Universidad de Los Andes
La Paz - Bolivia

Facultad de Ingeniería y Tecnología
Carrera de Ingeniería de Sistemas

PROYECTO FINAL

"Desarrollo de sistema multifunción para la creación y control en tiempo real de microclima agrícola en condiciones frías de La Paz"

Materia : **Sistemas De Tiempo Real**

Integrantes : **Alejandro Luis Laura Mendoza**

Luis Fernando Pinto Mamani

Josue Leonardo Manriquez Lopez

Jose Luis Quispe Cruz

Leandro Kenneth Huayna Quiñones

Maribel Esquivel Tarqui

Hilda Alejandra Roldan Yucra

Docente : **Ing. Alejandro Marco Cortes Montes**

La Paz, Septiembre de 2025

INDICE

I. Antecedentes organizacionales

La ciudad de La Paz, Bolivia, se caracteriza por tener un clima frío y variable, lo que representa un desafío para la agricultura, ya que las plantas y cultivos requieren condiciones específicas de temperatura, humedad, riego e iluminación para desarrollarse de manera adecuada. Las variaciones climáticas provocan pérdidas en la productividad y dificultades para mantener la estabilidad de los cultivos. En este contexto, se plantea la necesidad de implementar un sistema tecnológico que permita automatizar y optimizar el control de los factores ambientales.

II. Identificación del problema

En La Paz, las bajas temperaturas y la irregularidad climática afectan el crecimiento de los cultivos, limitando la eficiencia de la producción agrícola. Los agricultores carecen de herramientas accesibles que les permitan controlar el microclima en espacios cerrados, lo que conlleva pérdidas económicas y menor calidad de los productos.

El problema central identificado es: *la falta de un sistema automatizado de riego y control climático multifuncional que garantice condiciones estables y adecuadas para el desarrollo de cultivos en climas fríos.*

Este sistema deberá calcular y mantener un tiempo de clima óptimo (número de horas acumuladas en rangos adecuados de temperatura y humedad, junto con el control de agua y luz), ajustando los parámetros internos para que los cultivos tengan un desarrollo saludable en un entorno cerrado.

III. Objetivos

3.1 Objetivo general

"Diseñar e implementar un sistema de riego multifunción para cultivos, que integre de forma automatizada el control de temperatura, humedad, iluminación y suministro de agua, garantizando un microclima óptimo que mejore la productividad agrícola en condiciones frías de la ciudad de La Paz."

3.2 Objetivos específicos

- Desarrollar el módulo de sensado ambiental: que mida temperatura, humedad del aire y del suelo, luminosidad y calidad del aire, enviando datos en tiempo real al ESP32 y a la base de datos en Firebase.
- Implementar el módulo de control de clima interno: que gestione ventiladores, calefactores, humidificadores y deshumidificadores, ajustando automáticamente las condiciones en base a parámetros óptimos.
- Diseñar el módulo de riego automatizado: capaz de activar electroválvulas o bombas según la humedad del suelo, permitiendo además control manual mediante la aplicación.
- Incorporar el módulo de gestión de iluminación: para controlar paneles LED de espectro completo, ajustando intensidad y horario de acuerdo a las necesidades de cada cultivo.
- Construir el módulo de monitoreo y alertas: que detecte valores fuera de rango y envíe notificaciones push en tiempo real utilizando Firebase Cloud Messaging.
- Desarrollar el módulo de interfaz en tiempo real: (dashboard web/móvil) para la visualización de datos y control del sistema, integrando gráficas y control instantáneo vía Firebase.
- Implementar el módulo de perfiles multicultivo: mediante una base de datos en Firebase con parámetros óptimos para distintos cultivos (ej. tomate, lechuga, fresa), permitiendo al usuario seleccionar y gestionar el cultivo desde la interfaz.

IV. MARCO TEÓRICO — herramientas

- Este apartado explica las herramientas que se usaron y se usarán en el proyecto. Para cada elemento se indica: qué es, por qué se eligió, para qué se utiliza en el sistema y cómo funciona en términos prácticos.

4.1 Concepto base: “tiempo real” y métricas necesarias

Tiempo real: El tiempo real se refiere a la capacidad de un sistema de recibir, procesar y responder a datos y eventos al mismo ritmo que ocurren, o con un retraso mínimo y predecible. Implica una respuesta y actualización de información casi instantánea

En este proyecto tiempo real significa que las lecturas ambientales (temperatura, humedad del aire, humedad del suelo, luz) son transmitidas y disponibles con latencia muy baja (segundos) para que el ESP32 tome decisiones inmediatas (activar riego, iluminación o ventilación) y el usuario reciba notificaciones instantáneas. El sistema usa Firebase Realtime Database para sincronización inmediata y NTP para timestamps exactos.

Métricas térmicas:

Tiempo en rango: número de horas/día en que la temperatura interna está entre T_min y T_max óptimos.

Growing Degree Days (GDD) / degree-hours: métrica acumulativa que estima el avance fenológico. Fórmula simplificada por día:

GDD = max(0, ((Tmax + Tmin)/2) - Tbase)

Para grado-hora (hour-based): sumar por hora max(0, Tactual - Tbase).

Uso en el proyecto: estimar cuántas horas/periodos a temperatura efectiva necesita el cultivo para completar etapas y decidir acciones de calefacción/ventilación.

4.2 ESP32

Qué es: microcontrolador con Wi-Fi y Bluetooth integrados (típicamente dual-core, 3.3 V).

Por qué se usa: combina capacidad de procesamiento, conectividad directa a Firebase (vía Wi-Fi) y múltiples pines ADC/Digital/PWM; ideal para prototipos IoT.

Para qué: leer sensores, ejecutar la lógica de control (histeresis/PID simple), accionar relés/MOSFETs y enviar/recibir datos de la nube.

Cómo funciona (práctico): se programa en Arduino IDE, PlatformIO o ESP-IDF; utiliza ADC para lecturas analógicas (cuidado: ADC no es lineal y requiere calibración), I2C/SPI/UART para periféricos, y Wi-Fi para comunicación con Firebase/REST/MQTT. Alimentación: 3.3 V regulada; nunca exponer a 5 V directamente en GPIO.

Consideraciones: calibrar ADC, usar divisores o amplificadores si el sensor entrega voltajes fuera de rango, proteger pines y aislar cargas inductivas (relés con diodos flyback o SSR).

Figura 1 ESP32 microcontrolador

Fuente: https://dariapp.com/Con-C-mara-M-dulo-WiFi-Bluetooth-Dual-Core-Para-Proyectos-IoT-l-348403

4.3 Sensor DHT11

Qué es: sensor digital económico que mide temperatura y humedad relativa.

Por qué se usa: fácil de integrar para pruebas iniciales y demostraciones; bajo costo.

Para qué: medir temperatura y humedad del aire dentro del sistema cerrado para decidir calefacción/ventilación.

Cómo funciona: comunicación por un único pin digital (protocolo propietario). Rango y precisión limitados (temperatura ~0-50 °C, precisión reducida, humedad con ±4-5 %), muestreo lento.

Recomendación: para un sistema más robusto considerar DHT22, SHT31 o BME280 (mayor precisión, mejor estabilidad y calibración).

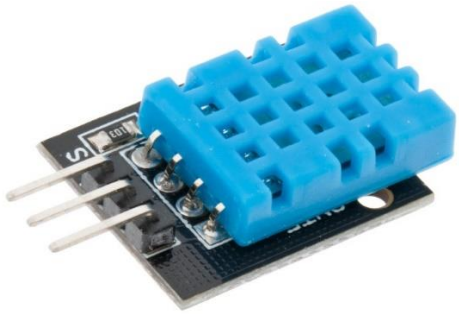

Figura 2 Sensor DTH11

Fuente: https://marboltec.com/producto/modulo-sensor-de-humedad-y-temperatura-dht11/

4.4 DHT22 (sensor temperatura y humedad) con LED indicador

Qué es: Es un sensor digital de temperatura y humedad (mayor precisión que DHT11).

Por qué se usa : Se usa para medir temperatura y humedad del microclima para decidir riego, calefacción, iluminación y ventilación.

Cómo funciona / conectar:

- Pines: VCC (3.3–6V), GND, DATA.
- Necesita una resistencia pull-up en DATA (4.7k–10k).

