



Universidad Nacional Experimental Politécnica
"Antonio José de Sucre"
Vice-Rectorado Académico "Puerto Ordaz"
Departamento de Ingeniería Electrónica
Trabajo de Grado

SISTEMA DE CONTROL DOMOTICO MEDIANTE LA PLACA DE DESARROLLO ESP32

Autor: Br. Hernández Luixandris

Ciudad Guayana, junio de 2021

**SISTEMA DE CONTROL DOMOTICO MEDIANTE LA PLACA
DE DESARROLLO ESP32**



Universidad Nacional Experimental Politécnica
“Antonio José de Sucre”
Vice-Rectorado Académico “Puerto Ordaz”
Departamento de Ingeniería Electrónica
Trabajo de Grado

**SISTEMA DE CONTROL DOMOTICO MEDIANTE LA PLACA DE
DESARROLLO ESP32**

Autor: Br. Hernández Luixandris

Tutor: Ing. Pateti Antonio

Ciudad Guayana, junio de 2021

Hernández Cova, Luixandris Elena

**“Sistema de control domótico mediante la placa de
desarrollo ESP32”
(2021).**

Páginas: 126

Universidad Nacional Experimental Politécnica “Antonio José de Sucre”

Vicerrectorado Puerto Ordaz.

Departamento de Ingeniería Electrónica

Tutor académico: Ing. Antonio Pateti.

DEDICATORIA

A papá Luis, mis tíos: Alfredo, Luirvis y Javier y a mi hermano Roberto. Donde sea que estén espero se sientan orgullosos de mí.

Siempre los llevo en mi corazón.

AGRADECIMIENTOS

¡Toda honra y gloria sea para Dios! Primeramente, me encuentro agradecida con mi Dios por la sabiduría que me dió para culminar esta meta tan importante en mi vida; sin El nada de esto hubiese sido posible.

A mi madre y mejor amiga, Marlin Hernández, por todo el amor, comprensión, cariño y apoyo que me ha brindado durante toda mi vida. Gracias por haberme criado con excelentes principios y valores que me han forjado como persona y profesional; gracias por estar conmigo en todas las noches de desvelos y por aguantar mis cambios de humor cuando algo no salía como esperaba. Te amo inmensamente.

A mi padre, Jimmy Márquez, por haberme llenado de amor y fortaleza aun cuando él no la tenía. Por enseñarme a luchar por mis sueños y formarme como una mujer independiente y temerosa del bien. Por haberme criado de la mejor forma, con el ejemplo. Gracias por todos los sacrificios que has hecho para darme lo mejor.

A mis hermanos Luiximar Hernández, Jeimmy Márquez y Jimmy Márquez. Siempre serán mi más grande motivación para superarme en la vida; gracias por toda la ayuda y apoyo durante mi carrera y por toda la paciencia que me tenían cuando me tocaba estudiar por semanas enteras.

A mi madre Rosa Cova y mi tía, Daidy Hernández; por ser pilares fundamentales en mi vida y siempre creer en mí, motivándome a seguir adelante y llenándome de palabras de aliento en mis bajones de ánimo.

A todos mis amigos y compañeros que estuvieron conmigo en mi aventura universitaria, Oriana Garrido, Anyelines Muñoz, Fernando Túnez, Ángel Sifontes, Andrea Hernández y en especial, Cristhian Urbaez, por ser mi más grande amigo

y apoyo en mi formación académica, por impulsarme y a veces obligarme a seguir adelante. Siempre te estaré agradecida.

A mi tutor, Ing. Antonio Pateti, por haberme orientado y apoyado en el desarrollo de este proyecto. Gracias a la forma en como daba clases y la pasión al transferir sus conocimientos, me enfoqué en el campo de programación de micro-controladores. ¡Excelente profesor!

Para concluir, gracias Señor por haber puesto en mi camino a las más increíbles personas, sin duda alguna los mejores.

SISTEMA DE CONTROL DOMOTICO MEDIANTE LA PLACA DE DESARROLLO ESP32.

Autor: Hernández Luixandris.

Tutor: Ing. Pateti Antonio.

Universidad Nacional Experimental Politécnica Antonio José de Sucre.
Vicerrectorado Puerto Ordaz.
Departamento de Ingeniería Electrónica

RESUMEN

El presente trabajo tiene como objetivo el desarrollo de un sistema domótico utilizando la placa ESP32 y a su vez el diseño e implementación de una aplicación móvil que permita monitorear y controlar las variables que el usuario desee.

Para el desarrollo y programación del ESP32, se utilizó la IDE de Arduino como entorno de desarrollo, siendo éste el más amigable y de fácil manejo para los requerimientos de nuestro sistema. Las variables que se han de monitorear y/o controlar son: temperatura, humedad, luz y movimiento; para ello se utilizarán sensores que nos proporcionen los datos necesarios, los cuales se guardarán en una base de datos en Firebase. La aplicación móvil se diseñó con el framework Flutter, el cual es ideal para el desarrollo de aplicaciones nativas. Flutter utiliza DART como lenguaje de programación; todo ello se ejecuta en el editor de código fuente Visual Studio Code y se emula en el dispositivo móvil.

Palabras claves: ESP32, Arduino IDE, aplicaciones nativas, Flutter, Dart, Real time Firebase.

INDICE

INTRODUCCION	1
CAPITULO I	3
EL PROBLEMA.....	3
Planteamiento del Problema	3
Objetivos	4
Objetivo General.....	4
Objetivos Específicos	4
Justificación.....	5
Delimitación o Alcance	6
CAPITULO II	7
MARCO TEORICO	7
Antecedentes de la Investigación.....	7
Bases teóricas.....	9
Origen de la Domótica.	9
Domótica.....	10
Internet.....	12
Protocolo TCP/IP.	13
Ventajas del modelo TCP/IP.	13
Wifi.....	14
Bluetooth.....	15
Esp32	16
Bluetooth Low Energy ESP32.....	20
Sensor de temperatura DTH11	23

Sensor de movimiento PIR	24
Arduino IDE	28
Flutter.....	29
DART.....	30
Visual Studio Code.	31
Firebase.....	33
CAPITULO III	37
MARCO REFERENCIAL	37
Tipo de investigación.....	37
Diseño de investigación	38
Unidades de análisis	39
Eventos o variables	40
Técnicas e instrumentos de recolección de datos.....	40
Libros o textos especializados	40
Internet.....	40
Trabajos de grados relacionados con el presente proyecto	40
Tutoriales básicos.....	41
Computadora personal	41
Realtime Database Firebase	41
Arduino IDE	42
Visual Studio Code	42
Flutter.....	42
CAPITULO IV.....	43
DISEÑO	43
Requerimientos del Sistema en general.....	43

Requerimientos del hardware	43
Requerimientos del Software.....	44
Descripción del diseño	44
Propuesta de hardware	45
Placa de desarrollo Esp32.	45
Propuesta de Software.....	49
Configuración de credenciales WiFi vía Bluetooth.....	50
Configuración de sensores y elementos en el ESP32	58
Comunicación ESP32 con Firebase:	64
Aplicación móvil	70
CAPITULO V.....	89
RESULTADOS.....	89
Pruebas con Arduino IDE Modo Wifi - Bluetooth:.....	89
Pruebas con Sensores:	93
CONCLUSIONES	100
RECOMENDACIONES.....	101
REFERENCIAS BIBLIOGRAFICAS.....	102
ANEXOS	104
Código ESP32.....	104

INDICE DE FIGURAS

Figura 1: Conexión de la red HAN.	12
Figura 2: Bluetooth.....	15
Figura 3: Diagrama de bloques Esp32.....	20
Figura 4. Transmisión BLE.	21
Figura 5. Servidor y cliente BLE.....	22
Figura 6. GATT ESP32	22
Figura 7. Sensor DTH11.	24
Figura 8: Sensor PIR y lente de Fresnel.	25
Figura 9: Detección de movimiento Sensor PIR	26
Figura 10. Conexión sensor PIR	27
Figura 11: Arduino IDE	29
Figura 12: Pantalla de inicio Visual Studio Code.	32
Figura 13: Visualización grafica de un programa en Visual Studio Code.....	32
Figura 14. Realtime database Firebase	36
Figura 15. Diseño general del sistema.....	44
Figura 16. Placa de desarrollo esp32	46
Figura 17. Esquema del hardware	47
Figura 18. Etapas para la programación del software.....	49
Figura 19. Diagrama de flujo conexión inicial Wifi – Bluetooth	50
Figura 20. Diagrama de flujo Función Bluetooth	51
Figura 21. Diagrama de flujo Funcion MyCallbacks	52
Figura 22. Librerías del programa para el ESP32	53
Figura 23. Void Setup ESP32	54

Figura 24. Funcion BLE	55
Figura 25. Funcion MyCallbacks	55
Figura 26. Diagrama de flujo Funcion Wifi	56
Figura 27. Librería Wifi.h.....	57
Figura 28. Función Wifi	57
Figura 29. Diagrama de flujo de sensor DTH11.....	58
Figura 30. Diagrama de flujo funcion Temperatura.....	59
Figura 31. Funcion Temperatura.....	59
Figura 32. Diagrama de flujo Funcion Humedad.....	60
Figura 33. Función Humedad.....	61
Figura 34. Declaración de constante PIR	62
Figura 35. Configuración PIR.....	62
Figura 36. Función para la interrupción PIR.	63
Figura 37. Diagrama de flujo status de las luces para ON/OFF.....	63
Figura 38. Comprobación status de las luces para ON/OFF room	64
Figura 39. Página web Firebase	64
Figura 40. Creación de un proyecto en Firebase.	65
Figura 41. Creación de base de datos.	65
Figura 42. Configuración de reglas Firebase.	66
Figura 43. Configuración general del proyecto.	66
Figura 44. DatabaseURL del proyecto.....	67
Figura 45. Token de la base de datos.....	67
Figura 46. Librería FirebaseESP32.h.....	68
Figura 47. Datos de Firebase en código ESP32.....	68
Figura 48. Inicialización BD Firebase.....	68

Figura 49. Host remoto.	69
Figura 50. Esquema Aplicación móvil	70
Figura 51. Configuración general del proyecto Firebase.	71
Figura 52. Agregar app a Firebase.	71
Figura 53. Registro de la app.....	72
Figura 54. Descarga de archivo .Json	73
Figura 55. Archivo google-services.json en VS CODE.	73
Figura 56. Modificación de archivo build.gradle	74
Figura 57. Modificación de archivo build.gradle en VS CODE	74
Figura 58. Modificación de archivo app/ build.gradle	75
Figura 59. Modificación de archivo app/ build.gradle en VS CODE	75
Figura 60. Paquetes y bibliotecas para Bluetooth VS.	76
Figura 61. StatefulWidget para bluetooth.....	77
Figura 62. Inicialización del escaneo bluetooth.....	77
Figura 63. Diseño configuración WiFi ESP32.	78
Figura 64. Credenciales WiFi.....	78
Figura 65. Composición base de la app.....	79
Figura 66. Imagen principal de la app.....	80
Figura 67. Selector de variable.	81
Figura 68. AppBar.....	81
Figura 69. Contenedor 1.	82
Figura 70. Contenedor 2	82
Figura 71. Estructura de Widget Row().	83
Figura 72 (a). Row () Temperatura	84
Figura 73. Programación para visualización de los datos de temperatura....	84

Figura 74. Stack() Movimiento	85
Figura 75 Stack Movimiento en “ALERTA”.	85
Figura 76. Programación Stack() Movimiento.	86
Figura 77. Diseño para luces	86
Figura 78. ToggleButtons con todas las luces encendidas.	87
Figura 79. Comprobación del botón seleccionado.	87
Figura 80. Selector de variable	88
Figura 81. Arduino IDE, Modo Bluetooth.	89
Figura 82. Escaneo y conexión del bluetooth en la app.....	90
Figura 83. Credenciales Wifi.	91
Figura 84. ESP32 leyendo las credenciales recibidas a través de la app.	92
Figura 85. Modo WIFI.	92
Figura 86. Lectura de datos de temperatura y humedad	93
Figura 87. Recepción de datos en Realtime Firebase.	94
Figura 88. Datos recibidos en App móvil.	94
Figura 89. Sensor de movimiento	95
Figura 90. Detección de movimiento en Firebase.....	95
Figura 91. Detección de alarma en app móvil.....	96
Figura. 92. Sin detección de movimiento en app móvil.....	96
Figura 93. Encendido de luz del baño.....	97
Figura 94. Status de luz del baño en Firebase.....	98
Figura 95. Chequeo del status de las luces en el ESP32.	98
Figura 96. Encendido de luz del baño, ESP32.....	99
Figura 97. SplashScreen App móvil.....	99

INTRODUCCION

Los sistemas automatizados son el pilar actual de cualquier industria; esto se debe a que se acelera la producción, reduce costos, previene los errores por parte humana y ahorra tiempo a los operarios encargados de manipular los sistemas. La automatización no solo se aplica a nivel industrial, sino también en los hogares. Por ello, desde la década de los 70 surge la domótica (Automatización de casas).

La domótica promete simplificar las tareas sencillas que se ejecutan en casa y aparte de brindar comodidad y confort, reduce considerablemente el consumo de energía; lo cual es beneficioso para el ambiente y permite el ahorro de gastos a los usuarios. Esta rama de la electrónica y programación ha tenido un crecimiento exponencial durante los últimos años, y cada vez se incluyen mejoras en los sistemas, teniendo realmente una casa inteligente.

El presente trabajo, propone el diseño de un sistema de control domotico que se encargará del control y monitoreo de variables como temperatura, humedad, movimiento y luces; todas ellas, siendo visualizadas y/o controladas a través de una aplicación móvil desarrollada en el proyecto.

La investigación se encuentra constituida por cinco capítulos según la siguiente estructura:

En el Capítulo I, se aborda el problema de la investigación (planteamiento del problema, objetivos, justificación y delimitación o alcance).

En el Capítulo II, se detallan los antecedentes y bases teóricas que sustentan la investigación de este proyecto.

En el Capítulo III se define el Marco metodológico, indicando cual es tipo de investigación que se lleva a cabo, las herramientas que se han de utilizar para el desarrollo de las misma y como se plantea la recolección de datos.

El Capítulo IV contiene el diseño del sistema, explicando a detalle la propuesta y el hardware y software empleado.

En el capítulo V se muestran los resultados obtenidos del proyecto, las pruebas realizadas y la comprobación de la eficiencia del mismo.

Por último, se añaden las conclusiones, recomendaciones, referencias bibliográficas y anexos de la investigación.

CAPITULO I

EL PROBLEMA

Planteamiento del Problema

Desde la antigüedad la humanidad ha buscado la manera de automatizar sus labores, haciendo de estas una tarea más fácil y sencilla, además de optimizar el entorno donde se desarrolla con el objetivo de disminuir considerablemente el tiempo para el cumplimiento de los deberes; por ello, a partir de los años 70 surge la domótica (sistemas capaces de automatizar una vivienda o edificación de cualquier tipo) como una solución para el control de las casas, sin necesidad de la presencia de los habitantes. La domótica contribuye a mejorar la calidad de vida del usuario, lo que permite una gestión eficiente del uso de la energía (siendo esto fundamental en la actualidad debido a la crisis energética mundial), aporta seguridad y confort, y permite realizar la interacción y/o comunicación entre el sistema y el usuario.

Para implementar un sistema domótico óptimo es necesario encontrar el balance de tres variables fundamentales, estas son: precio, comodidad y eficiencia; por eso, se plantea el uso de la placa de desarrollo esp32 como controlador principal del sistema. Este dispositivo lanzado hace pocos años, presenta múltiples ventajas y sólidas características en el área de automatización social e industrial; además de su bajo costo en el mercado y su fácil programación. El esp32 integra Wifi y bluetooth, lo cual lo hace un chip ideal para aplicaciones IOT (por sus siglas en inglés Internet of Things) o El internet de las cosas.

El sistema propuesto, prevé garantizar al usuario el control de las luces de forma automática o manual, a través de dispositivos inteligentes como teléfonos o

tabletas; tener un registro de temperatura y humedad, de manera que, si el pH requerido en la casa para un ambiente agradable se pasa de los límites, se accione un sistema de ventilación que asegure la temperatura y humedad configurada; disponer de un sensor de gas para detectar fugas y evitar accidentes en la vivienda; y por último, un sistema de seguridad donde se utilizaran sensores de movimientos, que tienen como finalidad detectar la presencia de personas no autorizadas en la casa y de accionar alarmas indicando la intrusión.

Objetivos

Objetivo General

Desarrollar un prototipo de un sistema domótico con la placa de desarrollo esp32 para el control de luces, temperatura, humedad y movimiento, a través de una aplicación móvil.

Objetivos Específicos

- Determinar las características de la placa esp32 para la implementación del sistema de control.
- Diseñar los aspectos a automatizar con el sistema domótico.
- Integrar los elementos de hardware y software utilizados en el prototipo.
- Enviar las variables del sistema a un Servidor WEB ya preestablecido para su posterior visualización.
- Probar el funcionamiento del proyecto en una aplicación móvil.

Justificación

“La domótica es el conjunto de técnicas aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, aporta seguridad y confort, además de comunicación entre el usuario y el sistema”. (Asociación Española de Domótica e Inmotica). Esta, ha tenido un auge en los últimos años debido al avance de la tecnología y la necesidad de controlar más, usando menos; es por ello, que hoy en día hay una innumerable cantidad de empresas que se dedican exclusivamente al desarrollo de esta ciencia.

Un sistema domótico tiene como objetivo garantizar 3 aspectos fundamentales para que se considere óptimo, estos son: precio, comodidad y eficiencia; por dicha razón se escoge la placa de desarrollo esp32 como controlador principal del proyecto. El esp32 es un potente dispositivo que cada vez sorprende más a los profesionales y aficionados de la tecnología, por sus múltiples ventajas y el amplio alcance que posee; además de ser una placa de bajo costo y alta funcionalidad.

La implementación de este sistema en comparación con las casas tradicionales, presenta notables ventajas para el usuario y el medio ambiente. Por un lado, permite que el usuario tenga el control de su casa de una manera rápida y sencilla, logrando manejar y/o controlar los aspectos principales de su hogar a través de un dispositivo inteligente; además de garantizar la seguridad de la vivienda al accionar alarmas en presencia de extraños; por otro lado, orienta la casa a tener intrínsecamente un sistema ecológico, debido al ahorro energético que proporciona al gestionar inteligentemente la iluminación, ventilación, entre otros; aunado a esto, mediante la monitorización del consumo, se obtiene información necesaria para modificar los hábitos y aumentar el ahorro y la eficiencia.

Una casa domótica, facilita el manejo del hogar a personas con discapacidades, convirtiéndola en una alternativa al alcance de todos.

Delimitación o Alcance

El presente proyecto tiene como objeto de estudio ampliar los conocimientos adquiridos durante la carrera académica, en el área de domótica, programación de dispositivos electrónicos (como es el caso de la placa de desarrollo a utilizar, esp32), diseño e implementación de sistemas automáticos; esto, con la finalidad de desarrollar un sistema automatizado para las casas, a bajo costo y eficiente, en donde todas las personas puedan tener el control de las áreas más importantes de su hogar, pese a las discapacidades que se puedan o no tener.

CAPITULO II

MARCO TEORICO

Antecedentes de la Investigación

ESP32 es el nombre de una serie de microcontroladores diseñados por EspressifSystems, una empresa china situada en Shanghai. Dicha empresa comenzó a comercializar módulos WiFi fabricados por Ai-Thinker, que permitían a otros microcontroladores conectarse a redes inalámbricas y realizar conexiones TCP/IP usando comandos AT. Este módulo sale al mercado a finales de septiembre del año 2016, y desde su fecha de lanzamiento no ha dejado de sorprender por sus interesantes características en el mundo de la automatización de las cosas, a través de internet. La investigación fue llevada a cabo por Álvaro Benito Herranz (2019) de la Universidad De Alcalá Escuela Politécnica Superior, para optar por el título de Ingeniero en tecnologías de las telecomunicaciones; por eso, presento su trabajo de grado *Desarrollo de aplicaciones para IoT con el módulo esp32*. (Herranz, 2019)

La investigación establece que la placa de desarrollo esp32, presenta potentes funcionalidades que pueden ser aplicadas en la mayoría de las áreas de ingeniería, aunado a esto, recomienda el uso de herramientas como ThingSpeak e IFTTT, debido a que facilitan la interacción entre el usuario y el dispositivo, permiten analizar datos mediante Matlab y tener acceso a diferentes servicios y plataformas como Twitter y Google Assistant.

Para el año 2018 en la Universidad Distrital Francisco José de Caldas, fue presentado el trabajo especial de grado *Desarrollo de un prototipo basado en Blockchain aplicado a la plataforma IoT sobre un sistema embebido* por Esteban

Adrián Restrepo y Daniel Arturo Olaya, con el objetivo de optar por el título de ingeniero electrónico. Este trabajo de investigación describe el desarrollo de un dispositivo IoT (Internet de las cosas, por sus siglas en inglés *Internet of Things*) que funciona bajo el protocolo MQTT con conexión WiFi basado en el sistema embebido esp32, el cual permitió adquirir el valor de medición de un sensor y visualizarlo a través de un dispositivo Android, todo esto gracias al manejo de los Topics de Suscribe y Publish, que permiten enviar y recibir información entre dispositivos IoT. (Restrepo & Olaya, 2013)

El trabajo ayudó en la comprensión del procesamiento de datos con Blockchain debido a que este, garantiza seguridad en las redes IoT y ofrece una mejor alternativa al protocolo MQTT, pues se elimina la necesidad de un Broker; además, permite a los dispositivos interactuar directamente con otros dispositivos de manera abierta y descentralizada. La red Blockchain proporciona un buen nivel de seguridad, ya que, para corromper los datos en los bloques, es necesario tener un poder computacional mayor al generado por el 50% de los nodos conectados y tener acceso a más del 50% de los nodos de la red Blockchain, algo que es muy difícil de lograr.

En la actualidad, la placa de desarrollo esp32 es considerada una herramienta asequible y ventajosa para aplicaciones de domótica; siendo esta, la solución planteada para aprovechar la energía eléctrica y así contribuir con el medio ambiente. Francisco Javier Calvo Torres, en su trabajo para optar por el título de Ingeniero Civil Electrónico de la Universidad Austral de Chile (año 2014), presenta el trabajo de grado *Análisis de diseño de una red domótica para viviendas sociales*, en el que expresa la necesidad de implementar un sistema domótico en los hogares, con el fin de garantizar seguridad, comodidad y eficiencia. Además, asegura que esta implementación ofrece una reducción considerable de la energía y una caída del 33% del consumo en stand by, si tan solo se desconectan los enchufes ocho horas diarias. (Torres, 2014)

Bases teóricas

Origen de la Domótica.

(Huidobro & Milan, Domotica. Edificios Inteligentes, 2004) recogen que el origen de la Domótica se remonta a los años setenta, cuando en Estados Unidos aparecieron los primeros dispositivos de automatización de edificios basados en la aún hoy exitosa tecnología X-10 (protocolo de comunicación que se rige bajo el accionar de un control remoto), desarrollada en Escocia en 1976 por la empresa Pico Electronics. Estas incursiones primerizas se alternaron con la llegada de nuevos sistemas de calefacción y climatización orientados al ahorro de energía, en clara sintonía con las crisis del petróleo. Los primeros equipos comerciales se limitaban a la colocación de sensores y termostatos que regulaban la temperatura ambiente. La disponibilidad y proliferación de la electrónica de bajo coste favoreció la expansión de este tipo de sistemas, despertando así el interés de la comunidad internacional por la búsqueda de la casa ideal.

El verdadero punto de quiebre en la historia de la domótica, ocurrió a finales de los años ochenta y principios de los noventa, momento en el cual los computadores personales comenzaron su boom de ventas y se convirtieron en un artículo de primera necesidad en las oficinas. La presencia de estos nuevos dispositivos llevó a que se comenzara a instalar en los edificios el Sistema de Cableado Estructurado o más conocido por sus siglas SCE, el cual permitía transportar internamente datos y voz. A partir de ese momento, los edificios que contaban con dicho sistema comenzaron a ser llamados Edificios Inteligentes.

A comienzos de los años noventa este tipo de tecnología que sólo era utilizada con fines comerciales, comenzó a expandirse hacia los hogares y con los avances en las redes informáticas de comunicación, el WIFI y una evolución de los protocolos de comunicación sentados por el sistema X-10, han permitido que en la actualidad se ofrezcan un sin fin de posibilidades para crear verdaderas casas inteligentes. En la actualidad es posible implementar soluciones destinadas a

brindar al usuario mayor confort, seguridad y ahorro energético que en un principio eran impensables.

Domótica.

Es el conjunto de tecnologías aplicadas al control y automatización inteligente de una vivienda. La palabra proviene de latín domus y tica, que significan casa y automático, respectivamente; es decir, casa automática. Consiste en la integración de los diversos sistemas o subsistemas que se encuentran presentes en cualquier vivienda, con el objetivo de poner a disposición del usuario una plataforma capaz de ofrecer una mayor comodidad y versatilidad de control. La idea de la integración es que estos elementos interactúen entre sí para conseguir un fin en común.

