEstadoReleChange() {
  // Añade tu código aquí para actuar cuando EstadoRele cambie
}

// Función para manejar cambios en ActivarCalendario
void onActivarCalendarioChange() {
  // Añade tu código aquí para actuar cuando ActivarCalendario cambie
}
```

```
// Función para manejar cambios en Calendario
void onCalendarioChange() {
  // Añade tu código aquí para actuar cuando Calendario cambie
}

// Función para manejar cambios en ActivarFoco
void onActivarFocoChange() {
  // Añade tu código aquí para actuar cuando ActivarFoco cambie
}
```