LED indicador: Este se conecta desde el pin de la placa → resistor 220Ω → LED → GND (o usa transistor si quieres encender con 12V).

Figura 3 DHT22 (sensor temperatura y humedad) con LED indicador

Fuente: https://electronicacaribe.com/product/dht22-sensor-de-temperatura-y-humedad-relativa/

4.5 Sensor de humedad de suelo FC-28

Qué es: sonda resistiva que mide la conductividad eléctrica del suelo como proxy de su humedad.

Por qué se usa: barato y fácil de conseguir; útil para prototipos iniciales.

Para qué: decidir apertura de electroválvulas o tiempo de bombeo según humedad detectada.

Cómo funciona: la sonda forma parte de un divisor de tensión y el ESP32 lee el valor analógico; valores altos/bajos indican más/menos agua.

Limitaciones y recomendaciones: la FC-28 es propensa a corrosión y lecturas influenciadas por sales del suelo. Es recomendable:

Calibrarla mediante pruebas gravimétricas (peso de muestra húmeda vs seca) para obtener umbrales reales.

Considerar reemplazo por sensores capacitivos (más fiables a largo plazo) en versiones posteriores del proyecto.

Figura 4 Sensor de humedad de suelo FC-28

Fuente: https://sgemx.com/fc-28-modulo-sensor-de-humedad-del-suelo-higrometro-compatible-con-arduino?srsltid=AfmBOorCH9BZP1NZtDHbv-GFA_ftznsw9tDgH9FQEjynccHsQktzTzjW

4.6 Sensor de Luz BH1750

Qué es: sensor digital económico que mide la intensidad de luz ambiental en lux.

Por qué se usa: fácil de integrar con microcontroladores mediante I²C; lectura rápida y precisa de luz ambiental.

Para qué: medir la luz disponible en el huerto para decidir encendido de iluminación artificial o registro de condiciones de crecimiento.

Cómo funciona: comunicación digital mediante protocolo I²C (dos pines: SDA y SCL); fotodiodo capta la luz, circuito interno convierte a valor digital; rango 1–65535 lux, alta precisión, bajo consumo.

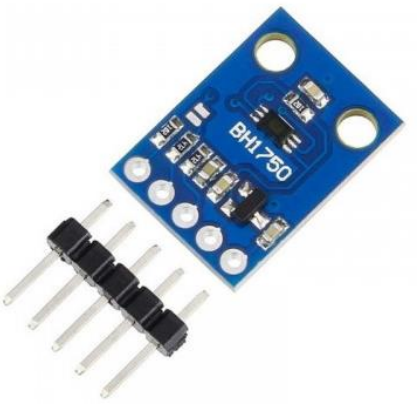

Figura 5 Sensor de luz BH1750

Fuente: https://tienda.sawers.com.bo/bh1750-modulo-iluminacion-de-luz-para-arduino-?srsltid=AfmBOoplY38kleVd-DyIk3H6PKGHKUjKrlzwwvwOIz5sKLTaL4LS9ap3

4.7 Cable jumpers de 20 cm

Qué es: Un jumper es un pequeño conector que sirve para unir o separar dos pines en un circuito. Permite configurar placas, módulos o sensores sin necesidad de soldadura.

Por qué se usa: Se utiliza porque es económico, fácil de manejar y reutilizable. Facilita cambiar la configuración de un dispositivo de manera rápida y segura.

Para qué: Los jumpers se usan para activar sensores, seleccionar modos de operación o ajustar funciones del sistema sin modificar el cableado.

Cómo funciona: Al colocarlo sobre dos pines, el contacto interno cierra el circuito y deja pasar la señal; al retirarlo, el circuito queda abierto.

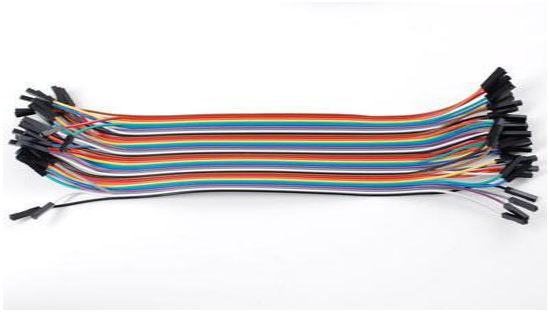

Figura 6 Cable Jumpers

Fuente: https://marboltec.com/producto/cable-jumper-dupont-20cm-varios-tipos/

4.8 Protoboard

Qué es: El protoboard es una placa de pruebas sin soldadura que permite conectar componentes electrónicos temporalmente. Está formada por filas de contactos metálicos que facilitan la creación de circuitos de forma rápida y segura.

Por qué se usa: Se utiliza porque permite armar y modificar circuitos fácilmente, sin necesidad de soldar, lo que es ideal para experimentos, prototipos y pruebas de proyectos electrónicos.

Para qué: El protoboard se usa para montar y probar sensores, módulos y microcontroladores antes de hacer conexiones permanentes o diseñar la placa final.

Cómo funciona: Los componentes se insertan en los agujeros del protoboard, y los contactos internos conectan eléctricamente los pines según la disposición de las filas y columnas, permitiendo que la corriente fluya por el circuito temporal.

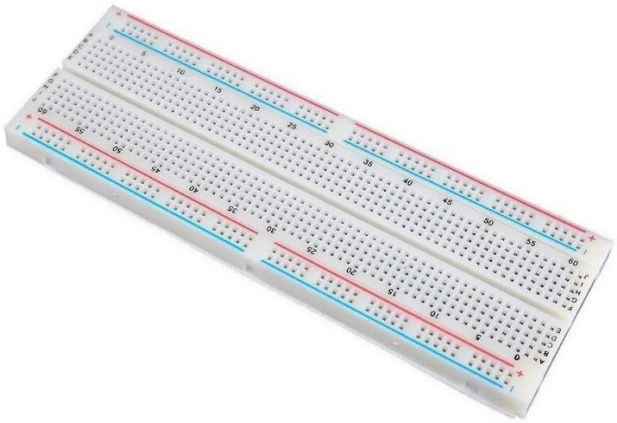

Figura 7 Protoboard o placa de pruebas

Fuente: https://marboltec.com/producto/protoboard-solderless-de-830-pts/

4.9 Cable USB a Tipo-C/ Micro-USB

Qué es: Es un cable de alimentación y datos entre PC y la placa (ESP32, Arduino, etc.).

Por qué se usa: Para programar la placa, depurar datos en Serial Monitor y alimentar temporalmente la placa en pruebas.

Cómo usar: Conecta al puerto USB de la placa para flashear el firmware. Para alimentación definitiva, usa la fuente 12V + regulador (ver abajo).

Figura 8 Cable USB a Tipo-C/ Micro-USB

Fuente: https://pvl.com.bo/producto/cable-2-en-1-tipo-c-y-micro-usb-1m-cb4056gy/

4.10 Fuente de poder (Switch Power) 12V

Qué es: Es una fuente DC regulada 12V.

Por qué se usa : Se usa para alimentar bombas, válvulas, controladores de relé o drivers que requieren 12V.

Cómo funciona : Mantén GND común entre la fuente y la placa de control.

NO alimentar directamente sensores/placa que piden 5V/3.3V. Usa un regulador (step-down / buck) o la entrada VIN adecuada.

Figura 9 fuente de poder

Fuente: https://tienda.sawers.com.bo/fuente-switching-360w-12v-20a-?srsltid=AfmBOooZIG1aeLE9ufePoq0fR705hwzjuSEFg51pqLWakmJQ6QMXv4h9

4.11 Resistencias

Qué es: Son componentes electrónicos que limitan la corriente y protegen los sensores y actuadores del sistema contra sobrecargas.

Para qué: limitar corriente de LEDs, pull-ups, divisores de tensión.

Valores típicos: LED indicador → 220 Ω; pull-up para DHT22 data → 4.7 kΩ – 10 kΩ; sensores I²C no necesitan resistors si el módulo ya trae pull-ups.

Figura 10 Resistencias

Fuente: https://ecrobotics.com.bo/producto/resistencias-1-4w/

4.12 Tiras de Led

Qué es: Una tira de LED es un conjunto de pequeños diodos emisores de luz montados en una cinta flexible, que permite iluminar espacios de manera uniforme y controlable.