Para (Huidobro & Milan, Domotica. Edificios Inteligentes, 2004) la Domótica se aplica a los sistemas y dispositivos que proporcionan algún nivel de automatización dentro de la casa, pudiendo ser desde un simple temporizador para encender y apagar una luz o aparato a una hora determinada, hasta los más complejos sistemas capaces de interactuar con cualquier elemento eléctrico del hogar. La vivienda domótica es por tanto "aquella que integra un conjunto de automatismos en materia de electricidad, electrónica, robótica, informática y telecomunicaciones, con el objetivo de asegurar al usuario un aumento del confort, la seguridad, el ahorro energético, las facilidades de comunicación y las posibilidades de entretenimiento". Se pretende con ello integrar todos los aparatos del hogar a fin de que funcionen de la forma más eficaz posible y con la necesidad de una intervención mínima o inexistente por parte del usuario. Según el Comité Español de la Domótica (CEDOM, s.f.), la domótica contribuye a mejorar la calidad de vida del usuario debido a cuatro aspectos fundamentales que la componen, estos son:

- Ahorro energético:

Gestiona inteligentemente la iluminación, climatización, agua caliente sanitaria, riego, electrodomésticos, entre otros., aprovechando mejor los

recursos naturales. Además, mediante la monitorización de consumos, se obtiene la información necesaria para modificar los hábitos y aumentar el ahorro y la eficiencia.

- **Accesibilidad:**

Facilita el manejo de los elementos del hogar a las personas con discapacidades de la forma que más se ajuste a sus necesidades.

- **Seguridad:**

Brinda la vigilancia automática de personas, animales y bienes, así como de incidencias y averías. Además, posee controles de intrusión, cierre automático de todas las válvulas, simulación dinámica de presencia, fachadas dinámicas, cámaras de vigilancia, alarmas personales, detección de incendios, fugas de gas, inundaciones de agua, fallos del suministro eléctrico, etc., con el objetivo de proporcionar la mayor seguridad a los habitantes del hogar.

- **Comunicación:**

Permite la comunicación constante hacia y desde el exterior, de avisos de anomalías e información del funcionamiento de equipos e instalaciones, todo a través de un teléfono, PC o cualquier dispositivo inteligente. Este aspecto es de gran importancia ya que, posibilita el control y supervisión remoto de la vivienda.

La domótica es, por lo tanto, la integración de varias redes y dispositivos en el hogar que permiten la automatización de actividades cotidianas y el control local o remoto de la vivienda. Para que todos estos dispositivos puedan trabajar de manera conjunta, es necesario que estén conectados a través de una red interna que generalmente se suele conocer por HAN (Home Area Network). Esta red cableada o inalámbrica, se divide en tres tipos de redes según el tipo de dispositivos a interconectar: red de control, red de datos y red multimedia. La red

HAN está conectada a su vez a la conocida pasarela residencial (residencial gateway), la cual debe garantizar la seguridad de las comunicaciones hacia/desde el hogar y debe ser gestionable de forma remota. (Huidobro, y otros, 2007)



Figura 1: Conexión de la red HAN.
Fuente: La Domótica como solución de futuro. Madrid, 2007

Internet.

Es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen constituyan una red lógica única de alcance mundial. Sus orígenes se remontan a 1969, cuando se estableció la primera conexión de computadoras, conocida como ARPANET, entre tres universidades en California (Estados Unidos). (Wikipedia, 2020)

A diferencia de lo que suele pensarse, Internet y la World Wide Web no son sinónimos. La WWW es un sistema de información desarrollado en 1989 por Tim Berners Lee y Robert Cailliau. Este servicio permite el acceso a información que se encuentra enlazada mediante el protocolo HTTP (HyperText Transfer Protocol). (Lapiente, 2018)

Protocolo TCP/IP.

TCP/IP es la identificación del grupo de protocolos de red que hacen posible la transferencia de datos en redes, entre equipos informáticos e internet. Las siglas TCP/IP hacen referencia a este grupo de protocolos:

- **TCP** es el Protocolo de Control de Transmisión que permite establecer una conexión y el intercambio de datos entre dos anfitriones. Este protocolo proporciona un transporte fiable de datos.
- **IP** o protocolo de internet, utiliza direcciones series de cuatro octetos con formato de punto decimal (como por ejemplo 75.4.160.25). Este protocolo lleva los datos a otras máquinas de la red.

El modelo TCP/IP permite un intercambio de datos fiable dentro de una red, definiendo los pasos a seguir desde que se envían los datos (en paquetes) hasta que son recibidos. Para lograrlo utiliza un sistema de capas con jerarquías (se construye una capa a continuación de la anterior) que se comunican únicamente con su capa superior (a la que envía resultados) y su capa inferior (a la que solicita servicios).

Ventajas del modelo TCP/IP.

- TCP/IP ofrece ventajas significativas respecto a otros protocolos de red. Una de esas ventajas es que es capaz de trabajar sobre una extensa gama de hardware y soporta muchos sistemas operativos (es multiplataforma). Internet está repleto de pequeñas redes con sus propios protocolos por lo que el uso de TCP/IP se ha estandarizado y es posible utilizarlo como protocolo de comunicación entre redes privadas intranet y extranet, facilitando una red más homogénea.
- TCP/IP es adecuado tanto para grandes y medianas redes como para redes empresariales o domésticas.

- TCP/IP está diseñado para enrutar y además presenta gran compatibilidad con las herramientas estándar para analizar y monitorizar el funcionamiento de una red.
- Es el protocolo estándar que se utiliza a nivel mundial para conectarse a internet y a los servidores web.

Wifi.

Se conoce como Wifi (derivado de la marca Wi-Fi) a una tecnología de telecomunicaciones que permite la interconexión inalámbrica entre sistemas informáticos y electrónicos, tales como computadores, consolas de videojuego, televisores, teléfonos celulares, reproductores, punteros, etc. Esta tecnología le permite a dichos dispositivos conectarse entre sí para intercambiar datos, o bien conectarse a un punto de acceso de red inalámbrica, pudiendo tener así conexión a Internet.

El Wifi surgió como respuesta a la necesidad de estandarización y compatibilidad en los modelos de conexión inalámbrica de los diversos dispositivos digitales, superando además otras formas no compatibles de conexión como son el Bluetooth, GPRS, UMTS, etc. A diferencia de estos, el Wifi emplea las ondas de radio como vehículo de transmisión de la información. Esta tecnología está diseñada para conectar dispositivos a distancias relativamente cortas (100 metros como máximo), en especial en entornos que ofrezcan mucha interferencia o ruido a la señal, como la producida por la saturación del espectro radioeléctrico debido a multiplicidad de emisiones. Una desventaja de las conexiones de este tipo tiene que ver con la seguridad, dado que cualquier dispositivo que capte la señal es susceptible de tener acceso al punto emisor. Para ello, se lo suele configurar mediante contraseñas y otros mecanismos de seguridad, pero la posibilidad de una violación cibernética queda siempre latente.

Bluetooth.

Bluetooth es una especificación tecnológica para redes inalámbricas que permite la transmisión de voz y datos entre distintos dispositivos mediante una radiofrecuencia segura (2,4 GHz). Esta tecnología, por lo tanto, permite las comunicaciones sin cables ni conectores y la posibilidad de crear redes inalámbricas domésticas para sincronizar y compartir la información que se encuentra almacenada en diversos equipos.

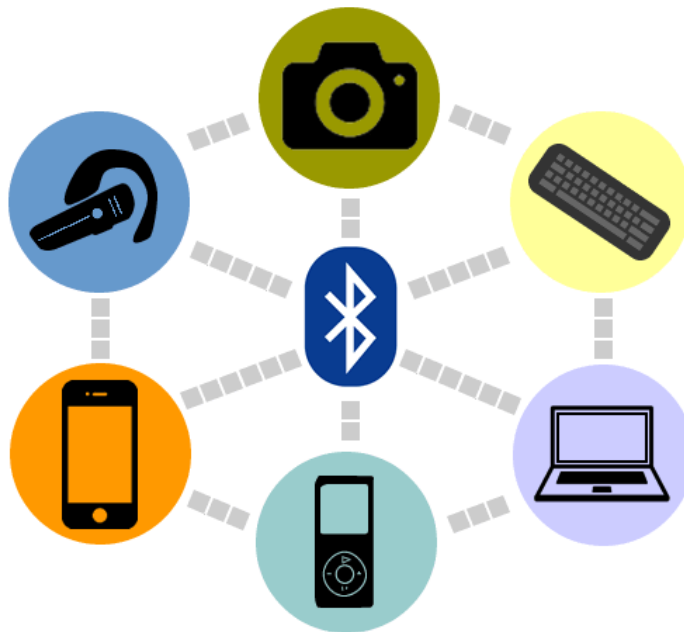


Figura 2: Bluetooth.

Fuente: <https://www.tecnologia-informatica.com/bluetooth/>

En sus inicios, este tipo de tecnología fue incorporada en diversos dispositivos del campo médico, científico e industrial, para posteriormente comenzar a utilizarse en aparatos hogareños, y desde sus comienzos el gran desafío fue lograr que la red de Bluetooth no interfiriera con otros sistemas inalámbricos que también operan por radio frecuencia. Para evitar este inconveniente, a través de Bluetooth se envían señales débiles que no llegan a superar 1 mili vatio, limitando el rango de

acción de los equipos a un máximo de 10 metros de distancia, con el objeto de que la comunicación entre los dispositivos no pueda ser interferida por otras señales. De esta manera, los datos viajan de manera eficaz y segura.

Esp32

El ESP32 es una serie de microcontroladores de bajo costo y consumo de energía, con tecnología Wi-Fi y Bluetooth de modo dual integrada. El ESP32 emplea un microprocesador Tensilica Xtensa LX6 en sus variantes de simple y doble núcleo e incluye interruptores de antena, balun de radiofrecuencia, amplificador de potencia, amplificador receptor de bajo ruido, filtros, y módulos de administración de energía. Fue creado y desarrollado por Espressif Systems, una compañía china basada en Shanghái. Es un sucesor del microcontrolador ESP8266.

ESP32 es capaz de funcionar de manera confiable en entornos industriales, con una temperatura de funcionamiento que varía de -40°C a $+125^{\circ}\text{C}$. Alimentado por circuitos de calibración avanzados. Puede eliminar dinámicamente las imperfecciones de los circuitos externos y adaptarse a los cambios en las condiciones externas. Está diseñado para dispositivos móviles, dispositivos electrónicos portátiles y aplicaciones de IoT, ESP32 logra un consumo de energía ultra bajo con una combinación de varios tipos de software patentado. (SISTEMAS ESPRESSIF, 2019)

Las características del ESP32 incluyen:

- **Procesador:**
 - CPU: microprocesador de 32-bit Xtensa LX6 de doble núcleo (o de un sólo núcleo), controlados independientemente con frecuencia de reloj ajustable, que van desde 80 MHz a 240 MHz.
 - Co-procesador de ultra baja energía (ULP)

- **Memoria:**
 - RAM: 520KB
 - ROM: 448KB

- **Conectividad inalámbrica:**
 - Wi-Fi: 802.11 b/g/n
 - Bluetooth: v4.2 BR/EDR y BLE

- **Interfaces periféricas:**
 - 12-bit SAR ADC de hasta 18 canales
 - 2 × 8-bit DACs
 - 10 × sensores de tacto (sensores capacitivos GPIOs)
 - 4 × SPI
 - 2 × interfaces I²S
 - 2 × interfaces I²C
 - 3 × UART
 - Controlador host SD/SDIO/CE-ATA/MMC/eMMC
 - Controlador esclavo SDIO/SPI
 - Interfaz Ethernet MAC con DMA dedicado y soporte para el protocolo IEEE 1588 Precisión Time Protocol
 - Bus CAN 2.0
 - Controlador remoto infrarrojo (TX/RX, hasta 8 canales)
 - LED PWM (hasta 16 canales)
 - Sensor de efecto Hall
 - Pre-amplificador analógico de ultra baja potencia

- **Alimentación:** 2.2v a 3.6v

- **Seguridad:**
 - Soporta todas las características de seguridad estándar de IEEE 802.11, incluyendo WFA, WPA/WPA2 y WAPI

- Arranque seguro
- Cifrado flash
- 1024-bit OTP, hasta 768-bit para clientes
- Criptografía acelerada por hardware: AES, SHA-2, RSA, criptografía de curva elíptica (ECC), generador de números aleatorios (RNG)
- **Administración de energía:**
 - Regulador interno de baja caída
 - Dominio de poder individual para RTC
 - Corriente de 5μA en modo de suspensión profundo
 - Despierta por interrupción de GPIO, temporizador, medidas de ADC, interrupción por sensor de tacto capacitivo.
- **Entornos de programación:**
 - IDE de Arduino

Entorno de desarrollo integrado, llamado IDE (sigla en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación.

Es una aplicación multiplataforma que se utiliza para escribir y cargar programas en tableros compatibles con Arduino o tableros de otros proveedores, como es el caso de EspressifSystem (empresa creadora de las placas de desarrollo esp32)
 - MicroPython

Es un entorno de desarrollo basado en el lenguaje MicroPython. Este lenguaje no posee todas las librerías estándar que lleva Python debido a que está especialmente diseñado para trabajar con microcontroladores y sistemas embebidos. La ventaja de usar MicroPython es la facilidad del lenguaje. Se pueden utilizar distintos IDE como uPyCraftPyCharm o EsPy.

- Mongoose OS

Es un IDE que permite trabajar con el sistema operativo de Mongoose OS, el cual es un framework de desarrollo para aplicaciones IoT. Es compatible con microcontroladores de bajo consumo, tal como el esp32; soporta como lenguaje de programación JavaScript y es código abierto. Se accede al entorno a través de su página web.

- LUA- NodeMCU

Basado en LUA-RTOS, un sistema operativo de tiempo real basado en el lenguaje LUA, que ha sido diseñado para sistemas embebidos con requerimientos mínimos de flash y RAM. Puede ser programado de dos maneras, usando el lenguaje LUA directamente, o usando programación en bloques, que posteriormente serán traducidos a LUA. Puede programarse desde entornos como WhiteCat IDE que se accede a través del navegador web y permite la programación en bloques o en código, o desde el entorno Eclipse.

- Zerynth

Es una implementación de software del lenguaje de programación Python para programar microcontroladores. Se dirige a plataformas de microcontroladores de 32 bits y está diseñado para mezclar Python con código C. Proporciona un IDE en el que se desarrollan las aplicaciones y posee una APP para dispositivos móviles que permite controlar dispositivos IoT remotamente.

- ESP- IDF (Espressif IoT Development Framework))

Entorno de desarrollo producido por EspressifSystems. Posee el mayor número de recursos al momento de desarrollar aplicaciones para el esp32, ya que cuenta con el soporte del fabricante; sin embargo, su uso no es recomendable debido a la dificultad con la terminal de programación, la lentitud de compilación y el posterior flash del código en el dispositivo. Este entorno está basado en el lenguaje C/C++.

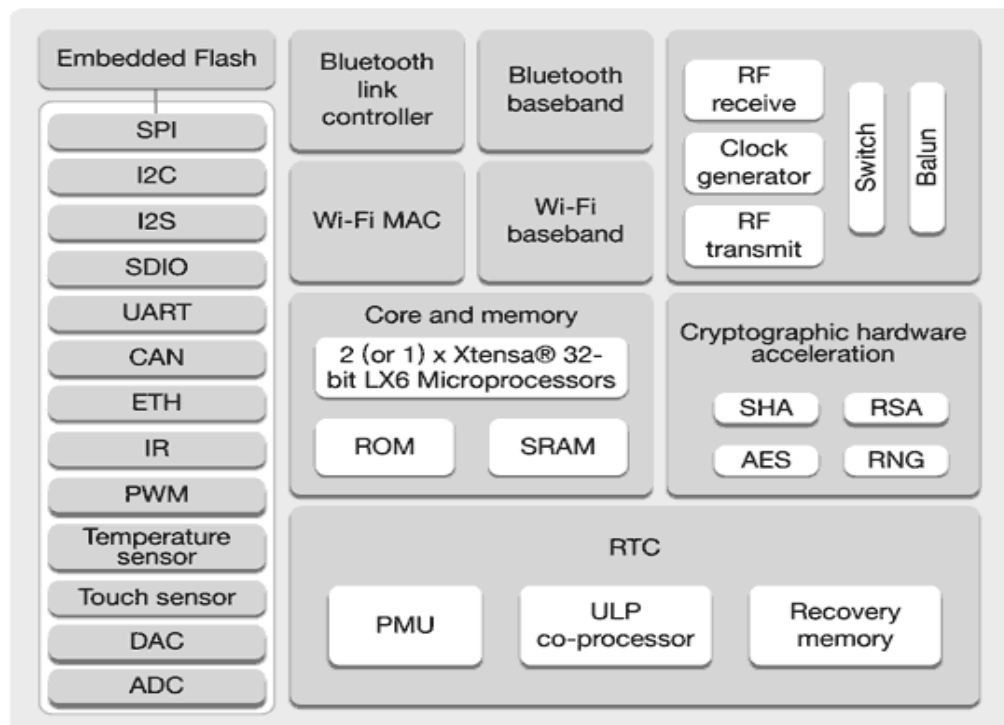


Figura 3: Diagrama de bloques Esp32.
Fuente: <https://www.luisllamas.es/esp32/>

Bluetooth Low Energy ESP32

El ESP32 tiene (2) formas de utilizar el Bluetooth, una es la conexión al Bluetooth clásico y otra, el uso de Bluetooth Low Energy (BLE).

BLE es una variante de Bluetooth que ahorra energía. La aplicación principal de BLE es la transmisión a corta distancia de pequeñas cantidades de datos (ancho de banda bajo). A diferencia de Bluetooth que está siempre encendido, BLE permanece en modo de suspensión constantemente, excepto cuando se inicia una conexión. Esto hace que consuma muy poca energía. BLE consume aproximadamente 100 veces menos energía que Bluetooth (según el caso de uso).

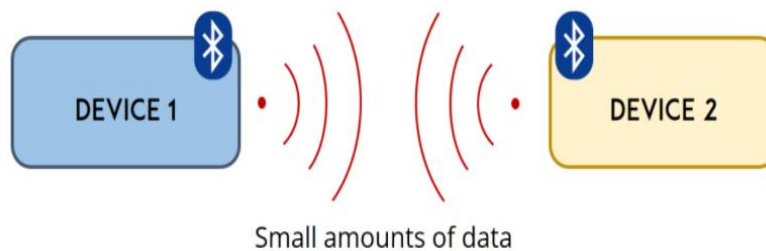


Figura 4. Transmisión BLE.

Fuente: <https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>

Debido a sus propiedades, BLE es ideal para aplicaciones que transmiten datos de forma periódica pero no constante.

Con Bluetooth Low Energy, existen dos tipos de dispositivos: el servidor y el cliente. El ESP32 puede actuar como cliente o como servidor.

El servidor anuncia su existencia, por lo que puede ser encontrado por otros dispositivos y contiene los datos que el cliente puede leer. El cliente escanea los dispositivos cercanos y, cuando encuentra el servidor que está buscando, establece una conexión y escucha los datos entrantes. A esto se le llama comunicación punto a punto.

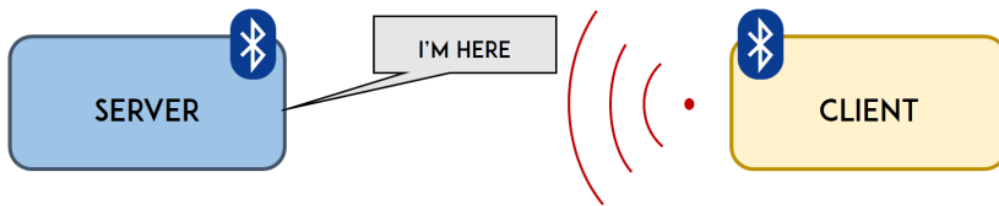


Figura 5. Servidor y cliente BLE
Fuente: <https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>

Para leer y escribir datos con BLE es necesario comprender GATT. GATT significa atributos genéricos y define una estructura de datos jerárquicos que está expuesta a los dispositivos BLE conectados.

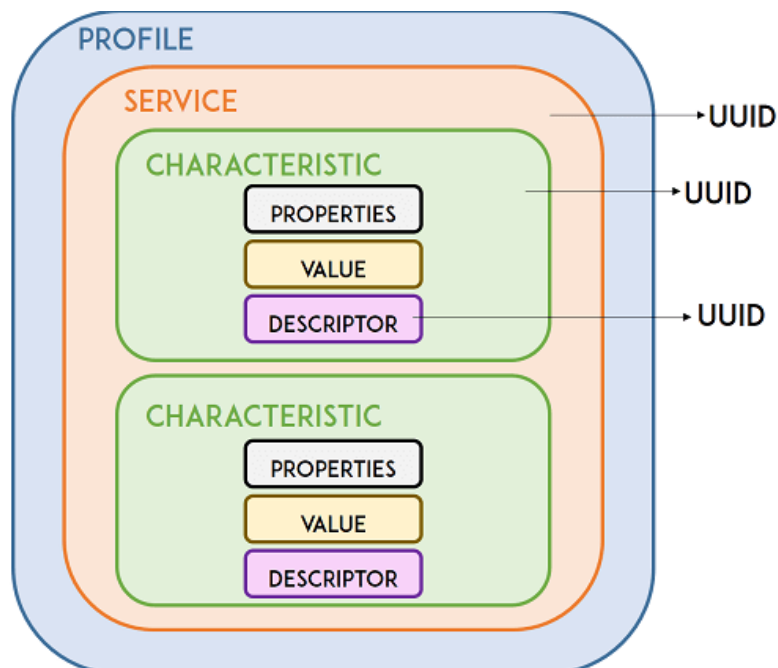


Figura 6. GATT ESP32
Fuente: <https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>

El nivel superior de la jerarquía es un perfil que se compone de uno o más servicios. Cada servicio contiene al menos una característica. Un servicio es una recopilación de información (sensores, por ejemplo); y la característica es una propiedad de un servicio y es donde los datos reales están contenidos en la

jerarquía “valor”. El valor de la característica, puede ir seguido de descriptores, que amplían los metadatos contenidos en la declaración de la característica. Por otro lado, las propiedades describen como se puede interactuar con el valor de la característica (transmitir, leer, escribir, indicar, notificar, entre otros).

Sensor de temperatura DTH11

El DHT11 es un sensor de temperatura y humedad digital de bajo costo. Utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos (no hay pines de entrada analógica). Es bastante simple de usar, pero requiere sincronización cuidadosa para tomar datos.

Este sensor proporciona una salida de datos digital. Entre sus ventajas podemos mencionar el despliegue de datos digitales. Esto supone una gran ventaja frente a los sensores del tipo análogo, como el LM35 por ejemplo, en los cuales las fluctuaciones en el voltaje alteran la lectura de datos. (WP, 2020)

Especificaciones:

- Alimentación: $3V_{dc} \leq V_{cc} \leq 5V_{dc}$
- Rango de medición de temperatura: 0 a 50 °C
- Precisión de medición de temperatura: ± 2.0 °C .
- Resolución de Temperatura: 0.1°C
- Rango de medición de humedad: 20% a 90% RH.
- Precisión de medición de humedad: 4% RH.
- Resolución Humedad: 1% RH
- Tiempo de sensado: 1 seg.

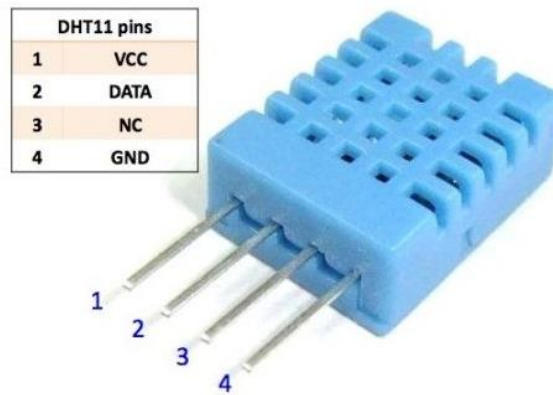


Figura 7. Sensor DTH11.

Fuente: <https://naylampmechatronics.com/sensores-temperatura-y-humedad/58-sensor-de-temperatura-y-humedad-relativa-dht22-am2302.html>

Este sensor no tiene mayor conexión que colocar los cables correspondientes en cada pin. No necesita resistencia de protección mientras su voltaje de entrada esté entre 3v y 5v.

Sensor de movimiento PIR

Los detectores PIR (Passive Infrared) o Pasivo Infrarrojo, reaccionan sólo ante determinadas fuentes de energía tales como el calor del cuerpo humano o animales. Básicamente reciben la variación de las radiaciones infrarrojas del medio ambiente que cubre. Es llamado pasivo debido a que no emite radiaciones, sino que las recibe. Estos captan la presencia detectando la diferencia entre el calor emitido por el cuerpo humano y el espacio alrededor.

Principio de funcionamiento

La radiación infrarroja:

Todos los seres vivos e incluso los objetos, emiten radiación electromagnética infrarroja, debido a la temperatura a la que se encuentran. A mayor temperatura, la radiación aumenta. Esta característica ha dado lugar al diseño de sensores de infrarrojo pasivos, en una longitud de onda alrededor de los

9.4 micrones, los cuales permiten la detección de movimiento, típicamente de seres humanos o animales. Estos sensores son conocidos como PIR, y toman su nombre de 'Pyroelectric Infrared' o 'Passive Infrared'.

El lente de Fresnel:

Es un encapsulado semiesférico hecho de polietileno de alta densidad cuyo objetivo es permitir el paso de la radiación infrarroja en el rango de los 8 y 14 micrones. El lente detecta radiación en un ángulo con apertura de 110° y, adicionalmente, concentra la energía en la superficie de detección del sensor PIR, permitiendo una mayor sensibilidad del dispositivo. Permite una alta precisión a la hora de detectar y simultáneamente protege al detector de falsas alarmas, resultantes de la exposición a la luz solar y artificial.