Por qué se usa: Se utilizan porque son económicas, consumen poca energía, se pueden cortar a medida y permiten controlar el color y la intensidad de la luz fácilmente.

Para qué: Las tiras de LED se usan para proporcionar iluminación artificial a las plantas, especialmente cuando la luz natural no es suficiente para su crecimiento.

Cómo funciona: Se conectan a una fuente de energía y, en muchos casos, a un controlador que permite encenderlas, apagarlas o regular su intensidad; los LEDs emiten luz cuando pasa corriente por ellos.

Figura 11 Tira de led

Fuente: https://lacarolinasanluis.com.ar/1892531/Led-Para-5050-De-10-Mm-Y-4-Pines

4.13 Plataforma en la nube: Firebase

Qué es: conjunto de servicios de Google para base de datos en tiempo real, autenticación y notificaciones.

Por qué se usa: permite replicar datos en tiempo real entre el ESP32 y las apps web/móvil sin montar servidor propio; FCM permite push notifications.

Para qué: almacenar lecturas, mostrar dashboards en tiempo real y enviar alertas cuando valores estén fuera de rango.

Cómo funciona (práctico): el ESP32 se conecta por Wi-Fi y realiza peticiones HTTP/REST o usa la librería Firebase para publicar datos; la app/web se suscribe a los nodos para recibir cambios instantáneos.

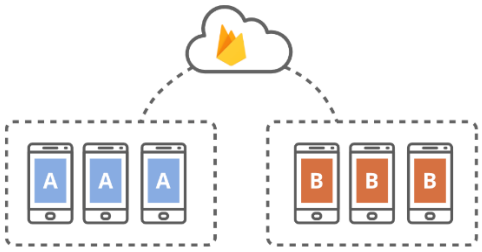

Figura 12 Firebase plataforma en la nube

Fuente: https://www.geeksforgeeks.org/firebase/firebase-introduction/

4.14 Lenguaje de programación C++ en Arduino IDE

Qué es: El Arduino IDE utiliza un lenguaje de programación basado en C++, adaptado y simplificado para controlar microcontroladores. Este lenguaje permite trabajar de forma directa con los pines de entrada y salida, facilitando la interacción entre el hardware y el software del sistema.

Por qué se usa: Se eligió este lenguaje porque es el más adecuado para programar placas Arduino y similares, ya que ofrece gran compatibilidad con librerías y módulos electrónicos como sensores, tiras de LED y actuadores. Además, es sencillo de aprender y muy utilizado en proyectos de automatización y prototipado.

Para qué: En el proyecto el C++ se usa para programar la lógica que controla los sensores (como el BH1750 o el de temperatura y humedad), interpretar los datos que generan

y activar actuadores como tiras de LED o ventiladores, asegurando un control automático del ambiente para el crecimiento de las plantas.

Cómo funciona: El programa en Arduino C++ se estructura principalmente en dos funciones: setup(), que se ejecuta una sola vez al inicio para configurar los sensores y pines, y loop(), que se repite continuamente mientras el sistema está encendido. De esta forma, el microcontrolador recibe datos, toma decisiones y ejecuta acciones de manera automática.

4.15 Librerias usadas

4.15.1 Library.properties:

Qué es: La Firebase Arduino Client Library for ESP8266 and ESP32 es una librería desarrollada por Mobizt que permite la comunicación directa entre las placas de desarrollo (ESP32/ESP8266) y la plataforma Google Firebase. Su versión utilizada en el proyecto es la 4.4.17.

Por qué se usa: Se utiliza porque ofrece compatibilidad nativa con ESP32, es gratuita, fácil de integrar en el Arduino IDE y permite conectar el sistema del huerto automatizado con la nube de Firebase para almacenar y consultar datos en tiempo real.

Para qué: En el proyecto, la librería se emplea para enviar datos de sensores (como temperatura, humedad o luz) hacia una base de datos en Firebase, y también para recibir instrucciones desde la nube, como activar ventilación o iluminación. De esta manera, se consigue un huerto controlado de forma remota y accesible desde cualquier dispositivo conectado a internet.

Cómo funciona: La librería implementa clientes de comunicación (WiFiClient, EthernetClient, GSMClient, etc.) que permiten a la ESP32 conectarse a los servicios de Firebase. A través de funciones específicas, el microcontrolador puede escribir en bases de

datos en tiempo real, leer información de Firestore, almacenar imágenes en Firebase Storage o interactuar con funciones en la nube. Todo esto se gestiona mediante comandos en C++ dentro del Arduino IDE.

4.15.2 Library.json :

Qué es: Es un archivo de configuración en formato JSON que describe la librería Firebase Arduino Client Library for ESP8266 and ESP32, versión (4.4.17), autor (Mobizt), palabras clave, descripción y compatibilidad con diferentes placas y frameworks.

Por qué se usa: Este archivo es necesario para que el Arduino IDE y otros entornos reconozcan la librería, sus dependencias y los dispositivos con los que puede trabajar. Facilita la instalación y actualización desde el gestor de librerías sin necesidad de configuraciones manuales.

Para qué: En el proyecto del huerto automatizado, este JSON asegura que la librería de Firebase se pueda instalar correctamente y sea compatible con la placa ESP32 que usamos. Gracias a él, los sensores y actuadores pueden comunicarse con Firebase sin errores de integración.

Cómo funciona: Cuando el IDE o un gestor de dependencias revisa las librerías instaladas, lee este archivo JSON para identificar la librería, mostrar su información al usuario y verificar que funcione en la plataforma usada (ESP32, ESP8266, etc.). En pocas palabras, actúa como una tarjeta de identidad de la librería.

4.15.3 Librería BH1750

Qué es: Es una librería para Arduino que permite el uso del sensor digital BH1750, especializado en medir la intensidad de luz ambiental en lux.

Por qué se usa: Porque simplifica la programación del sensor, evitando cálculos complejos y ofreciendo funciones listas para obtener lecturas precisas de luz.

Para qué: En el proyecto se usa para monitorear la cantidad de luz que reciben las plantas y decidir si es necesario encender la iluminación artificial (tiras LED).

Cómo funciona: La librería utiliza el protocolo I²C para comunicarse con el sensor. A través de funciones en C++, se leen los valores de luz en lux y se envían al microcontrolador para tomar decisiones automáticas.

4.15.4 DHT Sensor Library – Adafruit:

Qué es: Es una librería desarrollada por Adafruit que permite el uso de sensores digitales de temperatura y humedad como el DHT11 y DHT22.

Por qué se usa: Porque facilita la obtención de datos ambientales sin tener que implementar manualmente el protocolo digital propietario del sensor.

Para qué: En el proyecto se utiliza para medir la temperatura y humedad relativa dentro del huerto, lo que permite decidir acciones como activar ventiladores, calefacción o riego.

Cómo funciona: La librería se comunica con el sensor a través de un pin digital, procesa la señal recibida y entrega directamente los valores de temperatura en °C y de humedad en %.

4.15.5 NTPClient

Qué es: librería que obtiene la hora desde servidores NTP (pool.ntp.org).

Por qué y para qué: para timestamp en registros de riego, programar riegos nocturnos diurnos, y correlacionar lecturas (importante en La Paz por horarios de sol/temperatura).

Cómo funciona (uso típico):

Conecta Wi-Fi → crea cliente UDP → consulta servidor NTP → obtiene epoch → aplicas offset de zona (La Paz = UTC−4).

4.15.6 addons/TokenHelper.h

Qué es: Es un archivo auxiliar (de algunas bibliotecas de Firebase) que contiene callbacks / funciones para manejar tokens de autenticación.

Por qué y para qué: Se usa cuando tu dispositivo se conecta a Firebase RTDB/Firestore, el token de acceso puede caducar; TokenHelper facilita depurar renovaciones, errores y estados del token.

Cómo funciona: Se incluye en el sketch y la librería de Firebase llama a sus funciones para loguear/mostrar en Serial el estado del token.

4.15.7 addons/RTDBHelper.h

Qué es: Helper para mostrar/depurar operaciones con la base de datos en tiempo real (RTDB).

Por qué y para qué: Para ver en Serial qué se escribió/leé, status codes y errores — útil para pruebas de riego automático y telemetría.

Cómo usar: Se usa como #include "addons/RTDBHelper.h" y pasar callbacks a la librería Firebase.

Si usas la librería Firebase ESP Client (ESP32/ESP8266), estas dos helpers suelen venir en la carpeta addons/ del ejemplo. Úsalas durante pruebas y quítalas en producción si no quieres logs.

4.15.8 Wire.h

Qué es: Es una librería I²C en Arduino.

Por qué se usa: Para sensores de luz (BH1750), pantallas OLED o expansores I/O usan I²C.

Cómo funciona: Wire.begin() en setup; luego Wire.requestFrom() / Wire.read() para comunicarse.

4.15.9 WiFi (librería nativa del ESP32)

Qué es: Es la librería oficial incluida en el gestor de tarjetas de Arduino para el ESP32, encargada de establecer la conexión del microcontrolador con una red WiFi.

Por qué se usa: Porque es imprescindible para conectar el sistema del huerto automatizado a internet, lo cual permite enviar y recibir datos desde Firebase.

Para qué: Se utiliza para mantener la comunicación entre los sensores/actuadores del proyecto y la nube, garantizando que la información esté disponible en tiempo real desde cualquier dispositivo.

Cómo funciona: La librería se inicializa con el nombre (SSID) y la contraseña de la red. Una vez conectada, abre un canal de comunicación que es aprovechado por otras librerías como la de Firebase para transferir datos.

4.15.10 Gestor de placas ESP32 – Espressif Systems

Qué es: Es un conjunto de archivos y configuraciones que permite al Arduino IDE reconocer y programar placas ESP32. Incluye compiladores, librerías básicas, definiciones de pines y herramientas necesarias para trabajar con este módulo.

Por qué se usa: Porque sin instalarlo, el Arduino IDE no puede compilar ni cargar programas en la placa ESP32. Facilita la integración del hardware con el software de manera sencilla y confiable.

Para qué: Se utiliza en el proyecto para programar la placa ESP32, de modo que pueda interactuar con sensores, actuadores y librerías (como Firebase o BH1750), asegurando que todas las funciones del sistema funcionen correctamente.

Cómo funciona: Se instala a través del Board Manager del Arduino IDE. Una vez instalado, el IDE reconoce la ESP32 como una placa disponible, habilitando la selección de modelo específico, velocidad de carga y otras configuraciones necesarias para compilar y subir el código correctamente.

4.16 Diagramas UML que se utilizaran

4.16.1 Diagrama de casos de uso

Qué es: Es un diagrama que describe qué hace el sistema desde el punto de vista del usuario o actores externos. No muestra cómo está implementado, sino qué servicios ofrece.

Por qué se usa: Porque es la mejor forma de comunicarse con personas que no son técnicas (clientes, usuarios, gerentes), ya que muestra funciones del sistema de manera simple.

Para qué sirve: Para identificar claramente los requisitos funcionales del sistema, los roles que interactúan con él y las funcionalidades que esperan.

Cómo funciona:

- Los actores (usuarios, sistemas externos, dispositivos) se dibujan como monigotes.
- Los casos de uso son óvalos con el nombre de la acción (“Registrar usuario”, “Iniciar sesión”).
- Se trazan líneas que conectan actores con casos de uso para mostrar la interacción.

- Se pueden usar relaciones como include (cuando un caso de uso siempre incluye a otro) o extend (cuando a veces se agrega una funcionalidad extra).

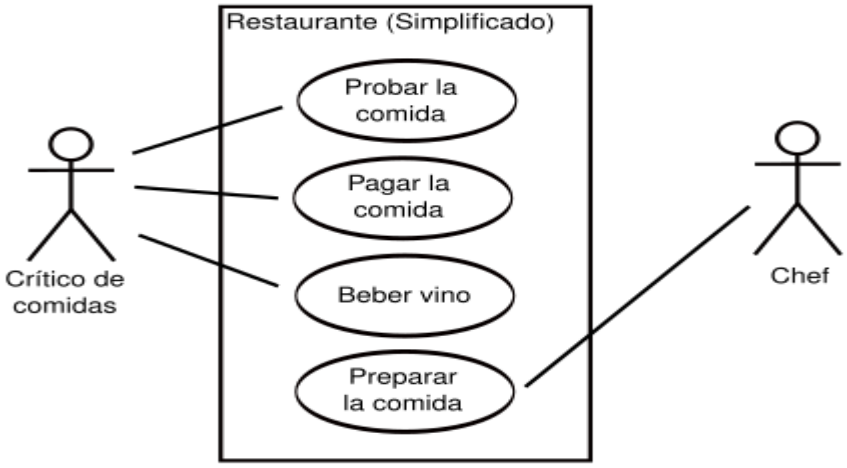

Figura 13 Ejemplo de Diagrama de clases

Fuente: https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

4.16.2 Diagrama de clases

Qué es: Es el diagrama que representa la estructura estática del sistema. Muestra las clases (objetos), sus atributos (datos), sus métodos (funciones) y las relaciones entre ellas.

Por qué se usa: Porque es la base de la programación orientada a objetos, y permite pasar del análisis al diseño detallado del software.

Para qué sirve: Para definir cómo se organiza la información y las funcionalidades dentro del sistema, y cómo se relacionan los diferentes objetos.

Cómo funciona:

- Una clase se dibuja como un rectángulo dividido en tres partes:
    1) Nombre de la clase
    2) Atributos
    3) Métodos

- Se establecen relaciones como:
  - *Herencia* (flecha con triángulo vacío) → una clase hija hereda de otra.
  - *Asociación* (línea simple) → dos clases se relacionan.
  - *Composición* (rombo negro) → una clase no puede existir sin la otra.
  - *Agregación* (rombo blanco) → una clase depende de otra pero puede existir por separado.

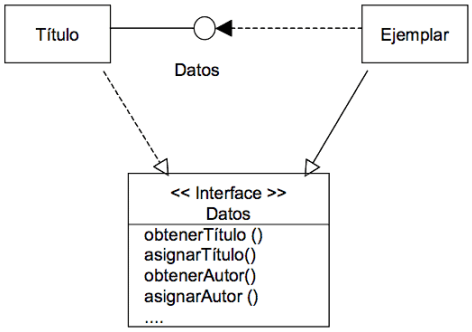

Figura 14 Ejemplo de Diagrama de Clases

Fuente: https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-clases/

4.16.3 Diagrama de Secuencia

Qué es: Es un diagrama dinámico que muestra cómo interactúan los objetos entre sí en un escenario concreto, a lo largo del tiempo.

Por qué se usa: Porque permite visualizar el orden en que suceden las operaciones y cómo se comunican los componentes en un proceso.

Para qué sirve: Para entender y documentar el flujo de mensajes que se intercambian en un caso de uso, facilitando la implementación posterior.

Cómo funciona:

- Los objetos o actores se colocan en la parte superior.
- Desde cada objeto baja una línea vertical llamada línea de vida.

- Se dibujan flechas horizontales de un objeto a otro para indicar los mensajes o llamadas de métodos.
- El orden de arriba hacia abajo representa el paso del tiempo.

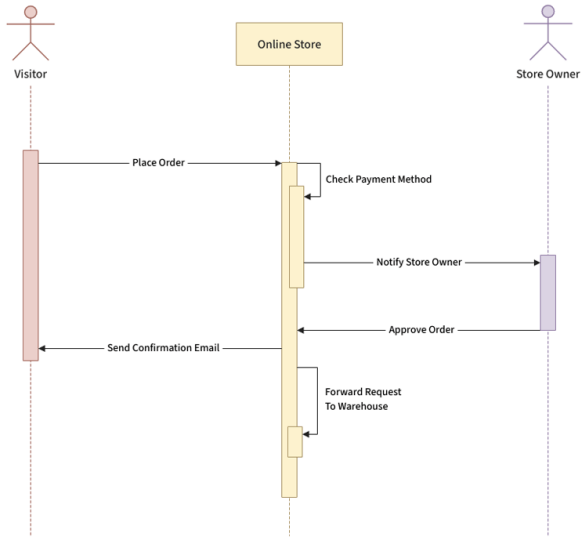

Figura 15 Ejemplo de diagrama de secuencia

Fuente: https://moqups.com/es/templates/mapping-and-diagramming/uml-diagrams/sequence-diagram/

4.16.4 Diagrama de Actividades

Qué es: Es un diagrama que representa el flujo de trabajo o las actividades que se ejecutan dentro de un proceso del sistema.