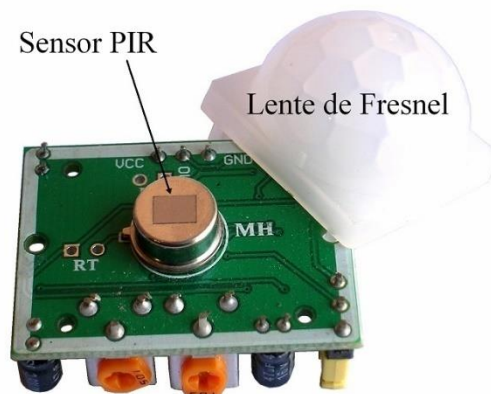


Figura 8: Sensor PIR y lente de Fresnel.

Fuente: <https://www.puntoflotante.net/MODULO-SENSOR-PASIVO-INFRARROJO-PIR-HC-SR501.htm>

En los sensores de movimiento, el sensor PIR consta en realidad de 2 elementos detectores separados, siendo la señal diferencial entre ambos la que permite activar la alarma de movimiento. En el caso del HC-SR501, la señal generada por el sensor ingresa al circuito integrado BISS0001, el cual contiene amplificadores operacionales e interfaces electrónicas adicionales.

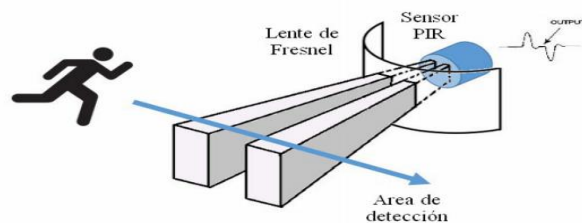


Figura 9: Detección de movimiento Sensor PIR
Fuente: Punto Flotante S.A. 2017

Especificaciones:

- Usa el PIR LHI778 y el controlador BISS0001
- Voltaje de alimentación: de 5 a 12 VDC
- Consumo promedio: <1 Ampere
- Rango de distancia de 3 a 7 metros ajustable.
- Angulo de detección: cono de 110°
- Jumper para configurar la salida de alarma en modo mono-disparo ó disparo repetitivo ('retriggerable')
- Salida de alarma activa Vo con nivel alto de 3.3 volts y 5 ma source.
- Tiempo de inicialización: después de alimentar el módulo HC-SR05, debe transcurrir 1 minuto antes de que inicie su operación normal. Durante ese tiempo, es posible que el módulo active 2 ó 3 veces su salida.
- Tiempo de salida inactiva: cada vez que la salida pase de activa a inactiva, permanecerá en ese estado los siguientes 3 segundos. Cualquier evento que ocurra durante ese lapso es ignorado.
- Temperatura de operación: -15° a +70° C.
- Dimensiones: 3.2 x 2.4 x 1.8 cm.

Ajustes y configuración del sensor:

Posición del jumper:

El usuario puede trabajar en 2 modos de operación:

- 1 solo disparo: En este modo, cuando ocurre una detección de movimiento (el cual llamaremos 'evento'), la salida del sensor se activa. Para efectos de ejemplo, supongamos que el tiempo de activación es de 60 segundos. Si durante esos 60 segundos ocurre un segundo evento, éste no será considerado.

- Disparos repetitivos: En este modo, cada evento detectado genera un nuevo tiempo de activación. Volviendo al ejemplo de tiempo de 60 segundos. Cuando ocurre el primer evento, la salida se activa. Si transcurridos 30 segundos ocurre un segundo evento, entonces se sumarán 60 segundos al tiempo transcurrido, dando un total de 90 segundos continuos con la salida activa. Y así, cada evento adicional, sumará un tiempo de 60 segundos de activación al tiempo ya transcurrido.

En cualquier caso, si la salida regresa a su estado inactivo, habrá un lapso de 3 segundos durante los cuales los nuevos eventos no serán considerados. Pasados esos 3 segundos, el dispositivo regresa a su funcionamiento normal. La conexión del sensor PIR es sencilla y no requiere de elementos adicionales. Con un led, bocina o notificación a través de la app móvil, los usuarios sabrán si hay presencia de intrusos.

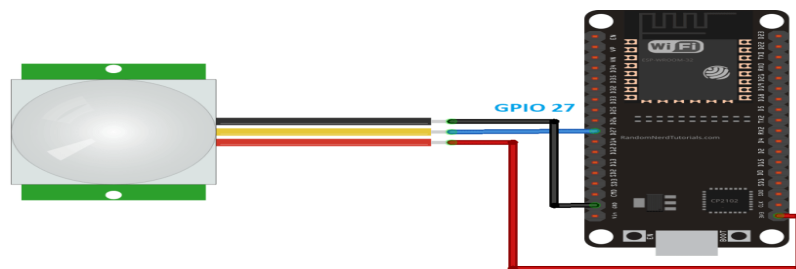


Figura 10. Conexión sensor PIR

Fuente: <https://randomnerdtutorials.com/telegram-esp32-motion-detection-arduino/>

Arduino IDE

Arduino es una comunidad tecnológica y compañía dedicada a la producción de hardware libre. En esta empresa lo que hacen es diseñar y manufacturar diferentes tipos de placas de desarrollo de hardware y software, que son los que integran el microcontrolador.

El entorno de desarrollo integrado (IDE) de Arduino es una aplicación multiplataforma (para Windows, macOS, Linux) que está escrita en el lenguaje de programación Java. Se utiliza para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, se puede usar con placas de desarrollo de otros proveedores.

El código fuente para el IDE se publica bajo la Licencia Pública General de GNU, versión 2. El IDE de Arduino admite los lenguajes C y C ++ utilizando reglas especiales de estructuración de códigos. Además, suministra una biblioteca de software del proyecto Wiring, que proporciona muchos procedimientos comunes de E/S. El código escrito por el usuario solo requiere dos funciones básicas, **Void setup()** (para iniciar el boceto) y **Void loop** (ciclo principal del programa), que se compilan y vinculan con un apéndice de programa *main()* en un ciclo con el GNU toolchain, que también se incluye. El IDE de Arduino emplea el programa *avrdude* para convertir el código ejecutable en un archivo de texto en codificación hexadecimal que se carga en la placa Arduino mediante un programa de carga en el firmware de la placa

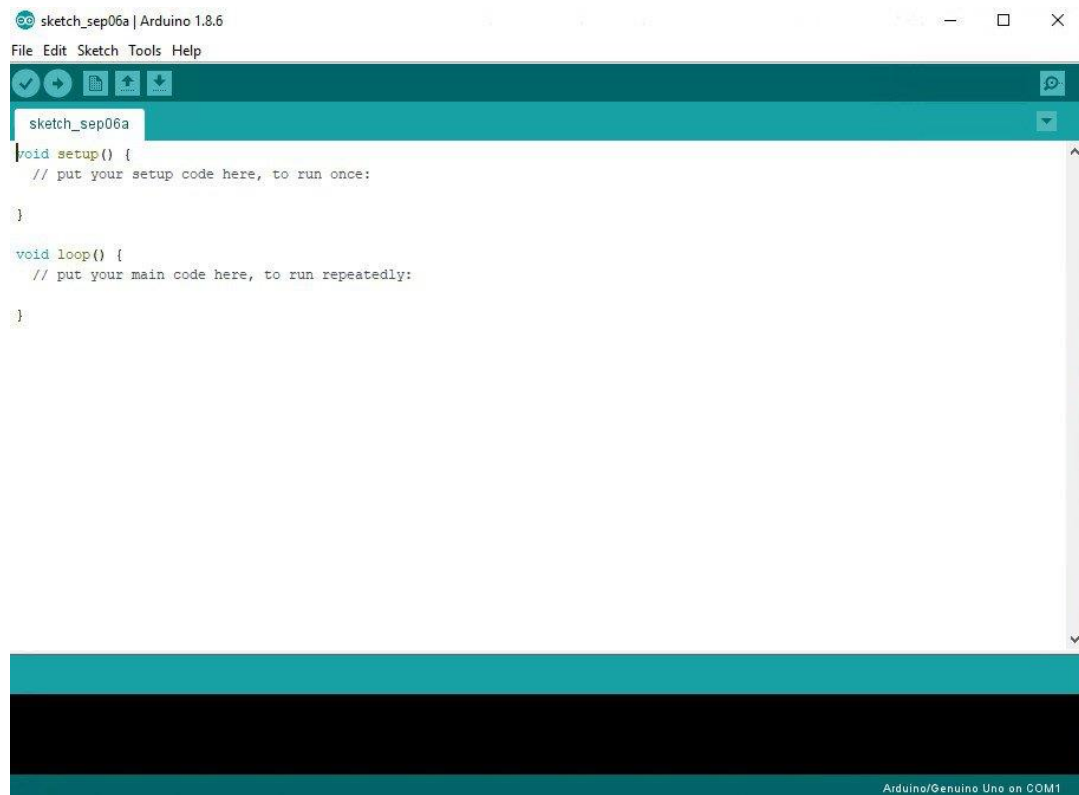


Figura 11: Arduino IDE
Fuente: Elaboración propia.

Flutter.

Es un framework de código abierto desarrollado por Google para crear aplicaciones nativas de forma fácil, rápida y sencilla. Su principal ventaja radica en que genera código 100% nativo para cada plataforma, con lo que el rendimiento y la UX es totalmente idéntico a las aplicaciones nativas tradicionales.

La principal y más importante ventaja de Flutter es que se desarrolla un solo proyecto para todos los sistemas operativos, lo que significa una reducción de costes y tiempo de producción.

Funcionalidades de Flutter

- **Calidad nativa:** Las aplicaciones nativas se desarrollan específicamente para un sistema operativo, Flutter utiliza todas las ventajas de las aplicaciones nativas para conseguir calidad en el resultado final.
- **Experiencia de usuario:** Flutter incluye Material Design de Google y Cupertino de Apple, con lo que la experiencia de usuario es óptima y las interfaces de usuario idénticas a las de las aplicaciones desarrolladas por las propias compañías.
- **Tiempo de carga:** Una de las principales causas de abandono de una aplicación es el tiempo que tarda en cargar, con Flutter se experimentan tiempos de carga por debajo de un segundo en cualquiera de los soportes iOS o Android.
- **Desarrollo ágil y rápido:** Gracias a la característica hot-reload, puedes programar y ver los cambios en tiempo real en tu dispositivo o en los simuladores.

DART.

Dart (originalmente llamado Dash) es un lenguaje de programación de código abierto, desarrollado por Google. Fue revelado en la conferencia goto; en Aarhus, Dinamarca el 10 octubre de 2011. En sus inicios se presentaba como un lenguaje que ofrecía una alternativa a JavaScript, pero más moderno con implementaciones que en su tiempo JavaScript no tenía. Originalmente Dart era usado para el desarrollo en el lado del cliente en un navegador web, sin embargo, hoy se puede usar Dart en Android, iOS y la Web.

Dart se presentó oficialmente con su versión 1.0 en el año 2013 y en fechas recientes se ha vuelto popular debido a que es el lenguaje para desarrollo de

aplicaciones multiplataforma con Flutter, un proyecto también de Google que es un SDK para el desarrollo de aplicaciones tanto para Android como iOS. Este lenguaje se caracteriza por su optimización para desarrollo del lado del cliente, un tipado de datos flexible, ser rápido y tener una sintaxis de desarrollo moderna. (Codigo facilito, s.f.)

Visual Studio Code.

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go, DART, entre otros) y tiempos de ejecución (como .NET y Unity).

En el rol de editor de código fuente, Visual Studio Code permite cambiar la página de códigos en la que se guarda el documento activo, el carácter que identifica el salto de línea (una opción entre LF y CRLF) y el lenguaje de programación del documento activo.

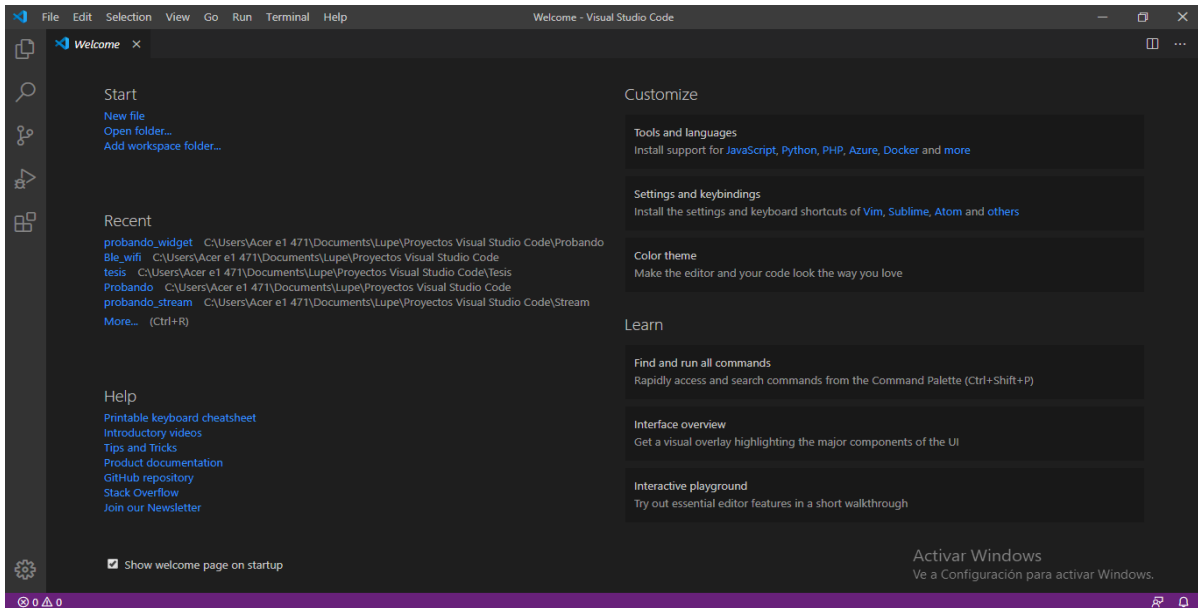


Figura 12: Pantalla de inicio Visual Studio Code.
Fuente: Elaboración propia.

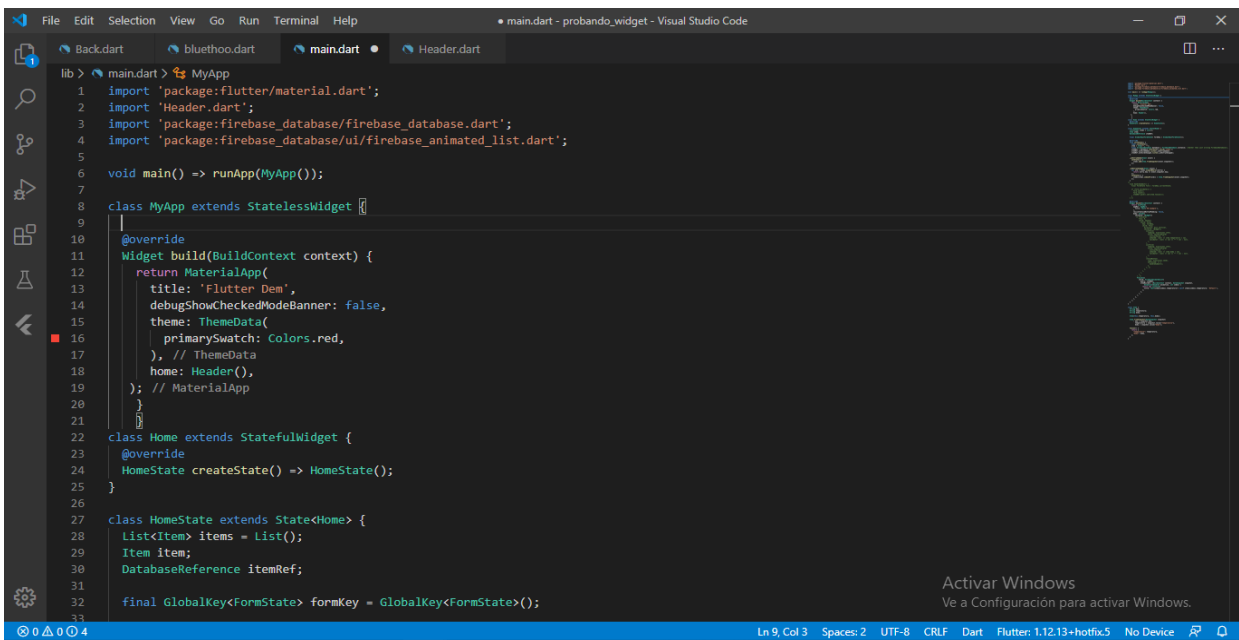


Figura 13: Visualización grafica de un programa en Visual Studio Code.
Fuente: Elaboración propia.

Firestore

Firestore es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles lanzada en 2011 y adquirida por Google en 2014. Es una plataforma ubicada en la nube, integrada con Google Cloud Platform, que usa un conjunto de herramientas para la creación y sincronización de proyectos que serán dotados de alta calidad.

Ventajas de Firestore:

- Sincroniza fácilmente los datos de los proyectos sin tener que administrar conexiones o escribir lógica de sincronización compleja.
- Usa un conjunto de herramientas multiplataforma: se integra fácilmente para plataformas web como en aplicaciones móviles. Es compatible con grandes plataformas, como IOS, Android, aplicaciones web, Unity y C++.
- Usa la infraestructura de Google y escala automáticamente para cualquier tipo de aplicación, desde las más pequeñas hasta las más potentes.
- Crea proyectos sin necesidad de un servidor: Las herramientas se incluyen en los SDK para los dispositivos móviles y web, por lo que no es necesario la creación de un servidor para el proyecto.

Servicios

- Firestore Analytics: Es una aplicación gratuita que proporciona una visión profunda sobre el uso de la aplicación por parte de los usuarios
- Desarrollo: Firestore permite la creación de mejores aplicaciones, minimizando el tiempo de optimización y desarrollo mediante diferentes funciones, entre las que destacan la detección de errores y el testeo, lo cual supone poder dar un salto de calidad a la aplicación. Poder almacenar toda la información en la nube y configurarla de manera distribuida.

- **Firestore Database:** Antiguamente conocido como Google Cloud Firestore (GCM), Firestore Database (FDB) es una plataforma para almacenar y recuperar datos para Android, iOS, y aplicaciones web que actualmente puede ser usada de forma gratuita.
- **Firebase Auth:** Es un servicio que puede autenticar los usuarios utilizando únicamente código del lado del cliente. Incluye la autenticación mediante proveedores de inicio de sesión como *Facebook*, *GitHub*, *Twitter*, *Google*, *Yahoo* y *Microsoft*, así como los métodos clásicos de inicio de sesión mediante correo electrónico y contraseña. Además, incluye un sistema de administración del usuario por el cual los desarrolladores pueden habilitar la autenticación de usuarios con correo electrónico y contraseña que se almacenarán en Firebase. Este servicio busca facilitar la creación de sistemas de autenticación, a la vez que mejora la incorporación, acceso y seguridad para los usuarios. Gracias a esto, el cliente no tiene que preocuparse por desarrollar métodos de autenticación clásicos, ya que Firebase le aporta de manera sencilla, eficaz y segura métodos para gestionar sus usuarios. También aporta muchas funcionalidades extra, como la recuperación y verificación de cuentas, tanto por correo electrónico como por SMS, y cuotas de registro para los usuarios, todo esto gestionado mediante los servidores de la plataforma.
- **Realtime Database:** Firebase proporciona una base de datos en tiempo real, back-end y organizada en forma de árbol JSON. El servicio proporciona a los desarrolladores de aplicaciones una API que permite que la información de las aplicaciones sea sincronizada y almacenada en la nube de Firebase. La compañía habilita integración con aplicaciones Android, iOS, JavaScript, Java, Objective-C, Swift y Node.js. La base de datos es también accesible a través de una REST API e integración para varios sistemas de Javascript como AngularJS, React,

Ember.js y Backbone.js. La REST API utiliza el protocolo SSE (del inglés Server-Sent Events), el cual es una API para crear conexiones de HTTP para recibir notificaciones push de un servidor. La sincronización en tiempo real de esta base de datos permite que los usuarios accedan a la información de sus datos desde cualquier dispositivo en tiempo real, compartiendo una instancia de Realtime Database, y cada vez que un usuario realice una modificación en esta, se almacena dicha información en la nube y se notifica simultáneamente al resto de dispositivos. Una funcionalidad interesante de esta base de datos, es que, si un usuario realiza cambios y pierde a la vez su conexión a Internet, el SDK de la plataforma usa una caché local en el dispositivo donde guarda estos cambios; y una vez que vuelva a tener conexión, automáticamente se sincronizan los datos locales.

- **Firestore Storage:** Firestore Storage proporciona cargas y descargas seguras de archivos para aplicaciones Firestore, sin importar la calidad de la red. El desarrollador lo puede utilizar para almacenar imágenes, audio, vídeo, o cualquier otro contenido generado por el usuario. Firestore Storage se basa en el almacenamiento de Google Cloud Storage.
- **Firestore Cloud Firestore:** Es un servicio de almacenamiento de datos derivado de Google Cloud Platform, adaptado a la plataforma de Firestore. Al igual que Realtime Database, es una base de datos NoSQL, aunque presenta diversas diferencias. Se organiza en forma de documentos agrupados en colecciones, y en ellos se pueden incluir tanto campos de diversos tipos (cadenas de texto, números, puntos geográficos, referencias a la propia base de datos, arrays, booleanos, marcas de tiempo, e incluso objetos propios) como otras subcolecciones. Entre sus limitaciones más destacadas encontramos la de no soportar las búsquedas de texto tipo "LIKE", eso es, buscar por subcadenas del texto almacenado, y la de no

poder filtrar las búsquedas con condiciones que impliquen más de un campo, si no es por búsquedas por el texto exacto.

(Wikipedia, 2021)

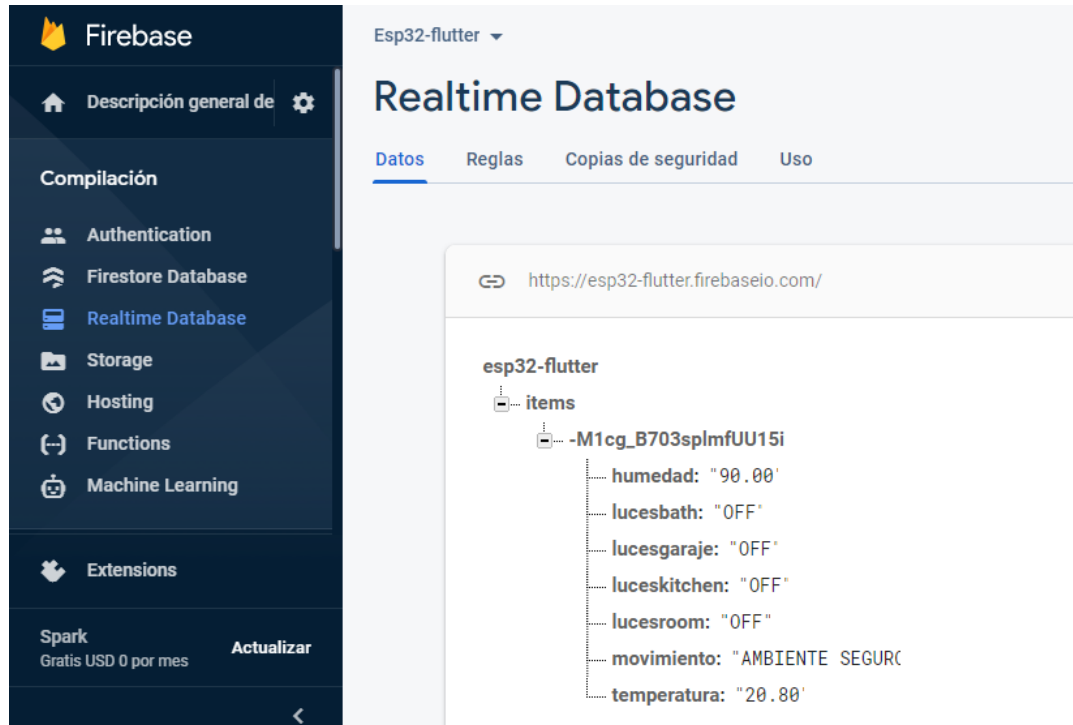


Figura 14. Realtime database Firebase
Fuente: Elaboración propia.

CAPITULO III

MARCO REFERENCIAL

Tipo de investigación

La investigación proyectiva consiste en la elaboración de una propuesta, un plan, programa o modelo, como solución a un problema o necesidad de tipo práctico, ya sea de un grupo social, institución o de una región geográfica; esto, orientado a un área particular del conocimiento a partir de un diagnóstico preciso de las necesidades del momento, los procesos explicativos y las tendencias futuras. Se realiza cuando hay situaciones que se desean modificar y mejorar, debido a que hay potencialidades que no se están aprovechando y/o problemas por resolver. Por ende, el investigador diagnostica el problema (evento a modificar), explica a qué se debe (proceso causal) y desarrolla la propuesta con base en esa información. Este tipo de investigación involucra la creación, diseño y elaboración de planes o proyectos; sin embargo, no todo proyecto es investigación proyectiva. Para que un proyecto se considere investigación proyectiva, la propuesta debe estar fundamentada en un proceso sistemático de búsqueda e indagación que requiere de descripción, análisis, comparación, explicación y predicción. (Barrera, Blog Investigacion Holistica, 2008)

Son ejemplos de investigación proyectiva, los proyectos de arquitectura e ingeniería, creación de programas de intervención social, diseño de programas de estudio, entre otros..., siempre que estén sustentados en un proceso de investigación. (Barrera, Guia para la comprension holistica de la ciencia, 2010). Según esta descripción, se puede considerar el presente trabajo de grado como

una investigación de tipo proyectiva, debido a que contempla el total desarrollo (diseño y construcción) de un sistema domótico controlado mediante el micro-controlador esp32; esto, basado en una extensa documentación previa.