Por qué se usa: Porque ayuda a visualizar claramente la lógica de los procesos, decisiones, bucles o tareas que pueden realizarse en paralelo.

Para qué sirve: Para describir paso a paso cómo se ejecutan las actividades, lo que facilita analizar la lógica de negocio antes de programar.

Cómo funciona:

- Se inicia con un nodo inicial (círculo negro).
- Se representan las actividades con rectángulos redondeados.

- Los puntos de decisión se muestran con rombos.
- Se conectan con flechas que indican el flujo del proceso.
- Termina con un nodo final (círculo negro con borde doble).

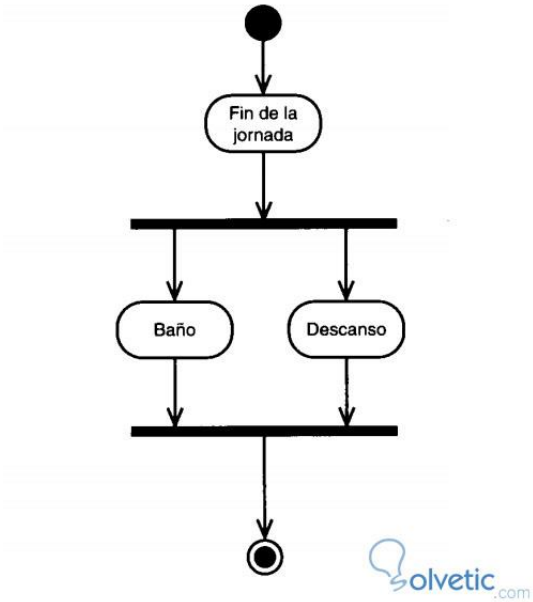

Figura 16 Ejemplo de Diagrama de Actividades

Fuente: https://www.solvetic.com/tutoriales/article/517-uml-diagramas-de-actividades-avanzado/

4.16.5 Diagrama de Componentes

Qué es: Es un diagrama que muestra los componentes lógicos o físicos que forman parte de un sistema de software, y cómo se conectan entre ellos.

Por qué se usa: Porque permite visualizar la arquitectura del software en módulos, lo que facilita el mantenimiento, escalabilidad y reutilización de código.

Para qué sirve: Para identificar qué partes conforman el sistema (por ejemplo: frontend, backend, base de datos, APIs externas) y cómo se relacionan.

Cómo funciona:

- Cada componente se representa como un rectángulo con pestañitas.
- Se conectan con líneas que muestran las dependencias o relaciones.

- Se puede especificar qué interfaces ofrece o requiere cada componente.

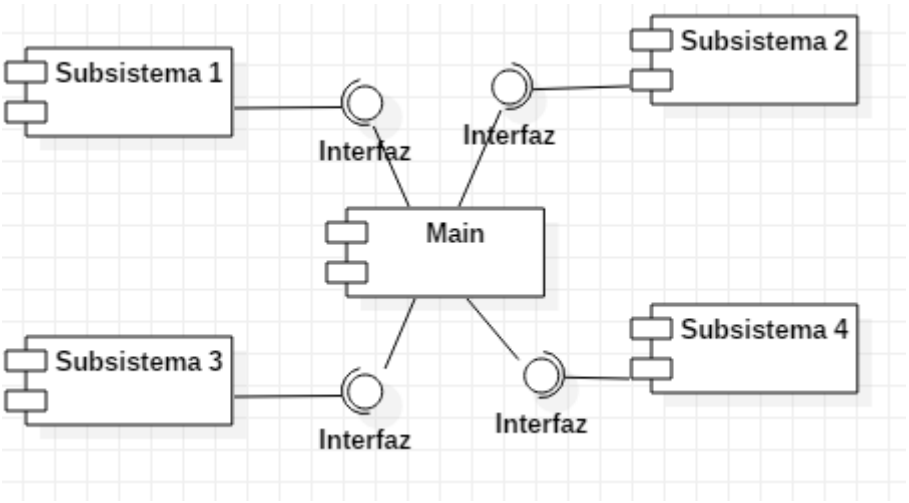

Figura 17 Ejemplo de Diagrama de Componentes

Fuente: https://diagramasuml.com/componentes/

4.16.6 Diagrama de Despliegue

Qué es: Es un diagrama que representa la distribución física del sistema en los dispositivos o nodos de hardware donde se ejecuta.

Por qué se usa: Porque ayuda a entender cómo se despliega el software en la infraestructura física (servidores, computadoras, móviles, sensores IoT).

Para qué sirve: Para planificar la arquitectura física del sistema: en qué máquina corre la base de datos, dónde está el servidor, cómo se conectan los dispositivos.

Cómo funciona:

- Los nodos físicos (servidores, PCs, móviles, sensores) se representan como cubos 3D.
- Dentro de cada nodo se colocan los artefactos o aplicaciones que se ejecutan en él.
- Se muestran las conexiones de red o comunicación entre nodos.

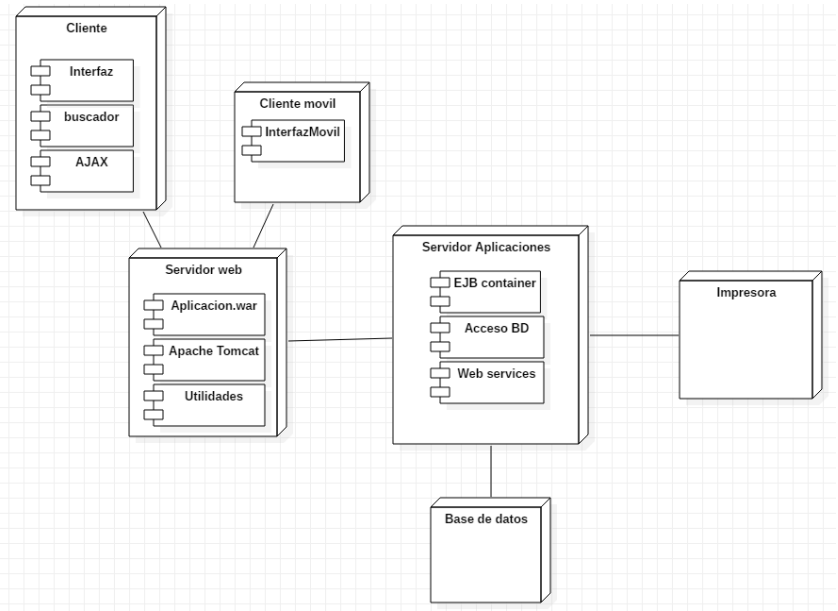

Figura 18 Ejemplo de Diagrama de Despliegue

Fuente: https://diagramasuml.com/despliegue/

4.17 Requisitos ambientales (apio, perejil, hierba buena)

4.17.1 Apio (Apium graveolens)

- Temperatura óptima: **15–21 °C** (no tolera heladas).
- Humedad relativa del aire: **60–80%**.
- Iluminación: luz moderada-alta; 10,000–30,000 lux (LED, 12–16 h/día según etapa).
- Humedad del suelo: mantener suelo constantemente húmedo (evitar secado — humedad del suelo > 60% ideal).

Notas: necesita riego frecuente en pequeñas cantidades; buen drenaje a la vez que retención.

4.17.2 Perejil (Petroselinum crispum)

- Temperatura óptima: **15–24 °C**.
- Humedad relativa: **50–70%**.
- Iluminación: luz moderada; 8–12 h/día, ~8,000–20,000 lux.
- Humedad del suelo: mantener ligeramente húmedo, riegos regulares (40–60% de humedad de suelo).

Notas: evita encharcar; tolera algo de sombra.

4.17.3 Hierba buena / Menta (Mentha spp.)

- Temperatura óptima: **15–22 °C**.
- Humedad relativa: **50–80%**.
- Iluminación: luz alta moderada; 8–12 h/día, 10,000–25,000 lux.
- Humedad del suelo: prefiere suelo húmedo pero bien drenado (45–65%).

Notas: crecimiento vigoroso, controla raíces si es en maceta.

V. MARCO PRACTICO

- En este apartado se describe la implementación concreta del sistema, incluyendo los dispositivos electrónicos, sensores, actuadores y la programación necesaria para el control automático de las condiciones del huerto, con el objetivo de optimizar el crecimiento de las plantas de manera eficiente y sostenible.

5.1 Construcción del huerto automatizado

- **Diseño inicial en 3D (planificación):**

Primero se realizó un modelo digital en 3D (como se ve en la primera imagen) para visualizar cómo quedaría el huerto con sus divisiones y patas de soporte. Este modelo permitió definir las medidas y la estructura antes de cortar la madera.

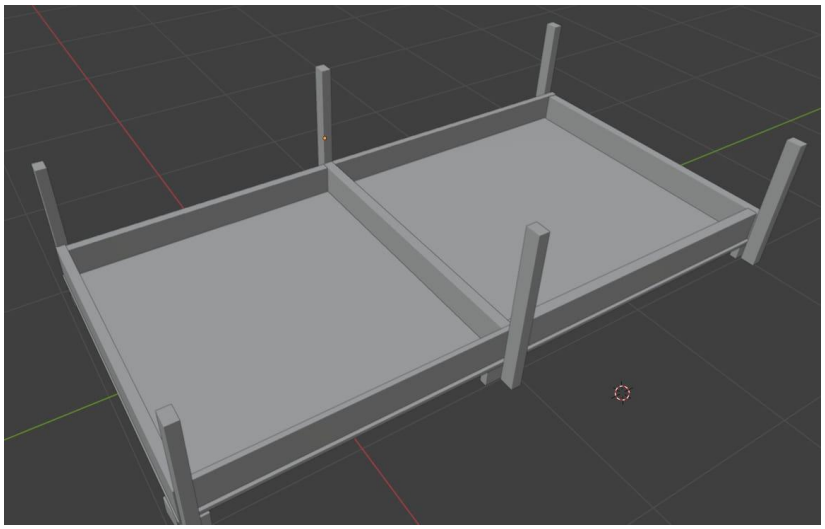

Figura 19 Diseño de huerto en 3D

Fuente: Elaboración propia (2025)

- **Corte de materiales:**

Se cortaron las piezas de madera en dos bloques rectangulares:

- Un bloque de **50 cm x 46 cm**
- Un bloque de **58 cm x 46 cm**
  Ambos bloques servirán como compartimientos para el huerto.

Figura 20 Corte de material para el huerto

Fuente: Elaboración propia (2025)

- **Construcción del primer bloque**:

En la figura 21 se muestra el armado del primer bloque, uniendo las tablas con tornillos y refuerzos de madera en las esquinas. Este proceso permitió comprobar la resistencia de la estructura.

Figura 21 Elaboración del primer bloque (Huerto)

Fuente: Elaboración propia (2025)

- **Unión de los bloques:**

Posteriormente, los dos bloques se unieron en un solo cajón dividido al centro, formando un huerto con dos compartimientos y se añadieron 6 patas de madera en total, distribuidas en las esquinas y en los puntos de unión para dar mayor estabilidad.

Figura 22 Huerto finalizado

Fuente : Elaboración propia (2025)

5.2 Elaboración del techo del huerto

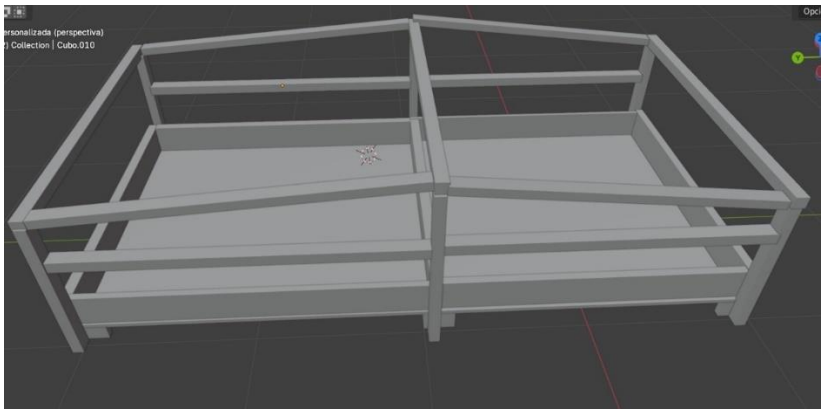

Figura 23 Prototipo de 3D de techo del huerto

Fuente : Elaboración propia (2025)

En la imagen se muestra la estructura en 3D del techo, formada únicamente por los palos que sirven de soporte en los bordes y laterales.

- Son las vigas y travesaños que delimitan el contorno superior del huerto.
- Aún no tiene cubierta (tejas, calamina, malla o plástico), solo se modelaron los bastidores de afuera.
- Los bloques de fondo y la base del huerto ya están hechos, lo que se ve aquí es únicamente el esqueleto superior que sostendrá el techo.

Figura 24 Unión de los componentes del techo

Fuente: Elaboración propia (2025)

- **<u>Colocación de los postes verticales:</u>**
  - Se instalaron los postes principales en las esquinas y en el centro para dar estabilidad.
  - Estos postes son los que sostendrán todo el peso del techo.

5.3 Sensor DHT11 para medir la temperatura y humedad

El objetivo es programar el ESP32 para que los datos de temperatura y humedad del DHT11 se envíen y se vean en Firebase en tiempo real (Realtime Database).

Componentes usados

- 1 cable USB a tipo C
- Fuente de alimentación
- 3 Dupont Cable 10cm M-H-M

Para las conexiones del ESP32 al DHT11

- 1 Esp32s 38 Pines Wifi + Bluetooth
- Tipo: Módulo Wifi + Bluetooth
- Modelo: ESP32 38 Pines
- Voltaje de Alimentación (USB): 5V DC
- Voltaje de Entradas/Salidas pines: 3.3V DC
- Consumo de energía de 0.5mA (sin nada de sensores)
- CPU principal: Tensilica Xtensa 32-bit LX6
- Desempeño: Hasta 600 DMIPS
- Frecuencia de Reloj: hasta 240Mhz

1 DHT11 con LED

- Voltaje de Operación: 3V - 5V DC.
- Rango de medición de temperatura: 0 a 50 °C.
- Precisión de medición de temperatura: ±2.0 °C.
- Resolución Temperatura: 0.1°C.
- Rango de medición de humedad: 20% a 90% RH.
- Precisión de medición de humedad: 5% RH.
- Resolución Humedad: 1% RH.
- Tiempo de sensado: 1 seg.

Conexiones

- Data = pin G4
- GND = GND
- VCC = V5

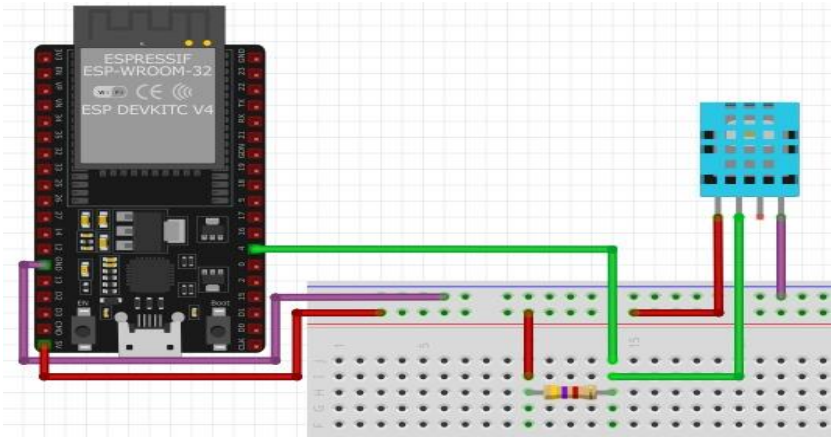

Figura 25 Sensor DHT11 conectado

Fuente: Elaboración propia (2025)