Diseño de investigación

El diseño de investigación se refiere a la estrategia que adopta el investigador para responder al problema, dificultad o inconveniente planteado en el estudio. (G.Arias, 2016)

Según el autor (Barrera, Guia para la comprension holistica de la ciencia, 2010) , se relaciona con los procesos específicos para recoger los datos (fuentes, tiempo y cantidad de eventos de estudio). Se clasifica en: documental, de campo y experimental.

La investigación documental es un proceso basado en la búsqueda, recuperación, análisis, crítica e interpretación de datos secundarios. La investigación de campo consiste en la recolección de datos directamente de los sujetos investigados, sin manipular o controlar variable alguna, es decir, el investigador no altera las condiciones existentes; y por último se entiende por investigación experimental, al proceso que consiste en someter a un objeto o un grupo de individuos, a determinadas condiciones para observar los efectos que se producen. (G.Arias, 2016)

La investigación experimental se caracteriza por la manipulación y control de las variables o condiciones, que ejerce el investigador durante el experimento. En función a esto, se puede decir que el presente trabajo de grado es de tipo experimental, ya que se realizara un sistema de control domótico en el que habrá observación, manipulación y control de cada una de las variables involucradas con el fin de obtener un resultado satisfactorio.

Unidades de análisis

Según el autor (Heredia, 2012) la unidad de análisis se refiere a la unidad representativa de estudio, concebida por el conjunto de escenarios o ámbitos generalizados donde suceden las ocurrencias del fenómeno objeto de la investigación. Implica procesos, sistemas, procedimientos y actividades; es un conjunto de elementos cuyo análisis permitirá la obtención de datos primarios de forma lógica y estructurada. Son las entidades que poseen el objeto de estudio. Estas unidades están conformadas por población y muestra, generalmente.

La población es un conjunto de individuos de la misma clase, limitada por el estudio. Se define como la totalidad del fenómeno a estudiar donde las unidades de población poseen una característica común la cual se estudia y da origen a los datos de investigación. (Mario, 1997). De lo explicado anteriormente y basándose en el propósito de esta investigación, se determina que la población viene dada por todos los softwares de programación para la placa de desarrollo esp32, software de simulación de solicitudes HTTP, editores de texto y framework utilizado para la creación de la aplicación móvil, servidores web, entre otros., que son necesarios para llevar a cabo el sistema de control domótico.

En algunos casos la población es tan grande e inaccesible que no se puede estudiar toda, entonces el investigador tendrá la posibilidad de seleccionar una muestra (Barrera, El proyecto de Investigacion Hurtado, 2015). Según (G.Arias, 2016) la muestra es un subconjunto representativo y finito que se extrae de la población accesible. De acuerdo a esto, se tomará como muestra en el presente trabajo de investigación, el entorno de desarrollo integrado de Arduino, más conocido como Arduino IDE, a través del cual se realizará la compilación y carga de los proyectos para la programación del micro-controlador utilizado, el servidor Web Firebase, en el cual se almacenarán todos los datos provenientes de los sensores conectados al Esp32.

Eventos o variables

Las variables independientes que se han de estudiar son la lectura de los distintos sensores que se tienen para la realización del sistema domotico, estas varían de acuerdo al entorno y las circunstancias en donde estén presente los sensores y debido a ellas y a las necesidades del usuario, se controlaran las variables dependientes, las cuales abarcan los actuadores que se tengan en el hogar.

Técnicas e instrumentos de recolección de datos

Las técnicas de recolección de datos se entienden como las distintas formas o maneras de obtener la información. (G.Arias, 2016)

Libros o textos especializados

Proporcionan una amplia documentación del tema a estudiar. Para esta investigación se consultó textos físicos y en formato digital.

Internet

Es imprescindible la utilización de este medio para el trabajo desarrollado, debido a que existe una gran cantidad de documentación escrita en otros idiomas (inglés, portugués, entre otros) y gracias a esta herramienta se puede traducir fácilmente para su posterior uso y aplicación.

Trabajos de grados relacionados con el presente proyecto

Presentan información vital para el desarrollo del proyecto, que se puede utilizar con el objetivo de mejorar la aplicación o diseño. En el presente trabajo de investigación se consultó y recopilo documentación e información de tesis (nacionales e internacionales) relacionadas con un sistema domótico.

Tutoriales básicos

Son aquellos que, a través de video, imágenes o documentos, enseñan una forma fácil y sencilla de hacer algún proceso o actividad. Son de suma importancia para la realización del sistema domotico ya que permiten la comprensión de los diferentes instrumentos y programas utilizados para el proyecto, entre ellos el esp32, Flutter, entre otros.

La aplicación de una técnica conduce a la obtención de información, la cual debe ser guardada en un medio material para que los datos puedan ser procesados, analizados e interpretados posteriormente. A dicho soporte se le denomina instrumento.

“Un instrumento de recolección de datos es cualquier recurso, dispositivo o formato (en papel o digital), que se utiliza para obtener, registrar o almacenar información.” (G.Arias, 2016)

Computadora personal

Empleada para la redacción del informe, documentación y manejo de los programas necesarios para llevar a cabo el proyecto, como Arduino IDE, Postman, Python, Visual Studio Code, Flutter, entre otros.

Realtime Database Firebase

Base de datos en tiempo real alojada en el servidor Firebase, que almacena todos los datos provenientes del Esp32, para luego enviarlos a la aplicación móvil diseñada, que se encargara de monitorear y controlar las variables medidas.

Arduino IDE

Es una aplicación multiplataforma que está escrita en el lenguaje de programación de Java. Se utiliza para escribir y cargar programas en tableros compatibles con Arduino o tablero de otros proveedores, como es el caso de EspressifSystem (empresa creadora de las placas de desarrollo esp32)

Visual Studio Code

Es un editor de código fuente desarrollado por Microsoft para Windows , Linux y macOS. En este editor, se instalarán las extensiones necesarias para trabajar con Flutter (Framework escogido para desarrollar aplicaciones móviles o de escritorio)

Flutter

Flutter es un framework de código abierto desarrollado por Google para crear aplicaciones nativas de forma fácil, rápida y sencilla. Su principal ventaja radica en que genera código 100% nativo para cada plataforma. Flutter utiliza como lenguaje de programación DART, el cual también fue desarrollado por Google.

CAPITULO IV

DISEÑO

En este capítulo se hará mención del microcontrolador utilizado, sus características y potencialidades; del hardware empleado como sistema modelo a controlar, de la programación del software necesario para integrar los elementos (hardware y software) que conforman el proyecto y de la creación de la aplicación móvil que permitirá el control y monitoreo de todas las variables a estudiar (temperatura, humedad, luz y movimiento).

Requerimientos del Sistema en general

Se requiere el diseño y desarrollo de un sistema de control domótico, el cual tiene como función automatizar áreas específicas de una casa, tales como: control de luces (On/Off), sistema de seguridad y monitoreo de variables (temperatura, humedad); mediante la placa de desarrollo esp32. El usuario podrá visualizar las variables antes mencionadas y controlar las mismas, a través de una aplicación móvil, que además podrá modificar las credenciales de la red Wifi del esp32 a través de bluetooth.

Requerimientos del hardware

- Se requiere un microcontrolador que posea las características necesarias para llevar a cabo el proyecto, entre ellas se mencionan las siguientes:
 - Wifi.
 - Bluetooth
 - ADC

- Puertos E/S digitales y analógicos.
- El microcontrolador debe ser económico, de fácil manejo y mantenimiento.
- Los sensores a utilizar deben ser de bajo coste, fácil instalación y programación.
- El sistema debe tener la capacidad de ser controlado por uno o más teléfonos móviles.

Requerimientos del Software

- Se requiere diseñar una aplicación móvil, capaz de interactuar con el usuario y que éste pueda monitorear y/o controlar las variables del sistema.
- El sistema debe tener una comunicación continua y estable.
- La comunicación será vía Wifi y Bluetooth.

Descripción del diseño

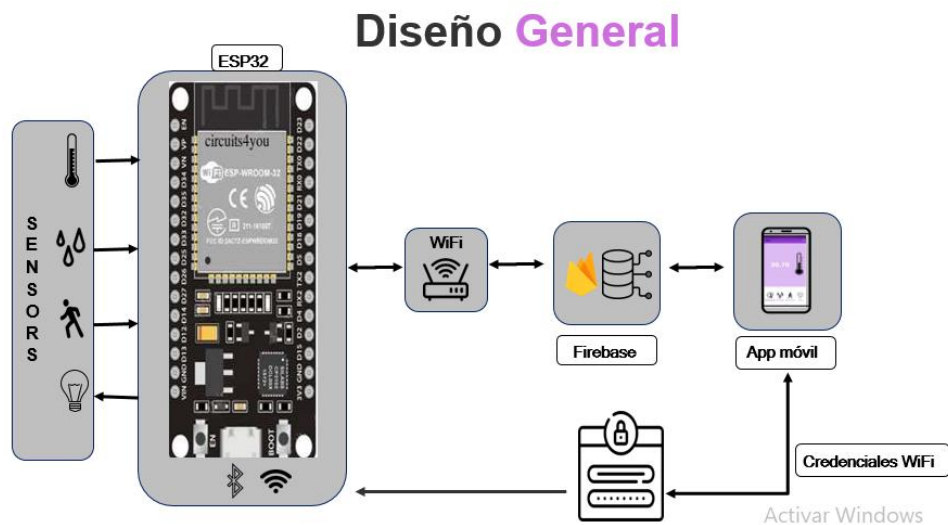


Figura 15. Diseño general del sistema.
Fuente: Elaboración propia.

El sistema tiene como objetivo principal la comunicación constante entre el usuario y el microcontrolador con el fin de poder visualizar los datos de los sensores

utilizados; para ello, el microcontrolador enviará los datos que los sensores proporcionen a través de Wifi a una base de datos alojada en Firebase. Firebase tiene la ventaja de crear diferentes bases de datos según las necesidades que se tengan; para este caso, se ha de utilizar Realtime basedata (Base de datos en tiempo real).

Firebase permite la configuración de credenciales, usuarios y permisos para poder utilizar los datos alojados en el, en una aplicación real.

La visualización y/o monitoreo de datos se hará a través de una aplicación móvil, la cual crearemos con el framework Flutter. Este framework a través de bibliotecas específicas, permite la comunicación con Firebase, por lo que tendremos una lectura de datos en tiempo real. Para la programación, Flutter utiliza como lenguaje DART.

Una vez los datos están contenidos en Firebase, esté los envía a la aplicación y el usuario al ingresar podrá visualizar la variable que guste. Toda la visualización, modificaciones o cambios se harán con la aplicación móvil; por ello, el cambio de las credenciales Wifi del ESP32, también se realizará con la app. Este cambio se efectúa sí y solo sí, la opción Bluetooth esta activada en el microcontrolador, una vez que la app detecte el bluetooth del ESP32, mostrará una pantalla para configurar las credenciales WIFI. La red y contraseña escritos en el ESP32 se guardan en la EEPROM, por lo que el microcontrolador no perderá conexión con la red Wifi. La modificación de las credenciales WiFi vía Bluetooth tiene un rol fundamental en el proyecto, ya que, si se pierde conexión con la red local que está conectada al ESP32, fácilmente podrían cambiarse las credenciales y conectarse a otra red.

Propuesta de hardware

Placa de desarrollo Esp32.

El esp32, original de la empresa EspressifSystems, será el encargado de interactuar con los diferentes sensores que se tendrán en el sistema, recibiendo

de estos la data y enviándola a un servidor web externo (Firebase). Esta placa integra tecnologías Wifi y Bluetooth, posee 38 pines que pueden ser utilizados según los requerimientos del sistema, y además por su bajo coste, es el ideal para el desarrollo de este proyecto.



Figura 16. Placa de desarrollo esp32

Fuente: <https://www.jaycar.com.au/esp32-main-board-with-wifi-and-bluetooth-communication/p/XC3800>

La placa Esp32 se programará con la IDE de Arduino, ya que es el entorno de desarrollo más amigable y cómodo para programar.

Los sensores que se utilizarán serán: sensor de temperatura y humedad y sensor de movimiento.

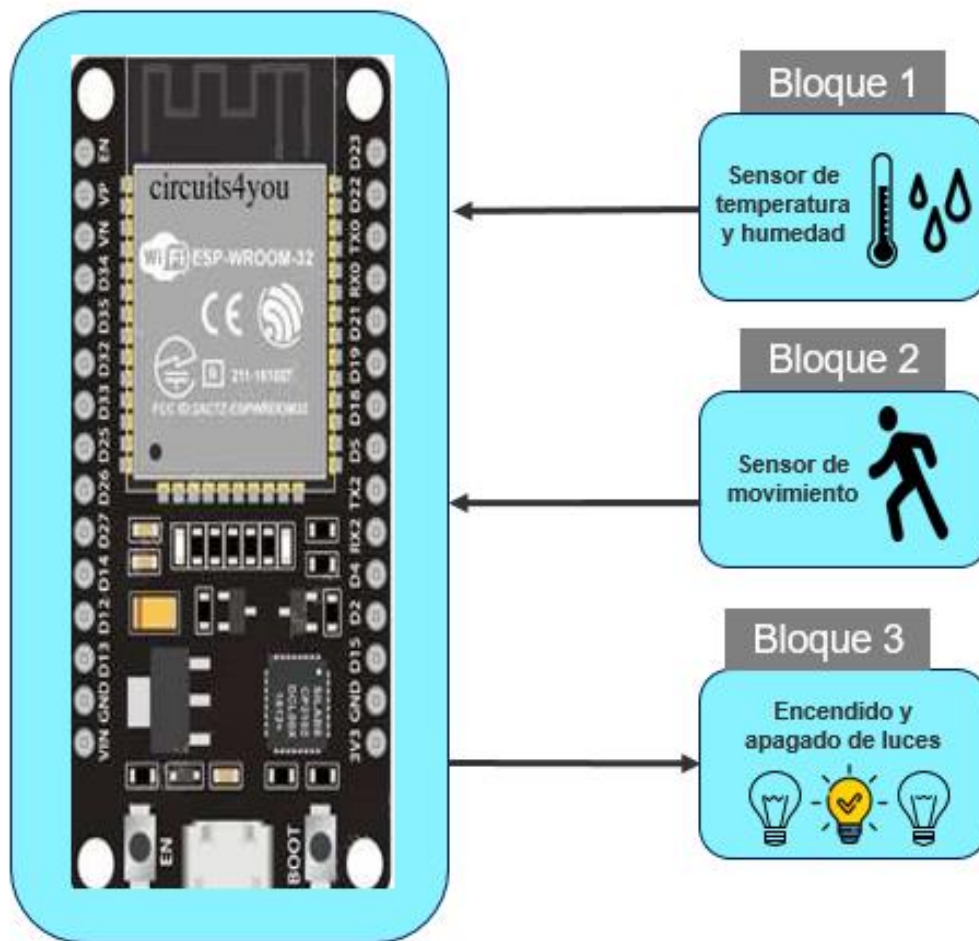


Figura 17. Esquema del hardware
Fuente: Elaboración propia

En la figura 17, se visualiza la interacción entre los sensores y el ESP32. Para la explicación se dividió el esquema en bloques.

Bloque 1:

En este bloque se utilizó el sensor de temperatura y humedad DTH11. Este, es un sensor digital que envía la data al ESP32 por medio de una señal digital. El ESP32 tiene 34 pines que pueden ser utilizados como pines digitales.

El sensor DTH11 utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante. En cuanto a los datos, se transmiten de manera digital al ESP32. Aunque el sensor en sí es analógico, incluye un sistema para realizar la conversión y se puede conectar directamente a una entrada digital del microcontrolador.

La señal analógica, que es una variación del voltaje del sensor, es pasada a formato digital para ser enviada al microcontrolador del ESP32. Se trasmite en una trama de 40-bits que corresponden a la información de humedad y temperatura captada por el DHT11. Los primeros dos grupos de 8-bit son para la humedad, es decir, los 16 bits más significativos de esta trama. Luego, los otros 2 grupos de 8-bits restantes para la temperatura. Es decir, tiene dos bytes para humedad y dos bytes para temperatura. Los últimos 8 bits son de paridad y se utilizan para evitar errores.

Bloque 2:

El sensor utilizado para detectar movimiento, es el PIR HCSR501. Este posee un pin de salida digital, el cual envía la señal al ESP32. Cuando detecte algún movimiento su estado bajo, pasará a alto.

Bloque 3:

El encendido y apagado de luces se realiza a través de software; es decir, el usuario a través de la aplicación móvil enviará la señal. Esta se guardará en la base de datos en Firebase y a su vez se enviará al ESP32 que encenderá la luz correspondiente.

Propuesta de Software

Contiene la explicación sobre la programación para la configuración de credenciales wifi vía Bluetooth, programación del hardware y la creación de la aplicación móvil. Por ello, se dividirá la estructura general en varias etapas

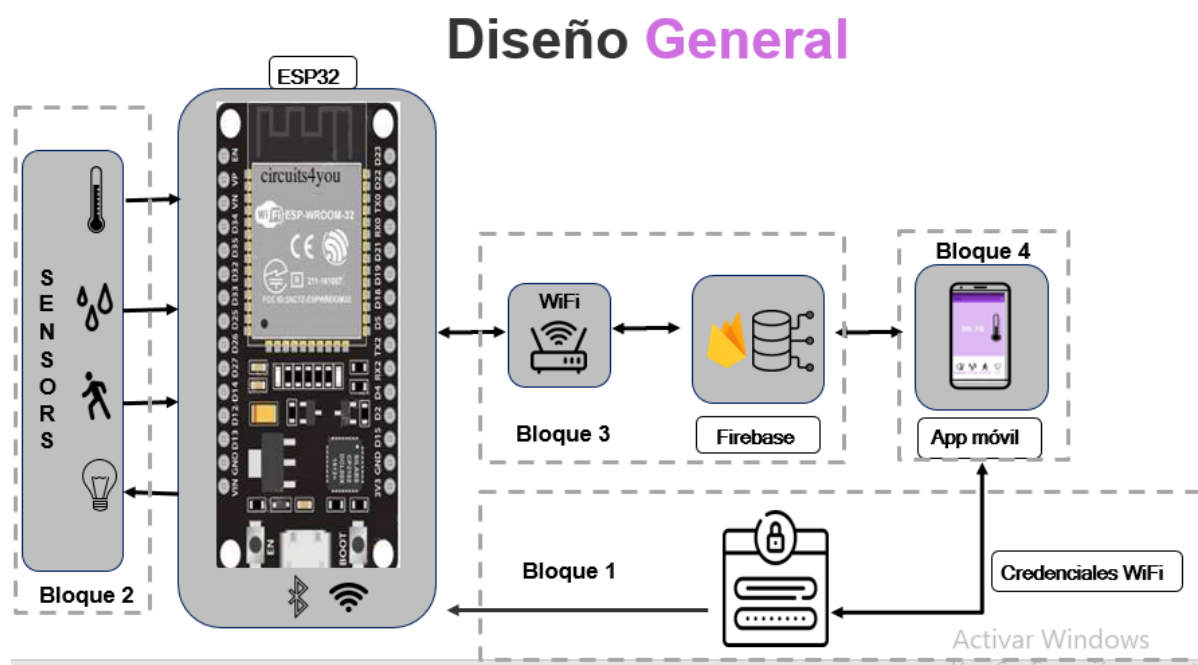


Figura 18. Etapas para la programación del software.
Fuente: Elaboración propia.

Bloque 1: Configuración credenciales WiFi vía Bluetooth

Bloque 2: Configuración de sensores y elementos en el ESP32

Bloque 3: Comunicación ESP32 con Firebase.

Bloque 4: Aplicación móvil.

Configuración de credenciales WiFi vía Bluetooth

Al iniciar el sistema por primera vez, el ESP32 no estará conectado a ninguna red WiFi. Entonces, se coloca el ESP32 en modo bluetooth para colocar las credenciales WiFi, luego se cambia de modo y se podrá enviar los datos vía WiFi con la red que antes le proporcionamos.

A continuación, se muestra el diagrama de flujo con los esquemas del algoritmo diseñado para escoger el modo de funcionamiento del ESP32. Esto se realiza para una mayor comprensión del funcionamiento del código en general.

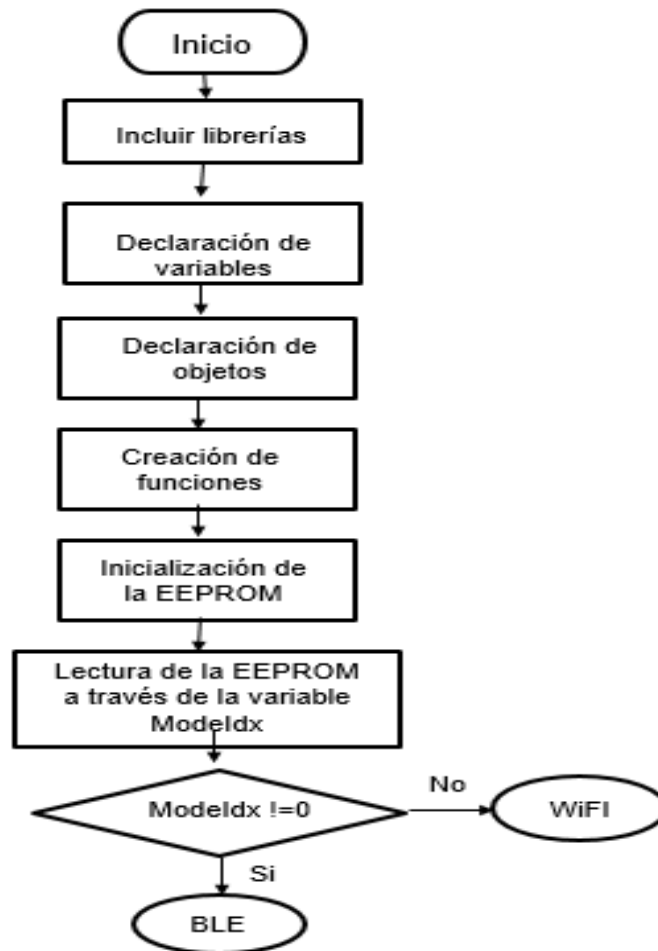


Figura 19. Diagrama de flujo conexión inicial Wifi – Bluetooth
Fuente: Elaboración propia.

El diagrama de flujo de la figura 19, contiene la estructura del inicio del programa. Es importante resaltar que el ESP32 podrá tener (2) modos de funcionamiento: WiFi o Bluetooth, dependiendo de las necesidades del usuario. No es conveniente que ambos modos estén activos de manera simultánea; ya que generan problemas de conectividad y transmisión. Por ende, se debe escoger un modo a la vez para trabajar.

El cambio de un modo a otro se hace a través de hardware, haciendo un reseteo en el ESP32. Por cada reseteo que se realice, una variable de tipo entero "Modeldx" tomara un valor. Si la variable es igual a "1" es modo Bluetooth, de lo contrario el ESP32 estará en modo Wifi. (Véase figura 20)

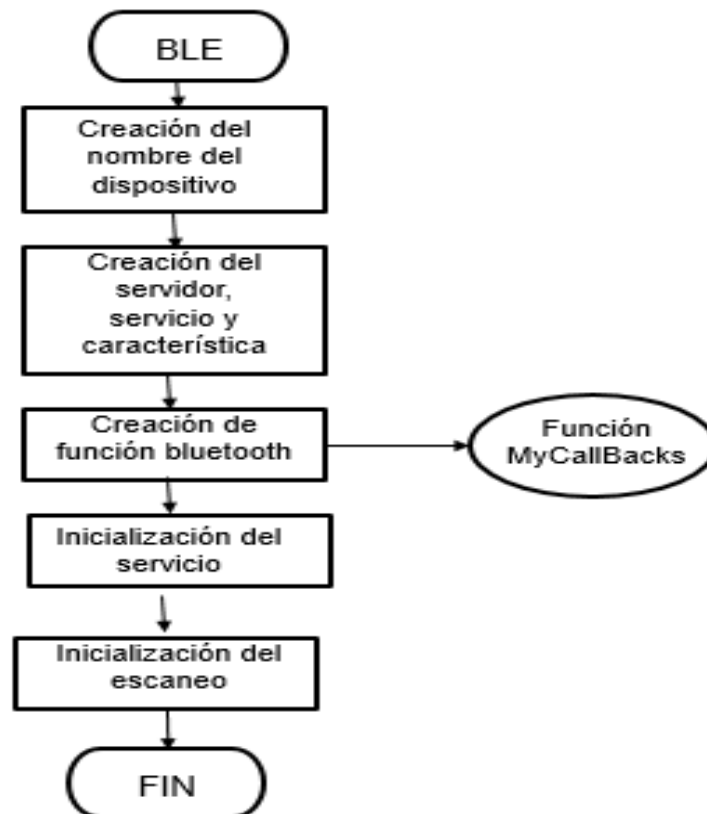


Figura 20. Diagrama de flujo Función Bluetooth
Fuente: Elaboración propia.

Si se escoge habilitar la opción de Bluetooth en el ESP32, el Wifi quedará deshabilitado, y se procede a realizar la configuración del bluetooth y posteriormente escribir en la EEPROM las credenciales del WiFi.

La configuración del bluetooth consta de la creación del nombre Bluetooth que tendrá nuestro microcontrolador (Este nombre será visible para otros dispositivos), creación e inicialización del servidor, servicio y característica (Propiedades del bluetooth Low Energy, véase figura 6) y la creación y llamada de la función MyCallbacks.

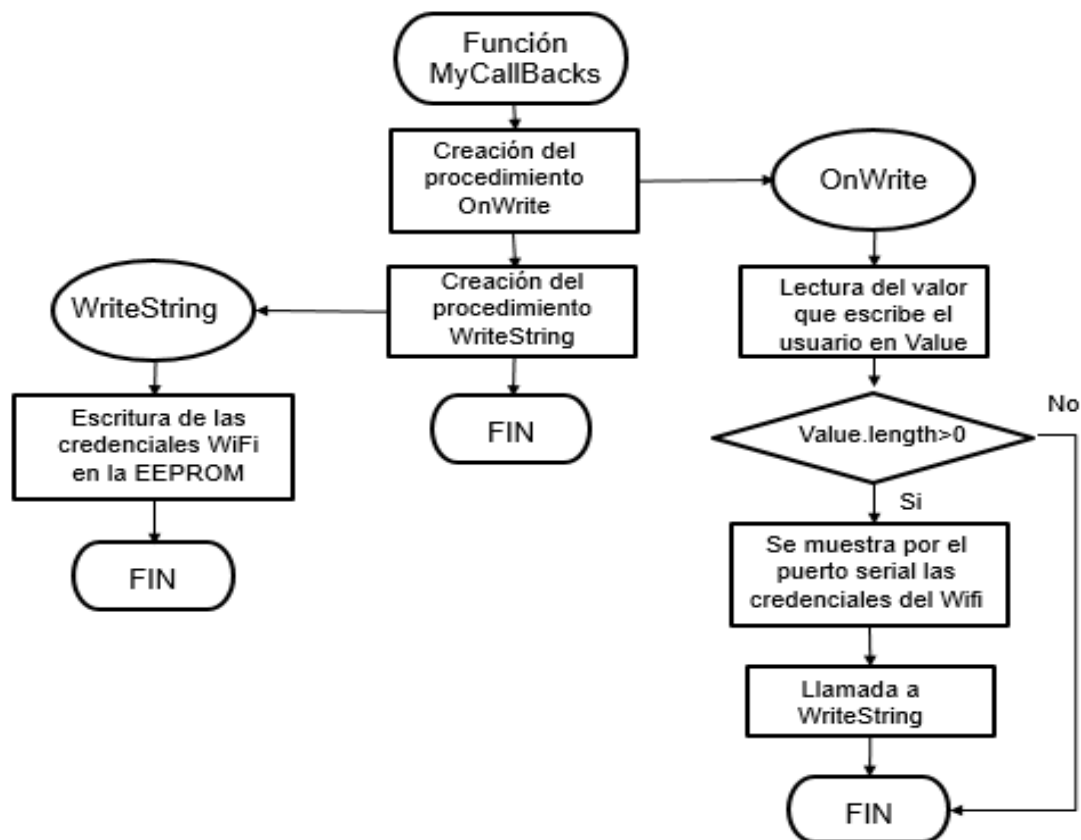


Figura 21. Diagrama de flujo Funcion MyCallbacks
Fuente: Elaboracion propia.

La funcion MyCallbacks es la encargada de recibir la data que escribe el usuario y almacenarla en la memoria EEPROM del ESP32. Para eso, ejecuta (2) procedimientos: OnWrite y WriteString.

OnWrite verifica que la data escrita por el usuario no sea NULL, y si es correcto llama a la funcion WriteString , la cual se encargará de guardar estos valores en la EEPROM.

A continuacion se muestra fragmentos del codigo para el modo Bluetooth

```
#include "EEPROM.h"
#include <WiFi.h>
#include <ESP32Ping.h>
#include "FirebaseESP32.h"
#include "DHTesp.h"
#include <stdlib.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
```

Figura 22. Librerias del programa para el ESP32
Fuente: Elaboracion propia.

La libreria EEPROM.h nos permite la lectura y escritura en la memoria EEPROM del ESP32. En esta memoria es donde se guardaran los datos de la red WiFi; estos no se modificaran o borrarán, al menos que ocurra un cambio por parte del usuario en la app.

Las librerias BLEDevice.h, BLEServer.h, BLEUtils.h y BLE902.h; son las encargadas de la inicializacion del bluetooth, creacion e inicializacion del servidor,

servicio, características y demás propiedades con que se desee trabajar según nuestros requerimientos.

```
void setup() {

    Serial.begin(115200);
    if(!EEPROM.begin(EEPROM_SIZE)){
        delay(1000);}
    delay(3000);
    modeIdx = EEPROM.read(modeAddr); //leo la dir de la Eeprom
    Serial.print("modeIdx : ");
    Serial.println(modeIdx);

    EEPROM.write(modeAddr, modeIdx !=0 ? 0 : 1);
    EEPROM.commit();

    if(modeIdx != 0){
        //BLE MODE
        Serial.println("BLE MODE");
        bleTask();

    }else{
        //WIFI MODE
        Serial.println("WIFI MODE");
        wifiTask();}
}
```

Figura 23. Void Setup ESP32
Fuente: Elaboración propia.

```
void bleTask(){
    // Create the BLE Device
    BLEDevice::init("ESP32 THAT PROJECT");

    // Create the BLE Server
    pServer = BLEDevice::createServer();
    pServer->setCallbacks(new MyServerCallbacks());

    // Create the BLE Service
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // Create a BLE Characteristic
    pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ |
        BLECharacteristic::PROPERTY_WRITE |
        BLECharacteristic::PROPERTY_NOTIFY |
        BLECharacteristic::PROPERTY_INDICATE
    );

    pCharacteristic->setCallbacks(new MyCallbacks());
}
```

Figura 24. Funcion BLE
Fuente: Elaboracion propia.

```
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
        BLEDevice::startAdvertising(); };

    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false; };

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic){
        std::string value = pCharacteristic->getValue();

        if(value.length() > 0){
            Serial.print("Value : ");
            Serial.println(value.c_str());
            writeString(wifiAddr, value.c_str());} }

    void writeString(int add, String data){
        int _size = data.length();
        for(int i=0; i<_size; i++){
            EEPROM.write(add+i, data[i]);}

        EEPROM.write(add+_size, '\0');
        EEPROM.commit(); };
```

Figura 25. Funcion MyCallBacks
Fuente: Elaboracion propia.

Una vez configurado el modo BLE, se procede a explicar como es el funcionamiento en modo WiFi. Vease figura 26.

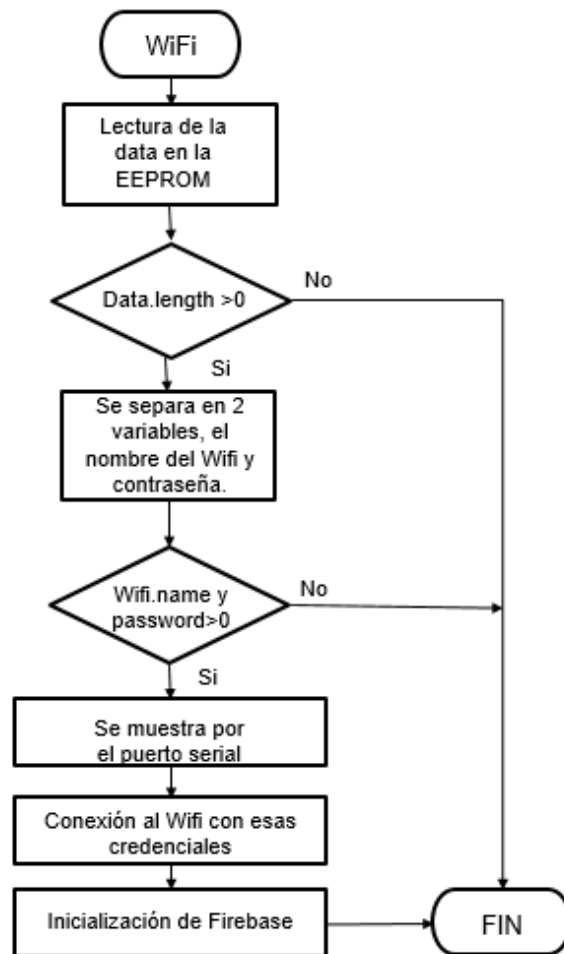


Figura 26. Diagrama de flujo Funcion Wifi
Fuente: Elaboracion propia.

Cuando el ESP32 esta en modo Wifi, leerá los valores almacenados en la EEPROM; ya que estas son las credenciales del WiFi con el que ahora se tendra conexion. Una vez leidos estos valores, se procede a confirmar si realmente hay datos almacenados o no. Si los hay, se separan y almacenan en (2) variables diferentes: Wifi.name y Wifi.password. Se realiza una nueva confirmación para saber si Wifi.name y Wifi.password contienen algún dato, y se procede a la conexión WiFi del ESP32.

Para poder establecer comunicación con el ESP32 vía WiFi, es necesario incluir la librería “WiFi.h”.

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

Figura 27. Librería Wifi.h
Fuente: Elaboración propia.

```
void wifiTask() {
    String receivedData;
    receivedData = read_String(wifiAddr);

    if(receivedData.length() > 0){
        String wifiName = getValue(receivedData, ',', 0);
        String wifiPassword = getValue(receivedData, ',', 1);

        if(wifiName.length() > 0 && wifiPassword.length() > 0){
            Serial.print("WifiName : ");
            Serial.println(wifiName);

            Serial.print("wifiPassword : ");
            Serial.println(wifiPassword);

            WiFi.begin(wifiName.c_str(), wifiPassword.c_str());
            Serial.print("Connecting to Wifi");
            while(WiFi.status() != WL_CONNECTED){
                Serial.print(".");
                delay(300);
            }
            Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
            Firebase.reconnectWiFi(true);
            Serial.print("Ping Host: ");
            Serial.println(remote_host);
        }
    }
}
```

Figura 28. Función Wifi
Fuente: Elaboración propia.

Una vez configurado el modo de funcionamiento del ESP32, se procede a la comunicación con los diferentes sensores y elementos que interactuarán en el sistema.

Configuración de sensores y elementos en el ESP32

El sistema interactuará con 2 sensores: DTH11 y sensor PIR, los cuales proporcionarán los diferentes datos que se visualizarán en la aplicación móvil.

Sensor DTH11:



Figura 29. Diagrama de flujo de sensor DTH11.
Fuente: Elaboración propia.

El sensor DTH11 puede leer valores de humedad y temperatura. El procedimiento para ambas lecturas es el mismo. Vease figuras 30 y 32.

Temperatura

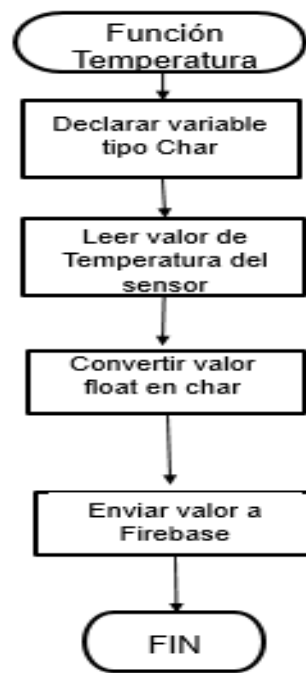


Figura 30. Diagrama de flujo funcion Temperatura.
Fuente: Elaboracion propia.

```
void Temperatura()
{
    char temp[20];
    delay(500);
    struct sensor mySensor;
    mySensor.deviceId = 1;
    mySensor.measurementType = "Temperatura";
    mySensor.value = dht.getTemperature();
    Serial.print("Temperatura:");
    Serial.println(mySensor.value);
    dtostrf(mySensor.value, 4, 2, temp);
    Firebase.setString(firebaseData, ruta + "/temperatura", temp);
    delay(10);
}
```

Figura 31. Funcion Temperatura.
Fuente: Elaboracion propia.

En la función temperatura se declara un arreglo de caracteres en el cual se guardarán los datos obtenidos del sensor, a través de la función “dht.getTemperature()”. Estos datos son de tipo float, por lo que es necesario convertirlos a tipo String. Para la conversión de datos, se utiliza la función `dtostrf()` y se envían a la base de datos, utilizando la función `Firestore.setString()`.

Humedad

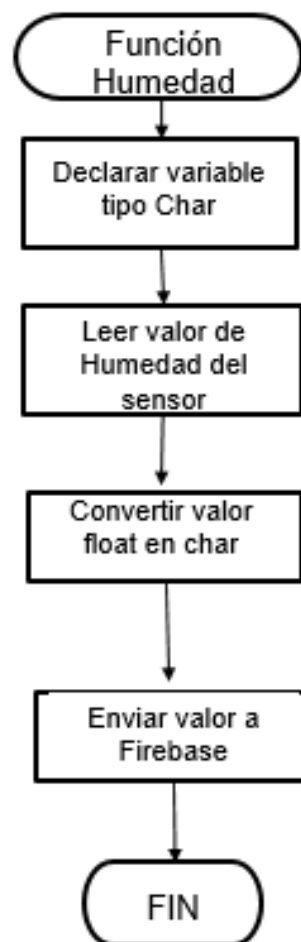


Figura 32. Diagrama de flujo Funcion Humedad
Fuente: Elaboración propia.

```

void Humedad ()
{
    char hum[20];
    struct sensor mySensor;
    mySensor.deviceId = 2;
    mySensor.measurementType = "Humedad";
    mySensor.value = dht.getHumidity();
    Serial.print("Humedad:");
    Serial.println(mySensor.value);
    dtostrf(mySensor.value,4,2,hum);
    Firebase.setString(firebaseData, ruta + "/humedad", hum);
    delay(500);
}

```

Figura 33. Función Humedad
Fuente: Elaboración propia.

El procedimiento para la lectura de estos datos, es similar al de temperatura. (Véase figura 31 y 33)

Sensor de Movimiento PIR:

Se declara una variable como entrada (Pirsensor) la cual se encargará de leer las señales que envíe el sensor. Este sensor envía señales digitales; cuando se detecta un movimiento pasará de BAJO a ALTO. Para que el programa detecte cuando ocurre un evento, utilizaremos interrupciones. La función a utilizar es "AttachInterrupt()", la cual tiene los siguientes parámetros:

- Pin usado para la interrupción.

La constante utilizada para el sensor PIR, debe ser declarado previamente. (Véase figura 34)

- Función utilizada cuando se interrumpe.

Es una función especial que se lleva a cabo cada vez que se interrumpe el pin. Se utiliza para cambiar el status de alguna variable, imprimir algún texto

o para algún delay. Es importante que sean pocas líneas de código; en nuestro caso solo se colocó 4 líneas. Véase figura 36.

- Modo de interrupción.

Controla como se dispara la interrupción. Existen 4 modos:

RISING: Cuando el pin pasa de BAJO a ALTO, se activa la interrupción

FALLING: Cuando el pin pasa de HIGH a LOW, se activa la interrupción

LOW: Cuando el pin es BAJO, se activa la interrupción

CHANGE: Cuando el pin cambia de BAJO a ALTO o ALTO a BAJO, se activa la interrupción.

Nos interesa detectar el cambio de bajo a alto en el sensor PIR, por eso, se configurará en RISING la interrupción. Véase figura 35.

```
const int PinLDR= 25;  
const byte PIRsensor = 26;  
int dhtPin = 27;  
int Ledroom= 13;  
int Ledbath = 14;  
int counterInter = 0;
```

Figura 34. Declaración de constante PIR
Fuente: Elaboración propia.

```
//Configuracion del sensor de movimiento  
pinMode(PIRsensor, INPUT_PULLUP);  
attachInterrupt(digitalPinToInterrupt(PIRsensor), detectsMovement, RISING);
```

Figura 35. Configuración PIR.
Fuente: Elaboración propia.

```
void IRAM_ATTR detectsMovement() {
  Serial.println("Movimiento detectado!!!");
  startTimer = true;
  lastTrigger = millis();
  contar = contar +1;
}
```

Figura 36. Función para la interrupción PIR.
Fuente: Elaboración propia.

Control de luces:

En esta sección, el usuario decidirá qué información enviarle al micro-controlador (si encender o apagar las luces de diferentes áreas de una casa) desde la app. Por ende, se lee directamente a la base de datos, la información que contenga el área que se desea controlar. Una vez leído el valor, se procede a la comprobación del estado. Si está ON, la luz de esa área se encenderá, en caso contrario se apagará. (Véase figura 37)

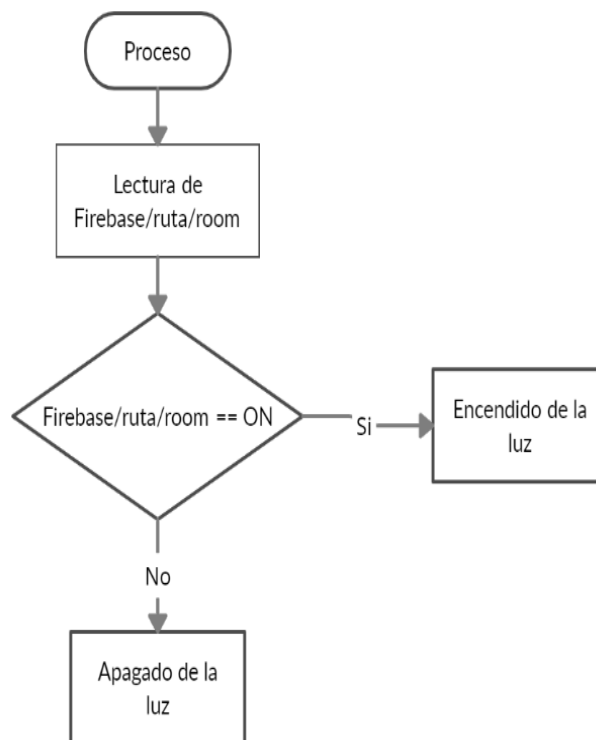


Figura 37. Diagrama de flujo status de las luces para ON/OFF
Fuente: Elaboración propia.

```

void lucess ()
{
  Firebase.getString(firebaseData, ruta + "/lucesroom");
  Serial.println(firebaseData.stringData());
  if (firebaseData.stringData() == "ON") {
    Serial.println("Luz del cuarto encendida");
    digitalWrite(Ledroom, HIGH);
  }
  else {
    Serial.println("Luz del cuarto apagada");
    digitalWrite(Ledroom, LOW);
  }
}

```

Figura 38. Comprobación status de las luces para ON/OFF room
Fuente: Elaboración propia.

Comunicación ESP32 con Firebase:

Luego que el ESP32 esté conectado exitosamente a una red WiFi, y la medición de los datos con los sensores funcione correctamente; se procede a enviar esa data a la base de datos en Firebase. Pero, ¿Cómo se conecta el ESP32 con Firebase? En primer lugar, creamos una cuenta en la página web de Firebase.

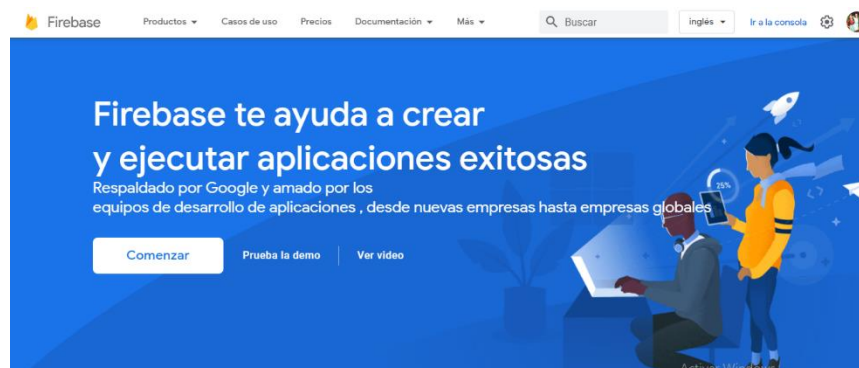


Figura 39. Página web Firebase
Fuente: Elaboración propia.

Luego se agrega un proyecto. En nuestro caso, el proyecto se llama Esp32-flutter.



Figura 40. Creación de un proyecto en Firebase.
Fuente: Elaboración propia.

Y se procede a crear la base de datos que necesitaremos para almacenar los datos.

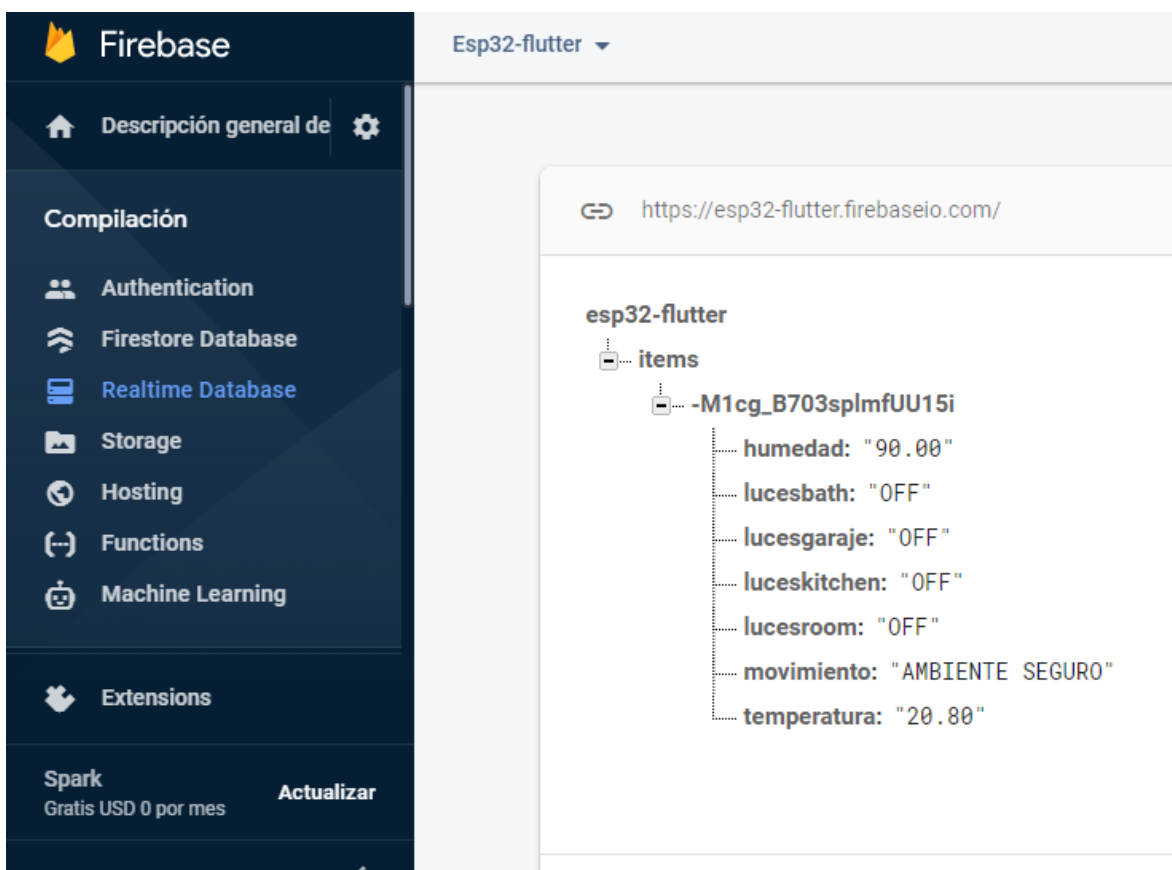


Figura 41. Creación de base de datos.
Fuente: Elaboración propia.

La base de datos se crea en Realtime Database. Se necesita poder leer (datos del sensor DTH11 y sensor de movimiento) y escribir (control de luces) en la BD. Por ende, se debe configurar los permisos para poder realizar estas operaciones. Véase figura 42.

La configuración de las reglas, se realiza en la sección “Reglas” que está en la parte superior de nuestra base de datos.

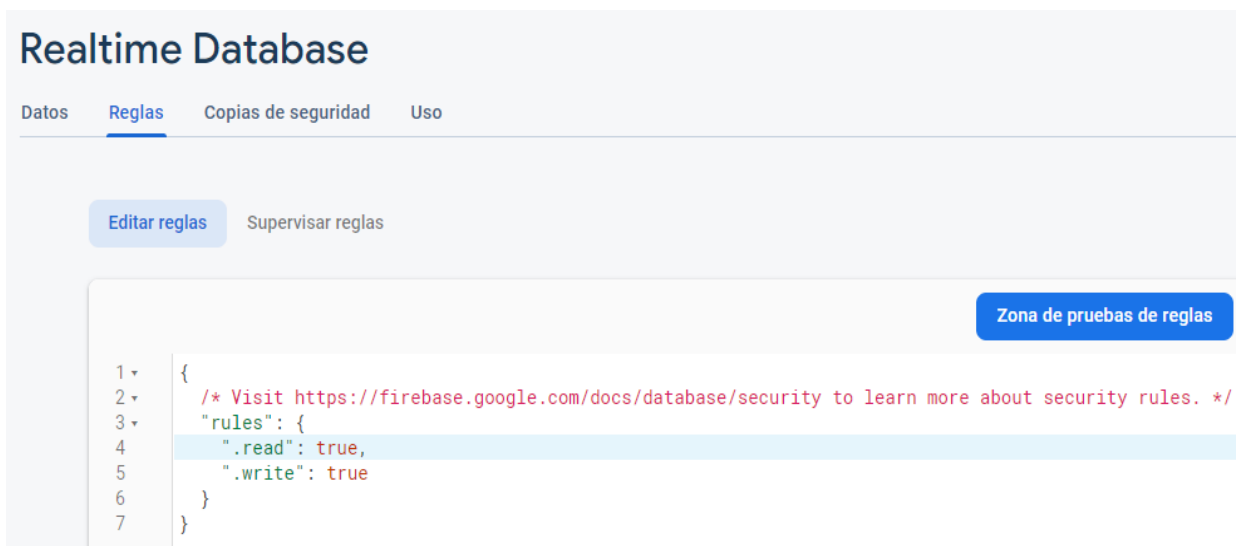


Figura 42. Configuración de reglas Firebase.
Fuente: Elaboración propia.