Código de conexión

```
#if defined(ESP32)
  #include <WiFi.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
#include "DHT.h"
// Archivos de ayuda de la librería Firebase
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"
// ------------------------------
// CONFIGURACIÓN SENSOR DHT11
// ------------------------------
#define DHTPIN 4       // Pin GPIO donde está conectado el DHT11
#define DHTTYPE DHT11  // Tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
// ------------------------------
// CONFIGURACIÓN WIFI Y FIREBASE
// ------------------------------
//#define WIFI_SSID "HUAYQU"
//#define WIFI_PASSWORD "SvMnf34Z4Nqv"

#define WIFI_SSID "Leo"
#define WIFI_PASSWORD "12345678"
#define API_KEY "AIzaSyDsEb_YAGcr9S9fKjzmJKuA7RdNuJtQZJg"
#define DATABASE_URL "https://huerto-unandes-default-rtdb.firebaseio.com/"
// Usuario y contraseña Firebase (habilitado en Authentication Email/Password)
#define USER_EMAIL "josueleonardomanriquez@gmail.com"
#define USER_PASSWORD "12341234"
// Objetos de Firebase
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
```

```
// ------------------------------
// VARIABLES DE CONTROL
// ------------------------------
unsigned long sendDataPrevMillis = 0;
const long timerDelay = 10000; // enviar datos cada 10
segundos
bool dht11NodeCreated = false; // bandera para verificar si
ya se creó
void setup() {
 Serial.begin(115200);
 dht.begin();
 // Conectar a WiFi
 WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
 Serial.print("Conectando a WiFi...");
 while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
 }
 Serial.println();
 Serial.println("WiFi conectado.");
 Serial.print("IP: ");
 Serial.println(WiFi.localIP());
 // Configuración Firebase
 config.api_key = API_KEY;
 config.database_url = DATABASE_URL;
 // Autenticación con email y contraseña
 auth.user.email = USER_EMAIL;
 auth.user.password = USER_PASSWORD;
 // Callback del estado del token
 config.token_status_callback = tokenStatusCallback;
 Firebase.begin(&config, &auth);
 Firebase.reconnectWiFi(true);
 Serial.println("Firebase inicializado correctamente.");
 // ------------------------------
 // Verificar si existe el nodo DHT11
 // ------------------------------
 if (!Firebase.RTDB.get(&fbdo, "DHT11")) {
  Serial.println("Nodo DHT11 no existe. Creándolo...");
  // Crear nodo vacío con valores iniciales
  Firebase.RTDB.set(&fbdo, "DHT11/Humedad", 0);
  Firebase.RTDB.set(&fbdo, "DHT11/Temperatura", 0);
  dht11NodeCreated = true;
 } else {
  Serial.println("Nodo DHT11 ya existe.");
  dht11NodeCreated = true;
 }
}
void loop() {
 if (Firebase.ready() && (millis() - sendDataPrevMillis >
timerDelay || sendDataPrevMillis == 0)) {
  sendDataPrevMillis = millis();
  // Leer valores del sensor
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
   Serial.println("Error al leer el DHT11");
   return;
  }
  // Mostrar en Serial
  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.println(" °C");
  // ------------------------------
  // Actualizar datos en Firebase
  // ------------------------------
  if (Firebase.RTDB.setFloat(&fbdo, "DHT11/Humedad",
h)) {
```

```
        Serial.println("Humedad enviada correctamente.");
      } else {
        Serial.println("Error al enviar humedad: " +
fbdo.errorReason());
      }
      if (Firebase.RTDB.setFloat(&fbdo,
"DHT11/Temperatura", t)) {
        Serial.println("Temperatura enviada correctamente.");
      } else {
        Serial.println("Error al enviar temperatura: " +
fbdo.errorReason());
      }
    }
  }
```

5.4 Conexión y programación – ESP32 + FC-28 + Firebase

El objetivo es programar el ESP32 para que lea los datos de humedad del suelo mediante el sensor FC-28 y los envíe en tiempo real a Firebase Realtime Database, de modo que puedan visualizarse remotamente.

Componentes utilizados

1. **Cable USB a micro USB o USB-C**
   - Para alimentación y programación del ESP32.
2. **ESP32 (38 pines, WiFi + Bluetooth)**
   - Tipo: Módulo WiFi + Bluetooth.
   - Voltaje de Alimentación (USB): 5V DC.
   - Voltaje de pines I/O: 3.3V DC.
   - CPU: Tensilica Xtensa 32-bit LX6.
   - Frecuencia de reloj: hasta 240 MHz.
   - Consumo típico: 0.5 mA sin periféricos.
3. **Sensor de humedad del suelo FC-28 (en modo analógico)**
   - Voltaje de operación: 3.3V – 5V DC.

- Salida: Analógica (AO) y digital (DO).
- Función: mide la humedad relativa del suelo mediante conductividad.

4. **Cables Dupont macho-hembra (x3)**
   - Para realizar las conexiones entre ESP32 y FC-28.

Conexiones

- **AO (salida analógica FC-28) → GPIO34** del ESP32.
- **VCC (FC-28) → 3.3V o 5V** del ESP32 (recomendado 3.3V).
- **GND (FC-28) → GND** del ESP32.

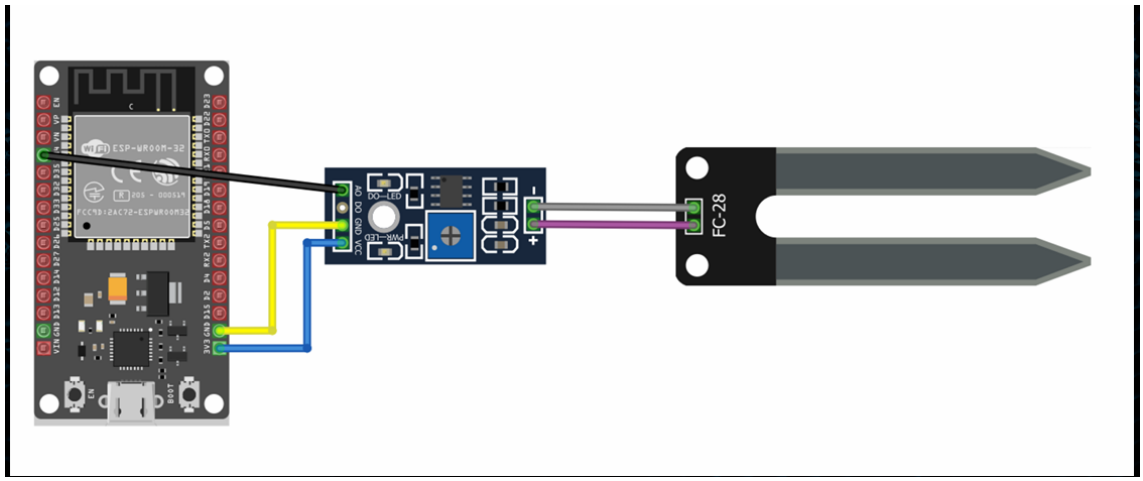

Figura 26 Conexión y programación – ESP32 + FC-28 + Firebase

Fuente: Elaboración propia (2025)

Código de conexión

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

// Archivos de ayuda de la librería Firebase
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"

// ------------------------------
// CONFIGURACIÓN WIFI Y FIREBASE
// ------------------------------
#define WIFI_SSID "TuRedWiFi"
#define WIFI_PASSWORD "TuClaveWiFi"
```

```
#define API_KEY "TU_API_KEY"
#define DATABASE_URL "https://tu-proyecto-default-rtdb.firebaseio.com/"

// Usuario y contraseña Firebase (habilitado en Authentication Email/Password)
#define USER_EMAIL "tu_correo@gmail.com"
#define USER_PASSWORD "tu_contraseña"

// Objetos Firebase
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// ------------------------------
// CONFIGURACIÓN DEL SENSOR FC-28
// ------------------------------
const int sensorPin = 34;  // Pin analógico en ESP32

// ------------------------------
// VARIABLES DE CONTROL
// ------------------------------
unsigned long sendDataPrevMillis = 0;
const long timerDelay = 10000; // enviar datos cada 10 segundos

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

  // Conexión WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Conectando a WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi conectado.");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());

  // Configuración Firebase
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;

  // Autenticación con email y contraseña
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  // Callback del estado del token
  config.token_status_callback = tokenStatusCallback;

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  Serial.println("Firebase inicializado correctamente.");
}

void loop() {
  if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();

    // Leer valores del sensor FC-28
    int valorBruto = analogRead(sensorPin);
    int humedad = map(valorBruto, 0, 4095, 100, 0); // 0% seco – 100% húmedo

    // Mostrar en Serial
```

```
        Serial.print("Valor bruto: ");
        Serial.print(valorBruto);
        Serial.print(" | Humedad: ");
        Serial.print(humedad);
        Serial.println(" %");