Como se visualiza en la figura 42 tanto la opción de lectura como escritura están habilitadas.

Realizado lo anterior, nos dirigimos a la configuración general del proyecto

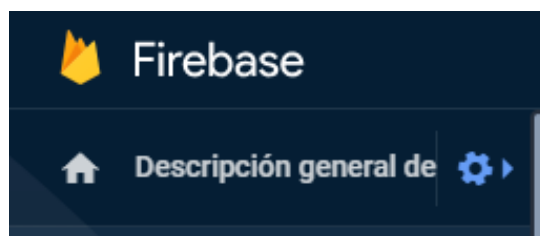


Figura 43. Configuración general del proyecto.
Fuente: Elaboración propia.

Y se procede a copiar (2) datos importantes de la sección “Cuentas de servicio”. Estos son: database URL y el token o llave de Firebase.

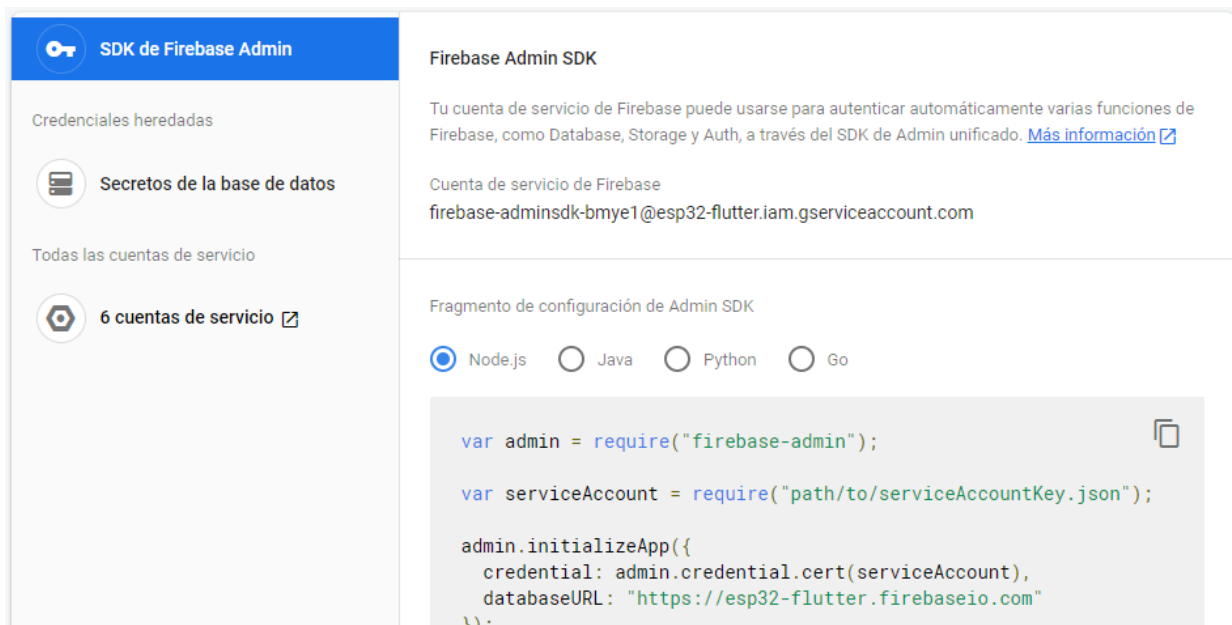


Figura 44. DatabaseURL del proyecto.
Fuente: Elaboración propia.

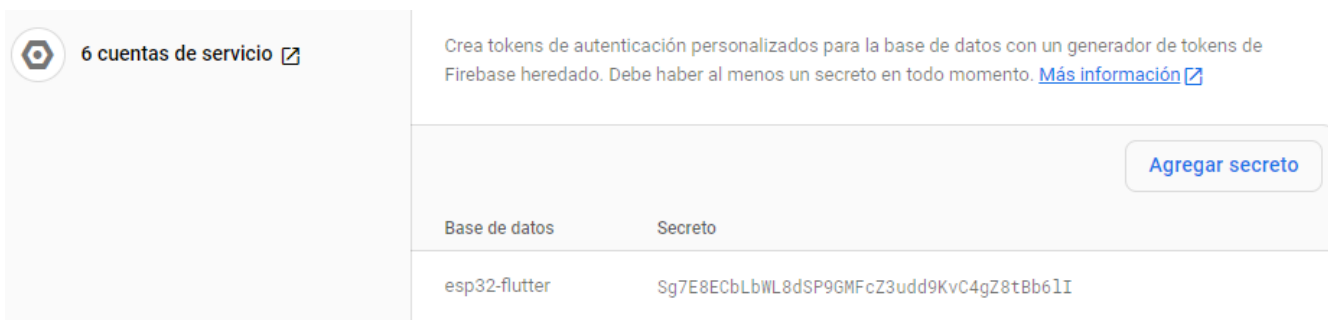


Figura 45. Token de la base de datos.
Fuente: Elaboración propia.

Estos datos son los que permiten la comunicación con el ESP32, y se deben agregar en el programa en ARDUINO IDE. Véase figura 47.

En el código del ESP32, primero se debe agregar la librería FirebaseESP32.h; la cual permitirá la comunicación con la base de datos alojada en Firebase.

```
#include <ESP32Ping.h>
#include "FirebaseESP32.h"
```

Figura 46. Librería FirebaseESP32.h
Fuente: Elaboración propia.

Luego se agregan los datos extraídos de la BD.

```
#define EEPROM_SIZE 128
#define SERVICE_UUID      "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define timeSeconds 2
#define FIREBASE_HOST "esp32-flutter.firebaseio.com"
#define FIREBASE_AUTH "Sg7E8ECbLbWL8dSP9GMFcZ3udd9KvC4gZ8tBb6lI"
```

Figura 47. Datos de Firebase en código ESP32.
Fuente: Elaboración propia.

Una vez realizado el proceso de configuración en el ESP32, se tiene comunicación con la BD en Firebase. Ahora, se procede a inicializarla con los datos antes vistos en la figura 47. Este procedimiento se realiza cuando se tiene conexión estable con la red Wifi.

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
Firebase.reconnectWiFi(true);
Serial.print("Ping Host: ");
Serial.println(remote_host);

if(Ping.ping(remote_host)){
    Serial.println("Success!!");
}else{
    Serial.println("ERROR!!");
}

}

}
```

Figura 48. Inicialización BD Firebase.
Fuente: Elaboración propia.

En la figura 48 se puede observar una conexión con un ping remoto, luego que se inicializa la BD. Este ping remoto es el host de google y esto se realiza para verificar que se tiene acceso a internet.

```
const char* remote_host = "www.google.com";
```

Figura 49. Host remoto.
Fuente: Elaboración propia.

Luego que se inicialice la comunicación entre el ESP32 y Firebase, se pueden realizar operaciones de lectura y escritura en la BD.

Para escritura:

Se pueden utilizar 4 funciones para guardar datos en Firebase.

Set: Escribe o reemplaza datos en una ruta de acceso definida, como messages/users/<username>

Update: Actualiza algunas de las claves de una ruta de acceso definida sin reemplazar todos los datos

Push: Agrega datos a una lista de datos en la base de datos. Cada vez que se envía un nodo nuevo a una lista, la base de datos genera una clave única, como messages/users/<unique-user-id>/<username>

Transaction: Se usa cuando se trabaja con datos complejos que podrían dañarse con las actualizaciones simultáneas

En nuestro caso, se utilizó la función Set; por lo tanto, cada vez que el sensor envíe un dato al ESP32, éste lo escribirá en Firebase en la ruta correspondiente. Véase figura 31 y 33.

La función Set va acompañada del tipo de dato que se desea escribir, por ejemplo, puede ser Firebase.setString, Firebase.setInt, Firebase.setFloat, etc.

Para lectura:

Se utiliza la función `Firebase.getString` y se le pasa como parámetro la ruta correspondiente. Esta función varía según el tipo de dato que se quiera leer y esté almacenado en Firebase. Por ejemplo, `Firebase.getString`, `Firebase.getInt`, `Firebase.getFloat`, etc.

Una vez explicado el proceso de selección de modo (WiFi o Bluetooth), como se realizan las mediciones de los sensores y el proceso de comunicación del ESP32 con Firebase; se procede a explicar la creación de la aplicación móvil y como esta, se conecta con el sistema antes descrito.

Aplicación móvil

En la actualidad un 80% de la información o contenido que tenemos en nuestros dispositivos móviles, están en aplicaciones. Estas, comunican, integran y simplifican muchas tareas diarias que realizamos. El objetivo de la aplicación móvil a diseñar es que sea sencilla, cómoda y con una interfaz gráfica que sea amigable con el usuario. Para el diseño y desarrollo se utilizó Flutter como framework principal, éste utiliza DART como lenguaje de programación. El editor de código a utilizar es VS CODE.

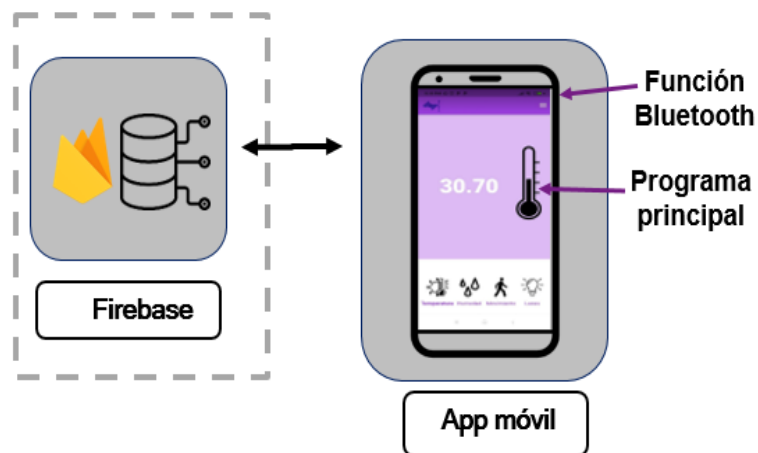


Figura 50. Esquema Aplicación móvil
Fuente: Elaboración propia.

La sección del proyecto, consta de 3 partes divididas en: Comunicación Firebase – Flutter, Función Bluetooth y Programa principal.

Comunicación Firebase – Flutter

Es la encargada en comunicar todo el proyecto Firebase con la aplicación que se desea crear. Para lograr esta comunicación se debe seguir una serie de pasos:

Paso 1:

Abrir la configuración general del proyecto de Firebase.

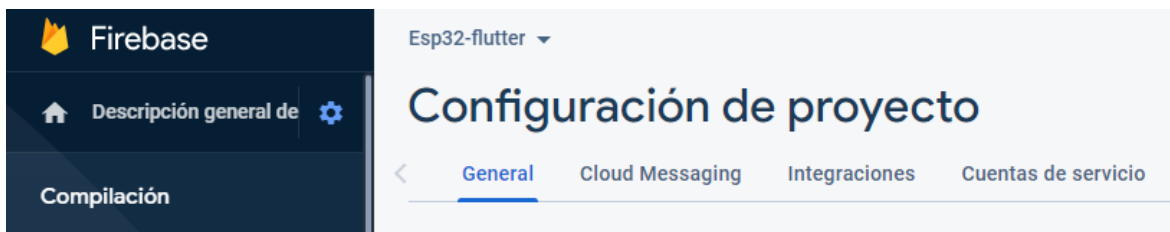


Figura 51. Configuración general del proyecto Firebase.
Fuente: Elaboración propia.

Paso 2:

Agregar una aplicación al proyecto y escoger el sistema operativo de la misma.

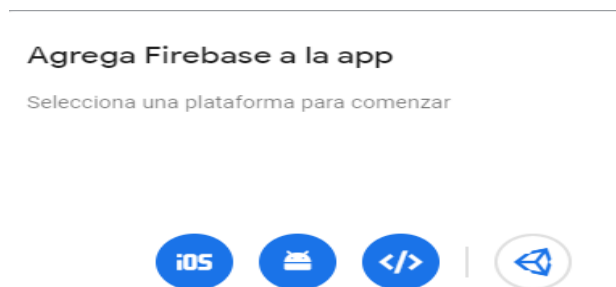


Figura 52. Agregar app a Firebase.
Fuente: Elaboración propia

Paso 3:

Registrar la aplicación en Firebase. En esta sección se debe colocar el nombre de nuestra aplicación. En este caso, el nombre es “Tesis_Unexpo”



× **Agrega Firebase a tu app para Android**

1 Registrar app

Nombre del paquete de Android ⓘ

com.example.tesis

Sobrenombre de la app (opcional) ⓘ

Tesis_Unexpo

Certificado de firma SHA-1 de depuración (opcional) ⓘ

Obligatoria para Dynamic Links y el Acceso con Google o la asistencia con un número de teléfono en Auth. Puedes editar las claves SHA-1 en Configuración.

Siguiente

Figura 53. Registro de la app
Fuente: Elaboración propia

Paso 4:

Descargar el archivo de configuración “google-services.json” y copiarlo en VS CODE.

2

Descargar archivo de configuración

Instrucciones para Android Studio a continuación | [Unity](#) [C++](#)

⬇ Descargar google-services.json

Cambia a la vista **Proyecto** de Android Studio para ver el directorio raíz de tu proyecto.

Coloca el archivo google-services.json que acabas de descargar en el directorio raíz del módulo de tu app para Android.

google-services.json

Project Packages Scratches

MyApplication (~/.Desktop/My

.gradle

.idea

app

build

libs

src

.gitignore

app.iml

build.gradle

google-services.json

proguard-rules.pro

gradle

Figura 54. Descarga de archivo .Json
Fuente: Elaboración propia

The image shows the VS Code Explorer sidebar. The root folder is 'TESIS'. It contains several subfolders and files: '.dart_tool', '.idea', '.vscode' (containing 'launch.json'), 'android' (containing '.gradle' and 'app'), and 'app' (containing '471', 'Code', 'src', 'build.gradle', and 'google-services.json'). The 'google-services.json' file is currently selected and highlighted with a blue background.

Figura 55. Archivo google-services.json en VS CODE.
Fuente: Elaboración propia

73

Paso 5:

Agregar el SDK de Firebase a nuestro proyecto en VS CODE. Esto se realiza modificando los archivos build.gradle en VS CODE.

Archivo build.gradle de nivel de proyecto (<project>/build.gradle):

```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
        // Add this line
        classpath 'com.google.gms:google-services:4.3.8'
    }
}

allprojects {
    ...
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
        ...
    }
}
```

Figura 56. Modificación de archivo build.gradle

Fuente: Elaboración propia

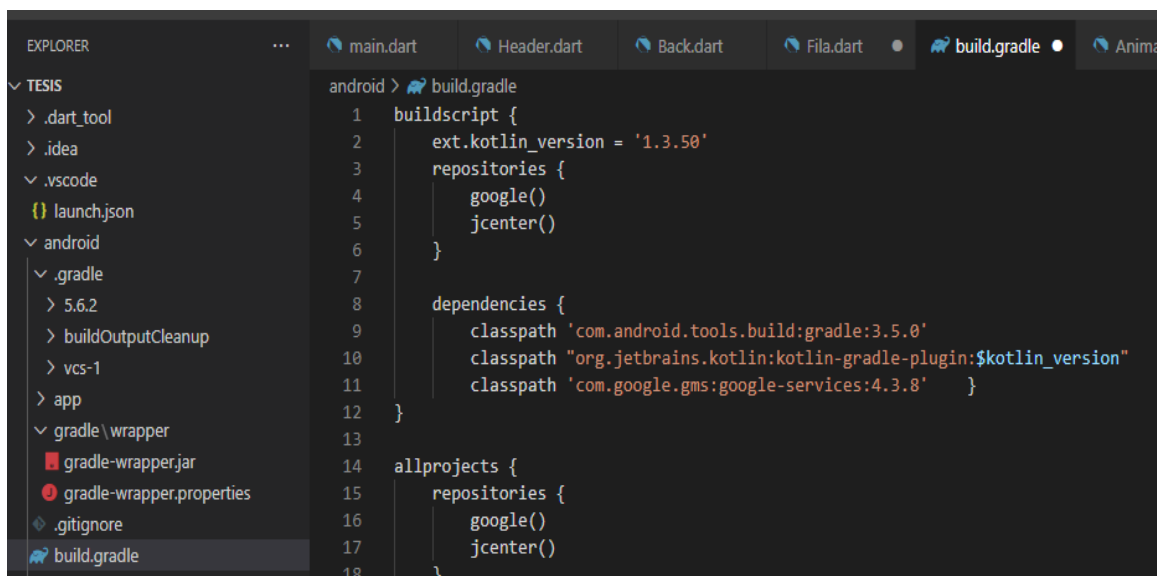


Figura 57. Modificación de archivo build.gradle en VS CODE

Fuente: Elaboración propia

Archivo build.gradle de nivel de app (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:28.2.0')

    // Add the dependency for the Firebase SDK for Google Analytics
    // When using the BoM, don't specify versions in Firebase dependencies
    implementation 'com.google.firebase:firebase-analytics'

    // Add the dependencies for any other desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Figura 58. Modificación de archivo app/ build.gradle
Fuente: Elaboración propia

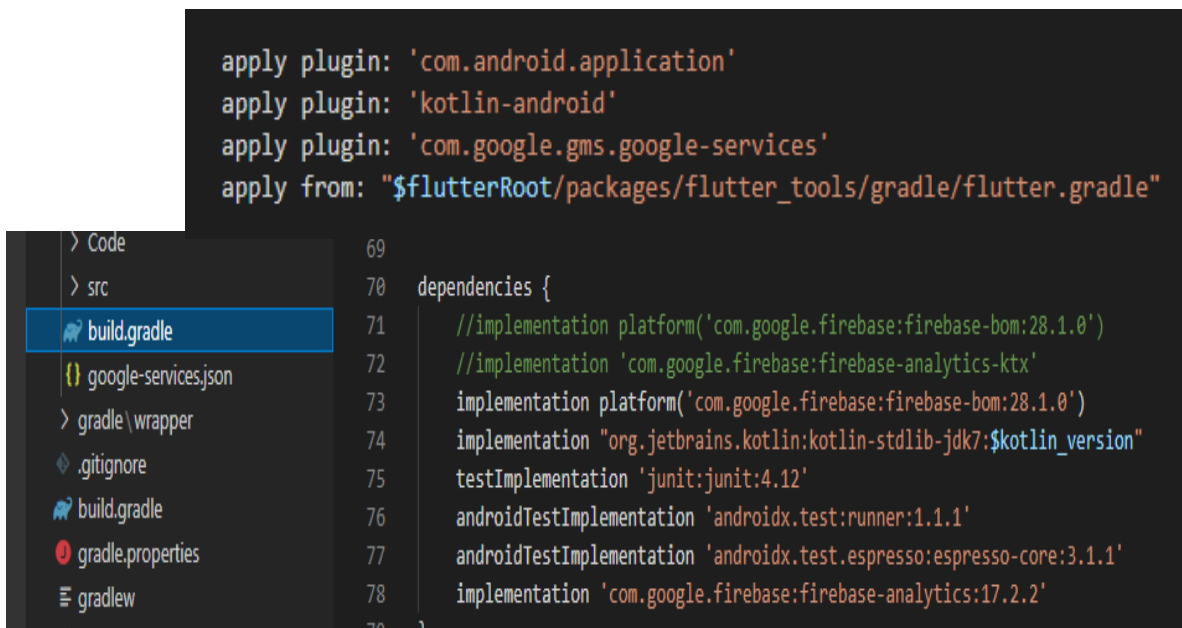


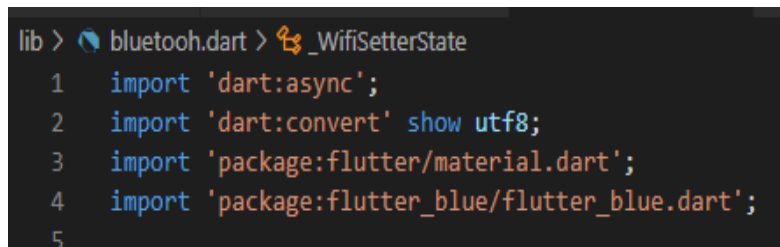
Figura 59. Modificación de archivo app/ build.gradle en VS CODE
Fuente: Elaboración propia

Una vez modificado los archivos, se procede a correr la aplicación Flutter en VS CODE, y ya se tiene comunicación con Firebase.

Lista la comunicación entre la aplicación y Firebase, se procede a explicar la creación de la función Bluetooth y de la pantalla principal de la aplicación Flutter del proyecto.

Función bluetooth

Es el archivo encargado de establecer la comunicación con el ESP32 vía bluetooth. Para realizar un archivo en el programa principal, es necesario colocar la extensión .DART. Luego incluir todos los paquetes y librerías asociadas a nuestro programa.



```
lib > bluetooth.dart > _WifiSetterState
1  import 'dart:async';
2  import 'dart:convert' show utf8;
3  import 'package:flutter/material.dart';
4  import 'package:flutter_blue/flutter_blue.dart';
5
```

Figura 60. Paquetes y bibliotecas para Bluetooth VS.
Fuente: Elaboración propia

Para la comunicación con el ESP32 o cualquier dispositivo en aplicaciones futuras, es necesario colocar el UUID del servicio, característica y el nombre del dispositivo, ya que queremos que conecte exclusivamente con el ESP32. Estos datos son los mismos que se colocaron en el código Arduino del ESP32.

Estas declaraciones van dentro de una class que contiene un “StatefulWidget”, los cuales devuelven un valor y/o ejecutan una función después de ser llamados. También están los StatelesWidget, los cuales no devuelven valor luego de ejecutarse (mantienen su estado inicial).

```

class WifiSetter extends StatefulWidget {
  @override
  _WifiSetterState createState() => _WifiSetterState();
}

class _WifiSetterState extends State<WifiSetter> {
  final String SERVICE_UUID = "4fafc201-1fb5-459e-8fcc-c5c9c331914b";
  final String CHARACTERISTIC_UUID = "beb5483e-36e1-4688-b7f5-ea07361b26a8";
  final String TARGET_DEVICE_NAME = "ESP32 THAT PROJECT";

  FlutterBlue flutterBlue = FlutterBlue.instance;
  StreamSubscription<ScanResult> scanSubscription;

```

Figura 61. StatefulWidget para bluetooth
Fuente: Elaboración propia.

```

bluetooth.dart > _WifiSetterState
@override
void initState() {
  super.initState();
  startScan();
}

startScan() {
  setState(() {
    connectionText = "Escaneando";
  });

  scanSubscription = flutterBlue.scan().listen((scanResult) {
    print(scanResult.device.name);
    if (scanResult.device.name.contains(TARGET_DEVICE_NAME)) {
      stopScan();

      setState(() {
        connectionText = "Tarjeta ESP32 encontrada";
      });

      targetDevice = scanResult.device;
      connectToDevice();
    }
  }, onDone: () => stopScan());
}

```

Figura 62. Inicialización del escaneo bluetooth.
Fuente: Elaboración propia.

StarScan(), comienza a buscar conexiones bluetooth cercanas. El objetivo de esta función es encontrar el ESP32 (Que funciona como servidor) y conectarse a él. Por ello, luego que busca todos los bluetooth, pregunta si alguno de los dispositivos encontrados contiene el nombre del ESP32, si es así, se detiene el

escaneo o búsqueda y se conecta al ESP32. Una vez establecida la conexión, se muestra el diseño (véase en la figura 63) para colocar las credenciales WiFi.



Figura 63. Diseño configuración WiFi ESP32.
Fuente: elaboración propia

```
writeData(String data) async {  
  if (targetCharacteristic == null) return;  
  
  List<int> bytes = utf8.encode(data);  
  await targetCharacteristic.write(bytes);  
}  
  
@override  
void dispose() {  
  super.dispose();  
  stopScan();  
}  
  
submitAction() {  
  var wifiData = '${wifiNameController.text},${wifiPasswordController.text}';  
  writeData(wifiData);  
}  
  
TextEditingController wifiNameController = TextEditingController();  
TextEditingController wifiPasswordController = TextEditingController();
```

Figura 64. Credenciales WiFi
Fuente: elaboración propia.

Programa principal

Contiene el diseño, recepción y envío de datos e interfaz gráfica de la aplicación.

La estructura de la misma está constituida por las siguientes partes: Fondo de la app, diseño de imágenes y recepción de datos.

- Para el fondo de la app:
Se divide en 3 partes: AppBar, contenedor 1 y contenedor 2. El AppBar es la franja superior que nos muestra el identificador del proyecto (logo de la Unexpo) y el icono del menú. En nuestro caso, este icono direccionará a la pantalla Bluetooth.

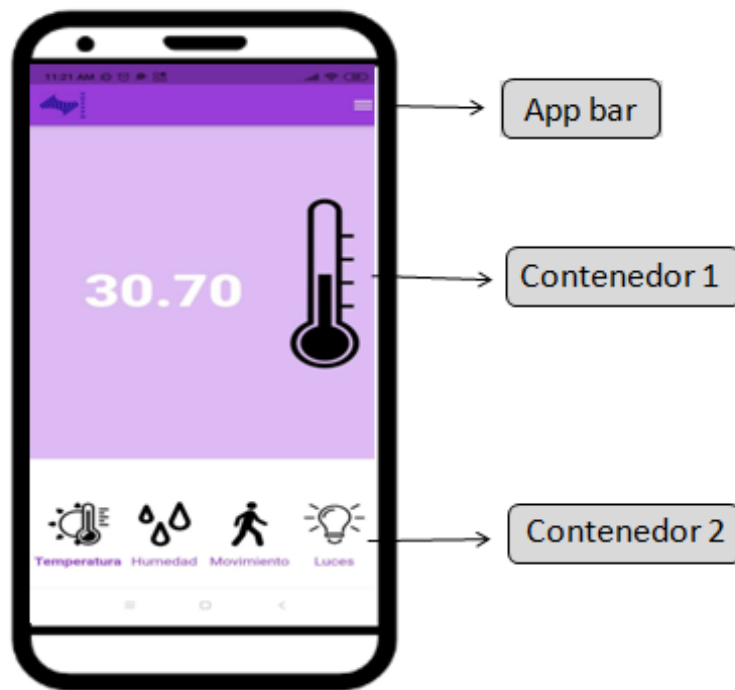


Figura 65. Composición base de la app
Fuente: Elaboración propia.

Para el contenedor 1 se utilizó el widget container(), el cual crea un contenedor rectangular personalizado (ancho, alto, color, borde, entre otros). Este widget contiene los datos recibidos desde el ESP32. Por defecto, la imagen inicial será la de Temperatura, esta será la primera vista una vez abramos la aplicación.

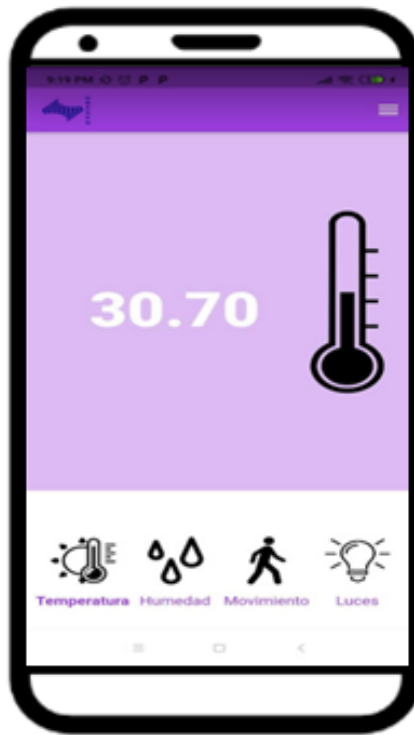


Figura 66. Imagen principal de la app.
Fuente: Elaboración propia

Para el contenedor 2, también se utilizó el widget container. Aquí se dispondrá de selectores para monitorear la variable que el usuario desee. Para la selección se utilizó la función Gesturedetector(), que detectará cuando se toque u oprima alguna variable.



Figura 67. Selector de variable.
Fuente: Elaboración propia.

A continuación, se muestran fragmentos del código para la programación del fondo de la aplicación. Véase figuras 68,69,70.

```
class _HeaderState extends State<Header> {
  @override
  Widget build(BuildContext context) {

    return new Scaffold(
      endDrawer: Drawer(
        child: new WifiSetter(),
      ), // Drawer
      appBar: AppBar(
        backgroundColor: Color(0xFF973CD7),
        title: Image.asset(
          "assets/images/unexpo.png",
          height: 40.0,
        ), // Image.asset
      ), // AppBar
    );
  }
}
```

Figura 68. AppBar
Fuente: Elaboración propia.


```

class GradientBack extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Container (
      decoration: new BoxDecoration(
        color:
          Color(0xFFDCBBF2)
      ) // BoxDecoration
    ); // Container
  }
}

```

Figura 69. Contenedor 1.
Fuente: Elaboración propia.

```

class BackWheater extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Stack(
      children: <Widget>[
        new GradientBack(),
        new Positioned(
          bottom: 0.0,
          child: new Container(
            width: MediaQuery.of(context).size.width,
            height: 175.0,
            color : Colors.white,
          ) // Container
        ), // Positioned
      ], // <Widget>[]
    ); // Stack
  }
}

```

Figura 70. Contenedor 2
Fuente: Elaboración propia

- Diseño para visualización de imágenes

Muestra los datos recibidos de Firebase y la interfaz principal de la aplicación. Consta de 2 partes fundamentales: Visualización de variables y selección de variables.

- Visualización de variables

Son 4 variables las que se visualizarán en la aplicación (temperatura, humedad, movimiento y luces). El diseño de las variables temperatura y humedad, está constituido por un widget Row(); es una fila que tiene n cantidad de “hijos” y cada uno de ellos puede ser un widget diferente. Sin embargo, se debe considerar el espacio disponible en la pantalla; ya que, Row(), no tiene desplazamiento.

Para el diseño de la app, se colocarán 2 hijos: Uno que contendrá el texto y otro con la imagen que lo acompañará.

Row() es totalmente personalizable (ancho, alto, color, borde, alineación, entre otros)

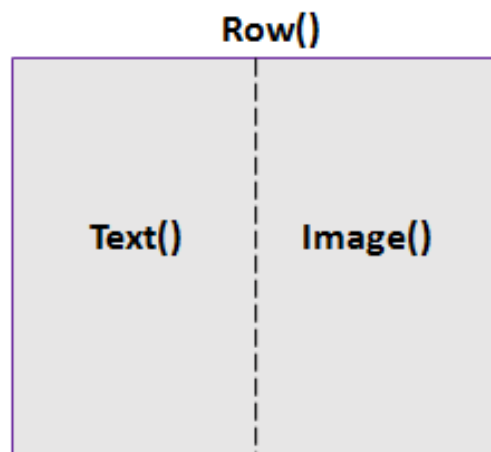


Figura 71. Estructura de Widget Row().
Fuente: Elaboración propia

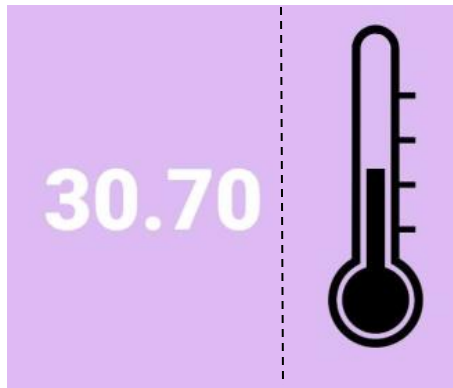


Figura 72 (a). Row () Temperatura Figura 72 (b). Row () Humedad
Fuente: Elaboración propia

```
return new Row(
  children: <Widget>[
    //Widget para mostrar valor de temperatura
    new InkWell(
      child: Padding(
        padding: EdgeInsets.only(
          top: 120,
          left: 60,
        ), // EdgeInsets.only
        child: Text(
          items[index].temperatura != null? items[index].temperatura: 'Default',
          style: const TextStyle(
            fontSize: 60.0,
            color: Color(0xFFFFFFFF),
            fontWeight: FontWeight.w900
          ) // TextStyle
        ), // Text
      ), // Padding
    ), // InkWell
  ],
);
```

Figura 73. Programación para visualización de los datos de temperatura.
Fuente: Elaboración propia.

```
return new Row(
  children: <Widget>[
    new InkWell(
      child: Padding(
        padding: EdgeInsets.only(
          top: 120,
          left: 60,
        ), // EdgeInsets.only
        child: Text(
          items[index].humedad != null? items[index].humedad: 'Default',
          style: const TextStyle(
            fontSize: 60.0,
            color: Color(0xFFFFFFFF),
            fontWeight: FontWeight.w900
          ) // TextStyle
        ), // Text
      ), // Padding
    ), // InkWell
  ],
);
```

Para la variable movimiento, se utilizó el widget Stack (). Este puede tener n cantidad de hijos y cada uno de ellos ser un widget diferente. Al igual que Row(), no tiene desplazamiento; así que, hay que considerar el tamaño de la pantalla.



Figura 74. Stack() Movimiento
Fuente: Elaboración propia

La variable movimiento puede tener (2) estados: “AMBIENTE SEGURO” o “ALERTA”. Cuando el sensor PIR no detecta ningún movimiento, enviara una señal de “AMBIENTE SEGURO” al ESP32, y este a Firebase. Si se detecta movimiento, el mensaje cambiará a “ALERTA”.



Figura 75 Stack Movimiento en “ALERTA”.
Fuente: Elaboración propia.

A continuación, se muestran fragmentos de la programación para el Stack Movimiento.

```
//Comprobacion para mostrar los valores de movimiento, almacenados en DB
else if (valor== 'movimiento'){
  if (items[index].movimiento=='AMBIENTE SEGURO'){
    return new Stack(
      children: <Widget>[
        new InkWell(
          child: Padding(
            padding: EdgeInsets.only(
              top: 90,
              left: 10,
            ), // EdgeInsets.only
            child: Text(
              items[index].movimiento != null? items[index].movimiento: 'Default',
              textAlign: TextAlign.center,
              style: const TextStyle(
                fontSize: 50.0,
                color: Color(0xFFFFFFFF),
                fontWeight: FontWeight.w900,
              ),
            ), // Text
          ), // Padding
        ), // InkWell
```

Figura 76. Programación Stack() Movimiento.
Fuente: Elaboración propia.

Por último, se tiene el diseño de las variables “luces”. La aplicación puede controlar las luces de 4 áreas: habitación, baño, cocina y garaje.



Figura 77. Diseño para luces
Fuente: Elaboración propia.

Se utilizó el widget `ToggleButtons()`, el cual nos permite diseñar una lista de botones independientes entre sí. Cada botón almacena un estado diferente y pueden estar “presionados” varios a la vez, sin que afecte el estado de los demás.



Figura 78. ToggleButtons con todas las luces encendidas.
Fuente: Elaboración propia.

El widget `ToggleButtons` permite cambiar los colores de los botones en estado activo e inactivo, los iconos internos, tamaño, entre otros. Para la comprobación del estado de cada botón, lo manejamos como un array y se verifica con el condicional `If` el status de cada botón. El estatus se almacena en la variable `isSelected` y cambia de estatus (`TRUE`) cuando es presionado. Una vez realizada la verificación se pasa el valor “ON” u “OFF” a la base de datos y esta lo pasa al ESP32.

```
//Comprobacion del boton seleccionado para el envio de data a DB
if (isSelected[0] == true){
  itemRef.child("-M1cg_B703splmfUU15i/lucesroom").set("ON");
}
else
  itemRef.child("-M1cg_B703splmfUU15i/lucesroom").set("OFF");
```

Figura 79. Comprobación del botón seleccionado.

- Selector de variables

Es la parte encargada de detectar que variable selecciona el usuario. Su diseño está en un widget Container que tiene como hijo un Row(). Dentro del Row() está el diseño para las (4) variables. Al usuario seleccionar cualquiera de las variables para su visualización, se activa el GestureDetector() y se procede a mostrar los datos e imagen asociado a esa variable.

```
children: <Widget> [
  GestureDetector(
    onTap: (){
      setState(() {
        sensorSelec = SelectionS.Temperatura;
        valor = 'temperatura';
      });
    },
    child: Padding(
      padding: EdgeInsets.all(12.0),
      child: Image.asset(
        "assets/images/temperatura2.png",
        width: 60,
        height: 60,
      ), // Image.asset
    ), // Padding
  ), // GestureDetector
  Text ("Temperatura",
    style: TextStyle(
      color: Color(0xFF973CD7),
      fontWeight: sensorSelec == SelectionS.Temperatura ? FontWeight.bold : FontWeight.normal,
      fontSize: 15,
    ), // TextStyle
  ), // Text
],
```

Figura 80. Selector de variable
Fuente: Elaboración propia.

CAPITULO V

RESULTADOS

En este capítulo mostraremos los resultados obtenidos del proyecto, comenzando desde las pruebas básicas hasta el resultado final.

Pruebas con Arduino IDE Modo Wifi - Bluetooth:

Pruebas iniciales para selección de modo Bluetooth o WiFi. Si Mode Idx = 1, es modo Bluetooth; de lo contrario es Wifi.

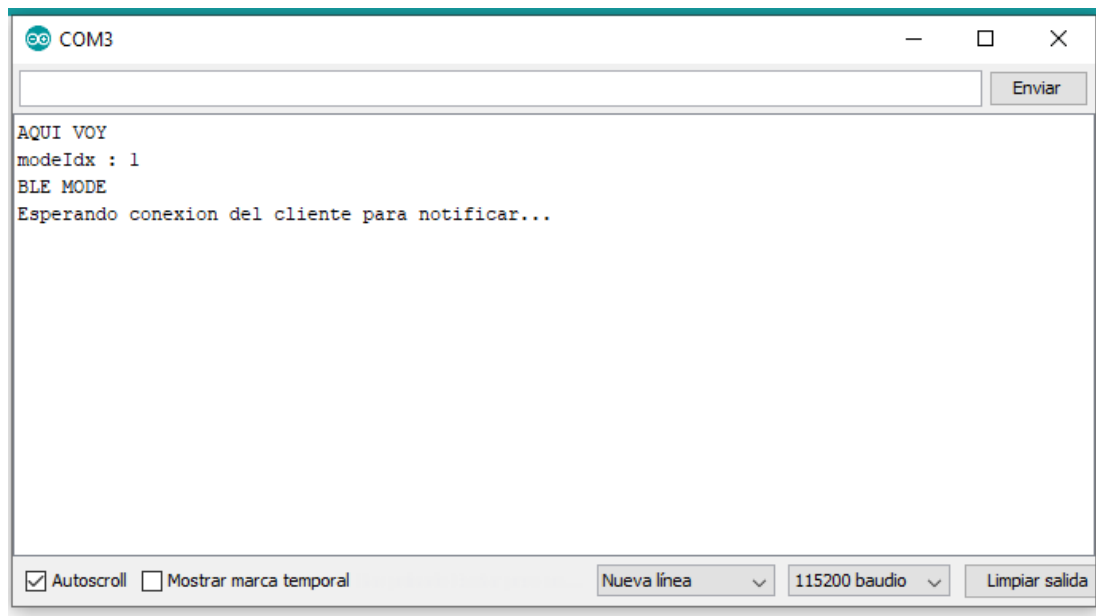


Figura 81. Arduino IDE, Modo Bluetooth.
Fuente: Elaboración propia.

En la aplicación móvil, el modo Bluetooth inicialmente se muestra así. En este punto el cliente (app) escanea y busca el bluetooth del servidor (ESP32)

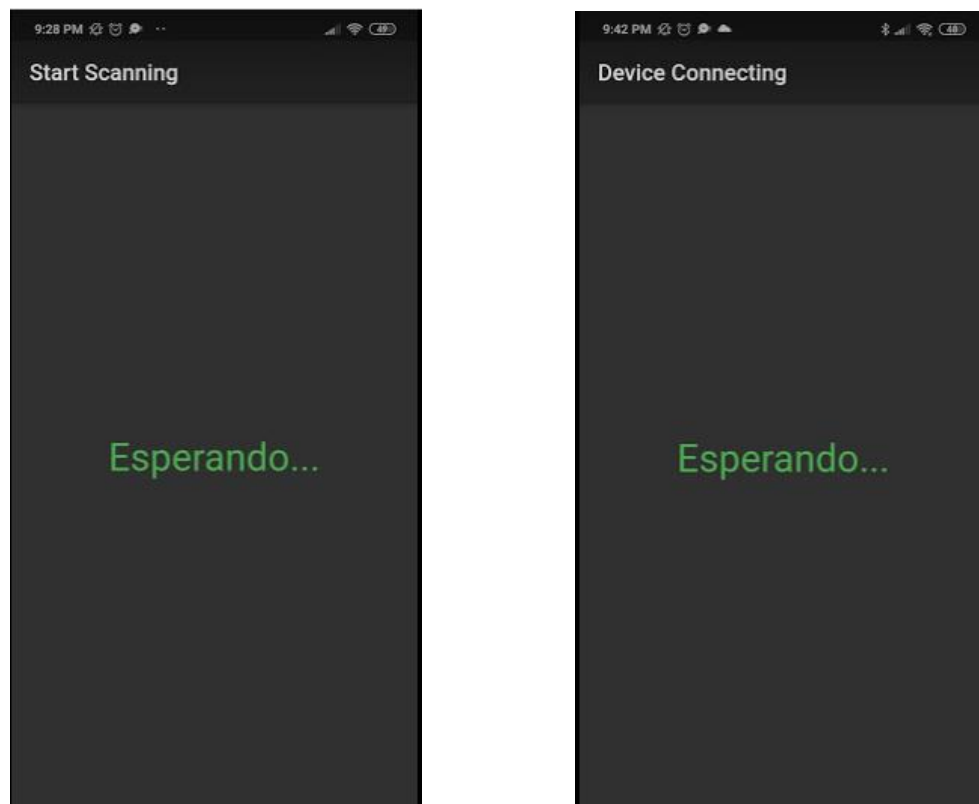


Figura 82. Escaneo y conexión del bluetooth en la app
Fuente: Elaboración propia.

Una vez los dispositivos se conecten, se procede a colocar las credenciales de la red WiFi y esperar la conexión de la misma. El cambio de modo se hace por vía hardware.

A screenshot of a mobile application interface. At the top, the status bar shows the time as 9:42 PM and various icons. The app's title bar reads "All Ready with ESP32 THAT ...". Below the title, there are two input fields: "Wifi Name" and "Wifi Password". A blue "Submit" button is positioned below the "Wifi Password" field. The bottom of the screen features a white navigation bar with three icons: a hamburger menu, a square, and a back arrow.

Figura 83. Credenciales Wifi.
Fuente: Elaboración propia.

A screenshot of the same mobile application interface, but with the input fields filled. The "Wifi Name" field contains the text "AZUMI" and the "Wifi Password" field contains "wifilupita". The blue "Submit" button remains below the password field. The status bar now shows the time as 9:43 PM. The bottom navigation bar is identical to the previous screenshot.

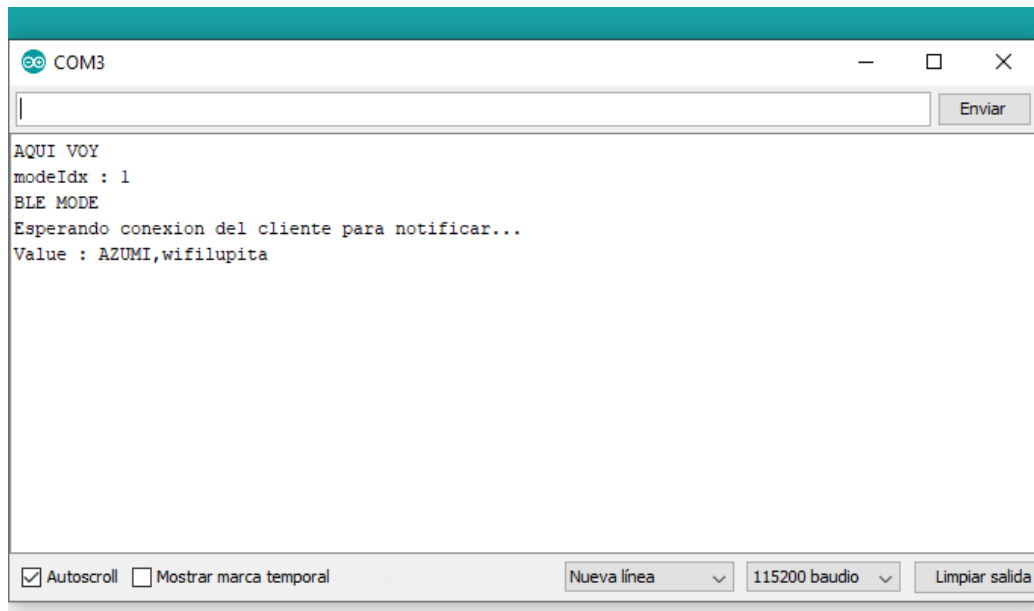


Figura 84. ESP32 leyendo las credenciales recibidas a través de la app.
Fuente: Elaboración propia.

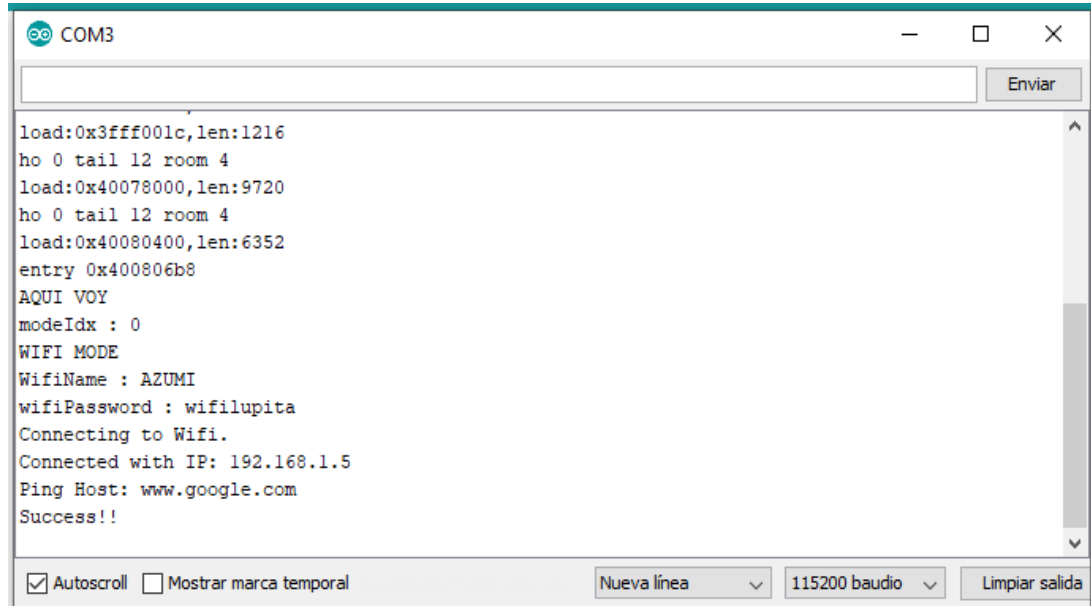


Figura 85. Modo WIFI.
Fuente: Elaboración propia.

Podemos verificar que la conexión tanto en modo bluetooth como en modo WIFI, se establece exitosamente.

Pruebas con Sensores:

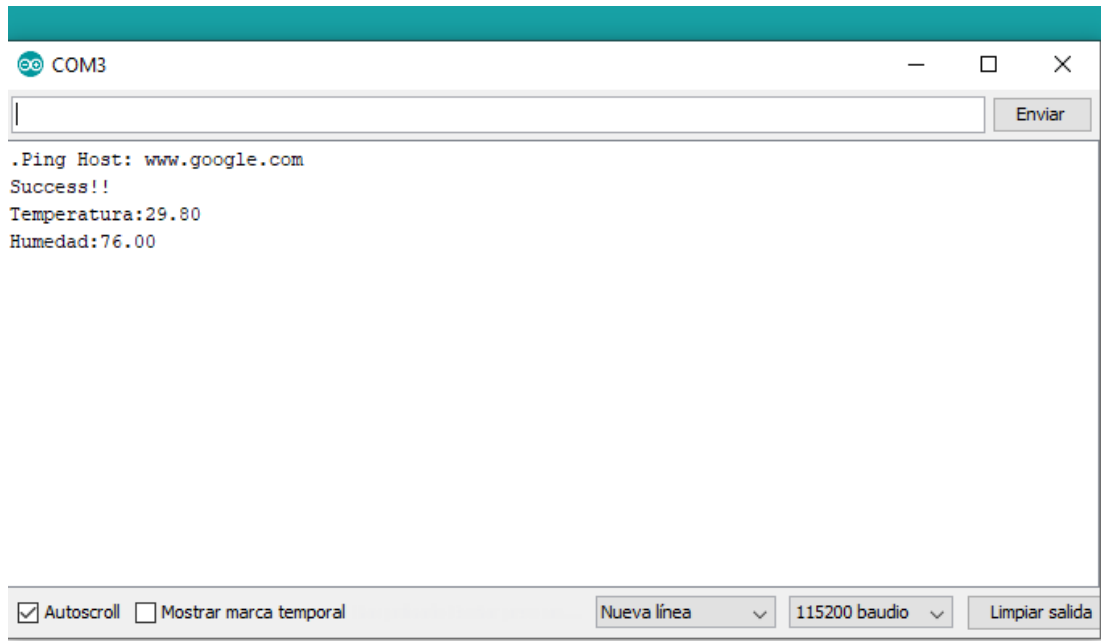


Figura 86. Lectura de datos de temperatura y humedad
Fuente: Elaboración propia

Una vez que los datos se leen en el ESP32, se envían a la base de datos en Firebase. Lógicamente, esta transmisión solo ocurre si el ESP32 está conectado a red WiFi. Véase figura 58

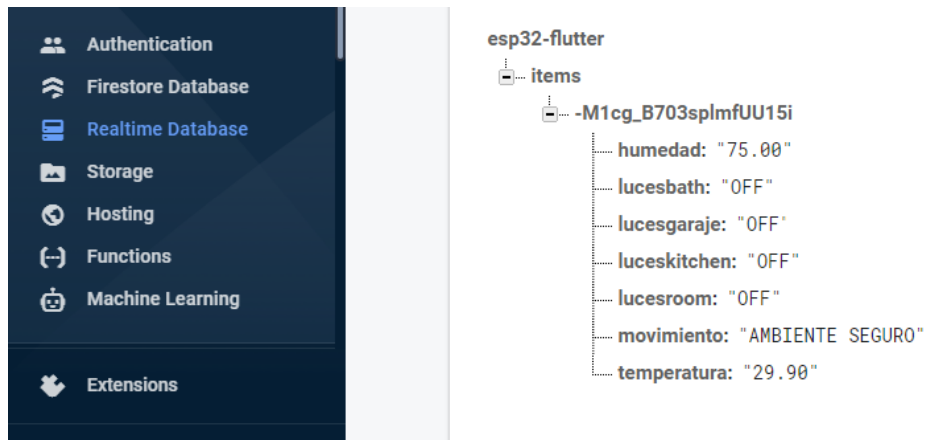


Figura 87. Recepción de datos en Realtime Firebase.
Fuente: Elaboración propia.

Se comprueba que los datos se envían a DB, de manera confiable. Luego Firebase, envía los datos recibidos a la aplicación móvil.

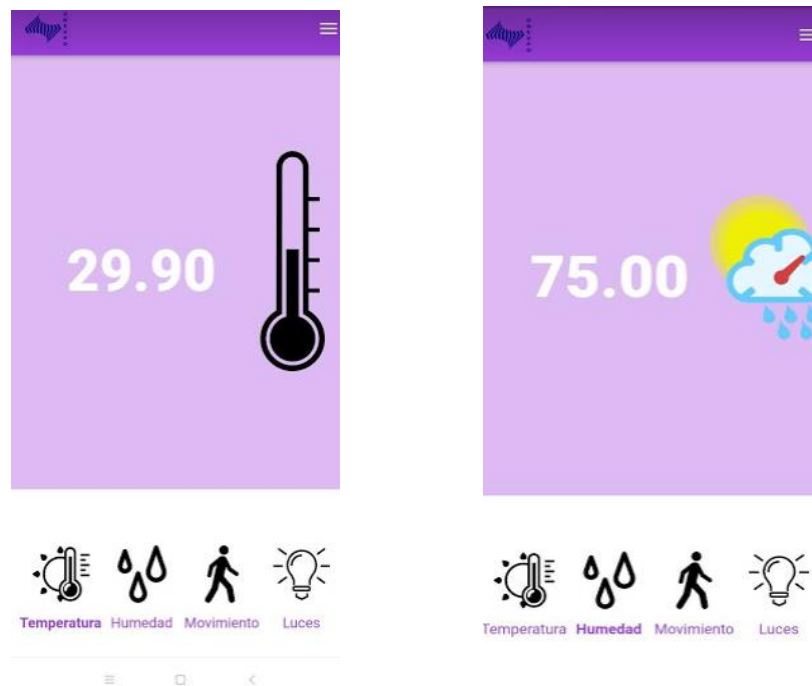


Figura 88. Datos recibidos en App móvil.
Fuente: Elaboración propia.

Para la lectura de los datos del sensor de movimiento, se espera que suceda una interrupción y enviar esta alarma a la base de datos.

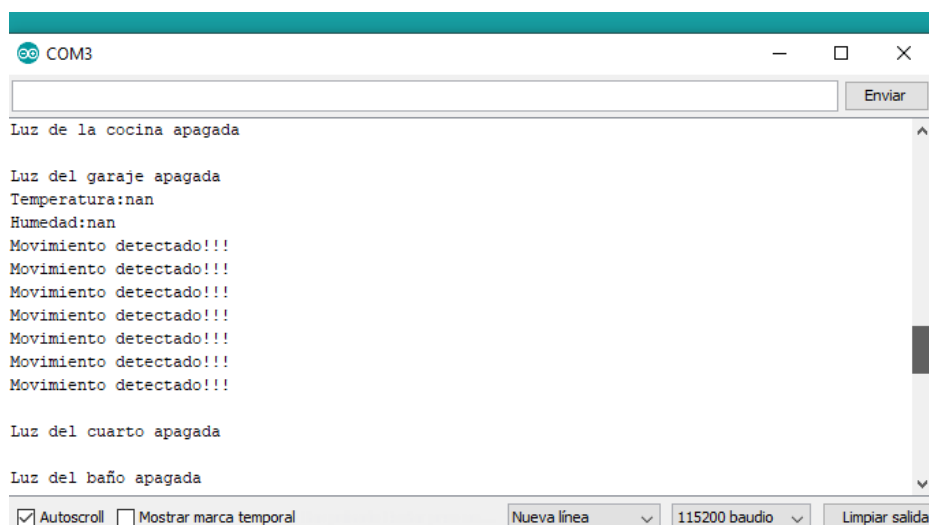


Figura 89. Sensor de movimiento
Fuente: Elaboración propia.

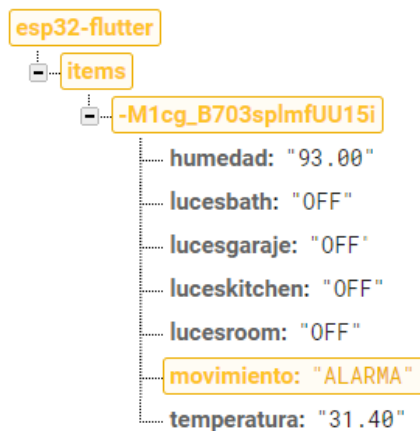


Figura 90. Detección de movimiento en Firebase.
Fuente: Elaboración propia.

La base de datos recibe exitosamente la alarma proveniente del sensor de movimiento.

Esta “alarma” la lee la aplicación móvil y la imprime en pantalla.

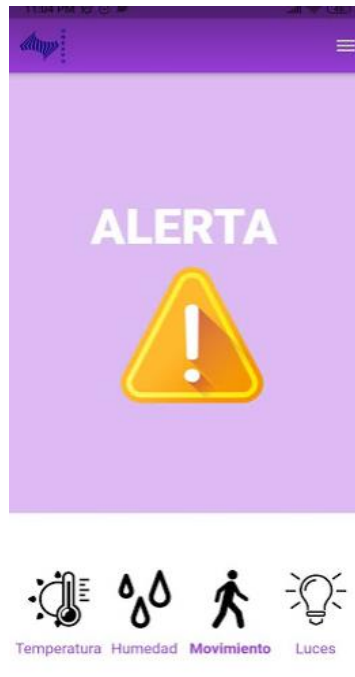


Figura 91. Detección de alarma en app móvil.
Fuente: Elaboración propia.

Mientras no se detecte movimiento, se mostrará un mensaje de “AMBIENTE SEGURO”, indicando que no hay intrusión en el área.



Figura. 92. Sin detección de movimiento en app móvil.
Fuente: Elaboración propia.

Por último, se tiene la lectura desde el ESP32 de los cambios que se realicen en la aplicación móvil para encender o apagar las luces.

La app móvil (según lo que el usuario seleccione) guardará el cambio de estado del área seleccionada, y luego la enviará a la base de datos. Para la prueba se encenderá la luz del baño.



Figura 93. Encendido de luz del baño
Fuente: Elaboración propia

Este cambio de estado se guarda en la base de datos como “ON” en la variable correspondiente. Para la prueba (se seleccionó la luz del baño), los cambios se almacenarán en la variable “Lucesbath”

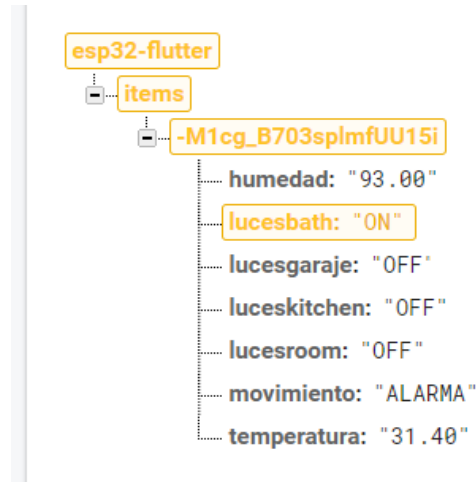


Figura 94. Status de luz del baño en Firebase.
Fuente: Elaboración propia.

Luego que están almacenados en la base de datos, el ESP32 hace un chequeo constante de los status de las variables

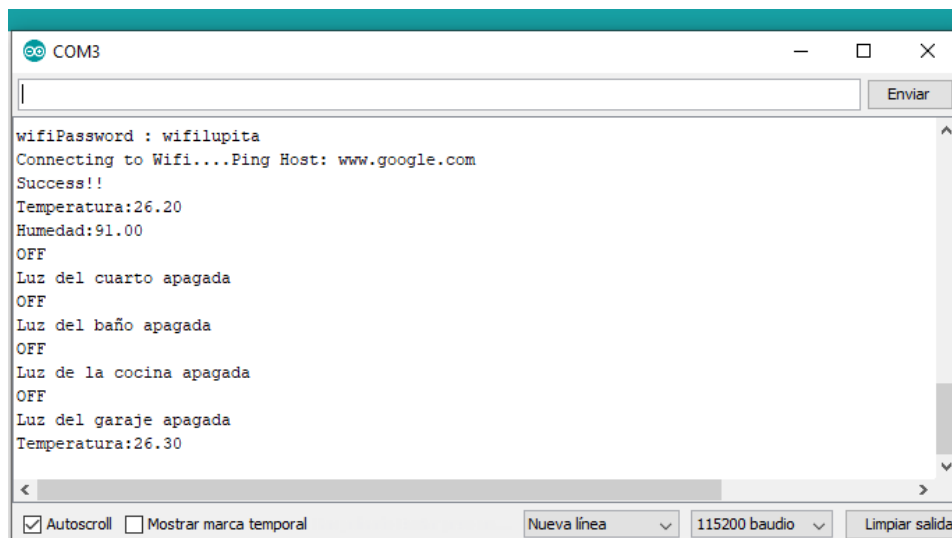


Figura 95. Chequeo del status de las luces en el ESP32.
Fuente: Elaboración propia.

Si ocurre un cambio, el ESP32 lo detectará y ejecutará la acción correspondiente (encender o apagar la luz).

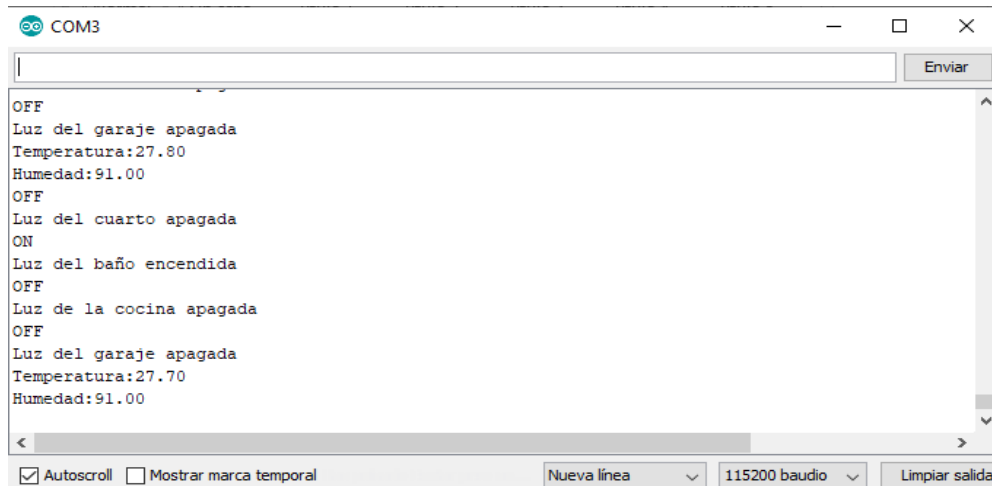


Figura 96. Encendido de luz del baño, ESP32.
Fuente: Elaboración propia.

Este proceso es el mismo para cada ON/OFF de luz en las distintas áreas (cuarto, cocina, garaje). Y logramos comprobar que el sistema cumple nuestros requerimientos.

Finalmente, como plus de la aplicación se colocó un SplashScreen, que tiene la función de mostrar una imagen mientras la aplicación se carga. Esto es comúnmente usado en aplicaciones comerciales.



Figura 97. SplashScreen App móvil.
Fuente: Elaboración propia.

CONCLUSIONES

Finalizado el proceso de recopilación, diseño e implementación del proyecto, se llegó a las siguientes conclusiones:

El ESP32 es un micro-controlador potente que puede ser usado en cualquier proyecto de automatización de alto nivel. Debido a que posee modulo WiFi y Bluetooth, se convierte en una alternativa económica y viable para la elaboración de proyectos futuros.

El sistema de control domotico tiene muchas características y beneficios tanto en el hogar como a nivel industrial. Permite el manejo y control de variables específicas a larga distancia y mantiene un monitoreo constante de las misma, mientras tenga acceso a internet.

El uso de Arduino como entorno principal en el proyecto, fue de gran utilidad debido a la documentación, tutoriales y librerías que posee. Además, es un entorno amigable con el programador y fácil de usar. Arduino IDE contiene librerías usadas por el ESP32 que se comunican con Firebase y permiten la transmisión de datos de manera confiable y segura. Por otro lado, el uso de Firebase ahorró tiempos de producción en el proyecto; ya que, construye bases de datos en tiempo real (como formato Json) de una manera simple y permite la configuración para asociar la base de datos con la aplicación en Flutter.

La aplicación móvil diseñada cumple todos los requerimientos que se necesitaban en el proyecto. Su diseño es minimalista y de fácil uso, con una interfaz gráfica amigable con el usuario. El desarrollo se llevó a cabo en Flutter, el cual contiene una amplia variedad de recursos para realizar una aplicación móvil de manera sencilla y con una excelente calidad.

RECOMENDACIONES

Para la implementación del proyecto a un nivel más avanzado, se recomienda:

Sensar la temperatura y humedad y cuando se supere un valor definido, se encienda un aire acondicionado que climatice el área.

Instalar alarmas sonoras al sensor de movimiento PIR para que emitan alerta en caso de detectarse algún movimiento.

Instalar cámaras de vigilancia para mejorar el sistema de seguridad propuesto.

Conectar al ESP32 mediante hardware o software (según el equipo requiera), equipos electrodomésticos para el control de los mismos desde la app. Esto incrementaría el ahorro de energía, debido a que el usuario decide cuando apagar o encender los equipos.

Configurar cada bombilla como PWM; y que el usuario desde la app y según sus necesidades varíe la luminosidad del área

Programar el ESP32 vía OTA; de esa manera se aprovecharía esa ventaja que ofrece el ESP32 y permitiría la actualización del sistema, aunque ya esté instalado.

REFERENCIAS BIBLIOGRAFICAS

- Barrera, J. H. (21 de 02 de 2008). *Blog Investigacion Holistica*. Recuperado el 14 de 07 de 2019, de <http://investigacionholistica.blogspot.com/2008/02/la-investigacin-proyectiva.html>
- Barrera, J. H. (2010). *Guia para la comprension holistica de la ciencia*. Caracas: Fundacion Sypal.
- Barrera, J. H. (2015). *El proyecto de Investigacion Hurtado*. Caracas: Ediciones Quiron.
- CEDOM. (s.f.). *Asociacion Española de Domotica e Inmotica*. Recuperado el 15 de 06 de 2019, de <http://www.cedom.es/sobre-domotica/que-es-domotica>
- Codigo facilito. (s.f.). Obtenido de <https://codigofacilito.com/cursos/dart>
- G.Arias, F. (2016). *El proyecto de investigacion* . Caracas: Episteme.
- Heredia, D. C. (2012). *Administracion de produccion*. Caracas.
- Herranz, A. B. (2019). *Desarrollo de aplicaciones para IoT con el módulo ESP32*. Madrid, España.
- Huidobro, J. M., & Milan, R. J. (2004). *Domotica. Edificios Inteligentes*. Madrid: Creaciones Copyright.
- Huidobro, J. M., Novel, B., Calafat, C., Suller, E., Escudero, A., & Toledano, J. C. (2007). *La Domotica como solocion de futuro*. Madrid, España. Recuperado el 5 de Julio de 2019, de <https://www.fenercom.com/pdf/publicaciones/la-domotica-como-solucion-de-futuro-fenercom.pdf>
- Lapiente, M. J. (Julio de 29 de 2018). *Hipertexto, el nuevo concepto de documento en la cultura de la imagen*. Obtenido de http://www.hipertexto.info/documentos/h_www.htm
- Mario, T. y. (1997). *El proceso de la investigacion cientifica*. Mexico: Limusa S,A.
- Naylmp Mechatronics. (s.f.). Recuperado el 28 de 07 de 2019, de <https://naylampmechatronics.com/>
- Restrepo, E. A., & Olaya, D. A. (2013). *Desarrollo de un prototipo basado en Blockchain aplicado a la plataforma IoT sobre un sistema embebido* . Bogota, Colombia.

SISTEMAS ESPRESSIF. (2019). Obtenido de <https://www.espressif.com/en/products/hardware/esp32/overview>

Torres, F. J. (2014). *Análisis de diseño de una red domótica para viviendas sociales*. Santiago de Chile.

Wikipedia. (Enero de 2020). Obtenido de Wikipedia: <https://es.wikipedia.org/wiki/Internet>

Wikipedia. (07 de Mayo de 2021). Obtenido de Wikipedia: <https://es.wikipedia.org/wiki/Firebase>

WP, K. (2020). *Geekbot Electronics*. Obtenido de <http://www.geekbotelectronics.com/producto/dht11-sensor-de-temperatura-y-humedad/>

ANEXOS

Código ESP32

```
#include "EEPROM.h"
#include <WiFi.h>
#include <ESP32Ping.h>
#include "FirebaseESP32.h"
#include "DHTesp.h"
#include <stdlib.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#define EEPROM_SIZE 128
#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define timeSeconds 2
#define FIREBASE_HOST "esp32-flutter.firebaseio.com"
#define FIREBASE_AUTH "Sg7E8ECbLbWL8dSP9GMFcZ3udd9KvC4gZ8tBb6lI"

const char* remote_host = "www.google.com";

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;
bool oldDeviceConnected = false;

//Declaracion de objetos
DHTesp dht;
WiFiClient espClient;
FirebaseData firebaseData;
String ruta = "/items/-Mlcg_B703splmfUU15i";

const int modeAddr = 0;
const int wifiAddr = 10;
int modeIdx;

const int PinLDR= 25;
const byte PIRsensor = 26;
int dhtPin = 27;
int Ledroom= 13;
int Ledbath = 14;
int counterInter = 0;
```

```

// Timer: Auxiliary variables}
int contar =0;
unsigned long now = millis();
unsigned long lastTrigger = 0;
boolean startTimer = false;

//-----Estructura para las mediciones respectivas-----
struct sensor {
    int deviceId;
    const char* measurementType;
    float value;
};

//-----Configuracion del Bluetooth-----

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
        BLEDevice::startAdvertising(); };

    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false; };
}

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic){
        std::string value = pCharacteristic->getValue();

        if(value.length() > 0){
            Serial.print("Value : ");
            Serial.println(value.c_str());
            writeString(wifiAddr, value.c_str());} }

    void writeString(int add, String data){
        int _size = data.length();
        for(int i=0; i<_size; i++){
            EEPROM.write(add+i, data[i]);}

        EEPROM.write(add+_size, '\0');
        EEPROM.commit(); };
}

```



```

//----- Interrupcion PIR-----

void IRAM_ATTR detectsMovement() {
    Serial.println("Movimiento detectado!!!");
    startTimer = true;
    lastTrigger = millis();
    contar = 1;
}

void setup() {
    Serial.begin(115200);

    if(!EEPROM.begin(EEPROM_SIZE)){
        delay(1000);}
    delay(3000);
    modeIdx = EEPROM.read(modeAddr); //leo la dir de la Eeprom
    Serial.print("modeIdx : ");
    Serial.println(modeIdx);

    EEPROM.write(modeAddr, modeIdx !=0 ? 0 : 1);
    EEPROM.commit();

    if(modeIdx != 0){
        //BLE MODE
        Serial.println("BLE MODE");
        bleTask();

    }else{
        //WIFI MODE
        Serial.println("WIFI MODE");
        wifiTask();}

    //----- DHT y PIR-----
    dht.setup(dhtPin, DHTesp::DHT11);

    //Configuracion del sensor de movimiento
    pinMode(PIRsensor, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(PIRsensor), detectsMovement, RISING);

    pinMode(Ledroom, OUTPUT);
    digitalWrite(Ledroom, LOW),
    pinMode(Ledbath, OUTPUT);
    digitalWrite(Ledbath, LOW);
}

```

```

void loop() {
  if( contar==1){
    Firebase.setString(firebaseData, ruta + "/movimiento", "ALERTA");
    now = millis();
    if(startTimer && (now - lastTrigger > (timeSeconds*100))) {
      Serial.println("AMBIENTE SEGURO");
      startTimer = false;
    }
    contar =0;
  }
  else {
    Firebase.setString(firebaseData, ruta + "/movimiento", "AMBIENTE SEGURO");
  }
  Temperatura();
  delay(3000);
  Humedad();
  delay(1200);
  lucess();
  delay(500);
}

//-----FUNCIONES-----

void Temperatura()
{
  char temp[20];
  delay(500);
  struct sensor mySensor;
  mySensor.deviceId = 1;
  mySensor.measurementType = "Temperatura";
  mySensor.value = dht.getTemperature();
  Serial.print("Temperatura:");
  Serial.println(mySensor.value);
  dtostrf(mySensor.value,4,2,temp);
  Firebase.setString(firebaseData, ruta + "/temperatura", temp);
  delay(10);
}

void Humedad ()
{
  char hum[20];
  struct sensor mySensor;
  mySensor.deviceId = 2;
  mySensor.measurementType = "Humedad";
  mySensor.value = dht.getHumidity();
  Serial.print("Humedad:");
  Serial.println(mySensor.value);
  dtostrf(mySensor.value,4,2,hum);
  Firebase.setString(firebaseData, ruta + "/humedad",hum);
  delay(500);
}

```

```

void lucess ()
{
    Firebase.getString(firebaseData, ruta + "/lucesroom");
    Serial.println(firebaseData.stringData());
    if (firebaseData.stringData() == "ON"){
        Serial.println("Luz del cuarto encendida");
        digitalWrite(Ledroom, HIGH);
    }
    else {
        Serial.println("Luz del cuarto apagada");
        digitalWrite(Ledroom, LOW);
    }

    Firebase.getString(firebaseData, ruta + "/lucesbath");
    Serial.println(firebaseData.stringData());
    if (firebaseData.stringData() == "ON"){
        Serial.println("Luz del baño encendida");
        digitalWrite(Ledbath, HIGH);
    }

    else {
        Serial.println("Luz del baño apagada");
        digitalWrite(Ledbath, LOW);
    }

    Firebase.getString(firebaseData, ruta + "/luceskitchen");
    Serial.println(firebaseData.stringData());
    if (firebaseData.stringData() == "ON"){
        Serial.println("Luz de la cocina encendida");
    }
    else {
        Serial.println("Luz de la cocina apagada");
    }

    Firebase.getString(firebaseData, ruta + "/lucesgaraje");
    Serial.println(firebaseData.stringData());
    if (firebaseData.stringData() == "ON"){
        Serial.println("Luz del garaje encendida");
    }
    else {
        Serial.println("Luz del garaje apagada");
    }
}

```

```

//-----
void bleTask(){
    // Create the BLE Device
    BLEDevice::init("ESP32 THAT PROJECT");

    // Create the BLE Server
    pServer = BLEDevice::createServer();
    pServer->setCallbacks(new MyServerCallbacks());

    // Create the BLE Service
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // Create a BLE Characteristic
    pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ |
        BLECharacteristic::PROPERTY_WRITE |
        BLECharacteristic::PROPERTY_NOTIFY |
        BLECharacteristic::PROPERTY_INDICATE
    );

    pCharacteristic->setCallbacks(new MyCallbacks());

    // Create a BLE Descriptor
    pCharacteristic->addDescriptor(new BLE2902());

    // Start the service
    pService->start();

    // Start advertising
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    pAdvertising->setScanResponse(false);
    pAdvertising->setMinPreferred(0x0); //
    BLEDevice::startAdvertising();
    Serial.println("Waiting a client connection to notify...");
}

```

```
//-----

void wifiTask() {
    String receivedData;
    receivedData = read_String(wifiAddr);

    if(receivedData.length() > 0){
        String wifiName = getValue(receivedData, ',', 0);
        String wifiPassword = getValue(receivedData, ',', 1);

        if(wifiName.length() > 0 && wifiPassword.length() > 0){
            Serial.print("WifiName : ");
            Serial.println(wifiName);

            Serial.print("wifiPassword : ");
            Serial.println(wifiPassword);

            WiFi.begin(wifiName.c_str(), wifiPassword.c_str());
            Serial.print("Connecting to Wifi");
            while(WiFi.status() != WL_CONNECTED){
                Serial.print(".");
                delay(300);
            }

            Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
            Firebase.reconnectWiFi(true);
            Serial.print("Ping Host: ");
            Serial.println(remote_host);

            if(Ping.ping(remote_host)){
                Serial.println("Success!!");
            }else{
                Serial.println("ERROR!!");
            }
        }
    }
}

String read_String(int add){
    char data[100];
    int len = 0;
    unsigned char k;
    k = EEPROM.read(add);
    while(k != '\0' && len < 500){
        k = EEPROM.read(add+len);
        data[len] = k;
        len++;
    }
    data[len] = '\0';
    return String(data);
}
```

```

String getValue(String data, char separator, int index){
    int found = 0;
    int strIndex[] = {0, -1};
    int maxIndex = data.length()-1;

    for(int i=0; i<=maxIndex && found <=index; i++){
        if(data.charAt(i)==separator || i==maxIndex){
            found++;
            strIndex[0] = strIndex[1]+1;
            strIndex[1] = (i==maxIndex) ? i+1 : i;
        }
    }
    return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}

```