        // ------------------------------
        // Actualizar datos en Firebase
        // ------------------------------
        if (Firebase.RTDB.setInt(&fbdo, "FC28/Humedad",
humedad)) {
          Serial.println(" Humedad enviada correctamente.");
        } else {
          Serial.println(" Error al enviar humedad: " +
fbdo.errorReason());
        }
      }
    }
```

5.5 Conexión y programación ESP32 + BH1750 + Tira LED RGB

Objetivo programar el ESP32 para que los datos de luminosidad del BH1750 se envíen a Firebase en tiempo real y controle automáticamente una tira LED RGB de 12V para complementar la iluminación cuando sea necesario.

Componentes utilizados

1. **cable USB**
   - Fuente de alimentación para programación del ESP32
2. **Cables Dupont**
   - Para las conexiones del ESP32 al BH1750
   - Cables M-H y M-M según necesidad
3. **1 ESP32s 38 Pines Wifi + Bluetooth**
   - Tipo: Módulo Wifi + Bluetooth
   - Modelo: ESP32 38 Pines

- Voltaje de Alimentación (USB): 5V DC
- Voltaje de Entradas/Salidas pines: 3.3V DC
- CPU principal: Tensilica Xtensa 32-bit LX6
- Frecuencia de Reloj: hasta 240MHz

4. **1 Sensor BH1750**
   - Voltaje de Operación: 3.3V - 5V DC
   - Comunicación: I2C
   - Dirección I2C: 0x23 (por defecto)
   - Rango de medición: 1 - 65535 lux
   - Resolución: 1 lux
   - Precisión: ±20%
5. **Tira LED RGB SMD 12V (2835/5050)**
   - Voltaje: 12V DC
   - Tipo: SMD 2835 o 5050
   - Colores: RGB (Rojo, Verde, Azul)
   - Pines: +12V, R, G, B
   - Fuente de poder (Switch Power) 12V
   - Voltaje de salida: 12V DC
   - Corriente: Según consumo de la tira LED (mínimo 2A recomendado)
6. **Resistencias**
   - Valor: 220Ω - 1kΩ
7. **Protoboard y cables jumper**
   - Para realizar las conexiones

Conexiones

- Fuente 12V (+) → Tira LED (+12V)
- Fuente 12V (-) → GND común → ESP32 GND
- USB → ESP32 (para programación y alimentación del microcontrolador)

Código de conexión

```
#include <WiFi.h>
#include <Wire.h>
#include <Firebase_ESP_Client.h>
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>
// WIFI
const char* ssid = "POCO X6 5G";
const char* password = "luisfer456";
// FIREBASE
#define API_KEY
"AIzaSyDsEb_YAGcr9S9fKjzmJKuA7RdNuJtQZJg"
#define DATABASE_URL "https://huerto-unandes-default-
rtdb.firebaseio.com/"
#define USER_EMAIL
"josueleonardomanriquez@gmail.com"
#define USER_PASSWORD "12341234"
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
// SENSOR BH1750
#define BH1750_ADDRESS 0x23
// PINES
const int LED_PIN = 2;
const int BUZZER_PIN = 5;
// NUEVOS RANGOS DE LUZ
const float LUZ_BAJA_MIN = 5000;
const float LUZ_BAJA_MAX = 10000;
const float LUZ_MINIMA_MIN = 15000;
const float LUZ_MINIMA_MAX = 20000;
const float LUZ_OPTIMA_MIN = 25000;
const float LUZ_OPTIMA_MAX = 50000;
const float LUZ_EXCESIVA = 55000;
bool led_encendido = false;
void setup() {
 Serial.begin(115200);
 Serial.println("Iniciando sistema...");
 pinMode(LED_PIN, OUTPUT);
 pinMode(BUZZER_PIN, OUTPUT);
 digitalWrite(LED_PIN, LOW);
 Wire.begin(21, 22);
 iniciarSensor();
 conectarWiFi();
 configurarFirebase();
 Serial.println("SISTEMA LISTO");
 sonar(2);
}
void loop() {
 float lux = leerSensor();
 if (lux >= 0) {
  mostrarDatos(lux);
  controlarLED(lux);
  enviarFirebase(lux);
```

```
  } else {
   Serial.println("Error leyendo sensor");
  }
  delay(10000);
 }
 float leerSensor() {
  Wire.beginTransmission(BH1750_ADDRESS);
  Wire.write(0x10);
  Wire.endTransmission();
  delay(120);

  Wire.requestFrom(BH1750_ADDRESS, 2);
  if (Wire.available() >= 2) {
   uint16_t nivel = Wire.read() << 8;
   nivel |= Wire.read();
   return nivel / 1.2;
  }
  return -1;
 }
 void controlarLED(float lux) {
  // Encender LED si la luz es muy baja (menos de 5000 lux)
  if (lux < LUZ_BAJA_MIN && !led_encendido) {
   digitalWrite(LED_PIN, HIGH);
   led_encendido = true;
   Serial.println("LED ON - Luz muy baja");
   sonar(1);
  }
  // Apagar LED cuando la luz sea aceptable (más de 7000
lux para evitar parpadeo)
  if (lux > 7000 && led_encendido) {
   digitalWrite(LED_PIN, LOW);
   led_encendido = false;
   Serial.println("LED OFF - Luz suficiente");
  }
 }
 void enviarFirebase(float lux) {
  if (!Firebase.ready()) return;
  String estado = obtenerEstado(lux);
  float porcentaje = calcularPorcentaje(lux);
  // USAR ESTAS FUNCIONES EXACTAS:
  if (Firebase.RTDB.setFloat(&fbdo, "/vegetales/lux", lux)) {
   Serial.println("Lux enviado");
  }


  if (Firebase.RTDB.setString(&fbdo, "/vegetales/estado",
estado)) {
   Serial.println("Estado enviado");
  }
  if (Firebase.RTDB.setFloat(&fbdo, "/vegetales/porcentaje",
porcentaje)) {
   Serial.println("Porcentaje enviado");
  }
  if (Firebase.RTDB.setBool(&fbdo, "/vegetales/led",
led_encendido)) {
   Serial.println("LED estado enviado");
  }
 }
 String obtenerEstado(float lux) {
  if (lux < LUZ_BAJA_MIN) {
   return "Luz Muy Baja";
  }
  else if (lux >= LUZ_BAJA_MIN && lux <=
LUZ_BAJA_MAX) {
   return "Luz Baja";
  }
  else if (lux > LUZ_BAJA_MAX && lux <
LUZ_MINIMA_MIN) {
   return "Luz Insuficiente";
  }
```

```
      else if (lux >= LUZ_MINIMA_MIN && lux <=
LUZ_MINIMA_MAX) {
        return "Luz Minima Aceptable";
      }
      else if (lux > LUZ_MINIMA_MAX && lux <
LUZ_OPTIMA_MIN) {
        return "Luz Adecuada";
      }
      else if (lux >= LUZ_OPTIMA_MIN && lux <=
LUZ_OPTIMA_MAX) {
        return "Luz Optima para Floracion";
      }
      else if (lux > LUZ_OPTIMA_MAX && lux <
LUZ_EXCESIVA) {
        return "Luz Alta";
      }
      else {
        return "Luz Excesiva";
      }
    }
    float calcularPorcentaje(float lux) {
      // Porcentaje basado en los rangos óptimos
      if (lux < LUZ_BAJA_MIN) {
        // Muy baja: 0-20%
        return (lux / LUZ_BAJA_MIN) * 20;
      }
      else if (lux >= LUZ_BAJA_MIN && lux <=
LUZ_BAJA_MAX) {
        // Baja: 20-40%
        return 20 + ((lux - LUZ_BAJA_MIN) /
(LUZ_BAJA_MAX - LUZ_BAJA_MIN)) * 20;
      }
      else if (lux > LUZ_BAJA_MAX && lux <
LUZ_MINIMA_MIN) {
        // Insuficiente: 40-60%
        return 40 + ((lux - LUZ_BAJA_MAX) /
(LUZ_MINIMA_MIN - LUZ_BAJA_MAX)) * 20;
      }
      else if (lux >= LUZ_MINIMA_MIN && lux <=
LUZ_MINIMA_MAX) {
        // Mínima aceptable: 60-80%
        return 60 + ((lux - LUZ_MINIMA_MIN) /
(LUZ_MINIMA_MAX - LUZ_MINIMA_MIN)) * 20;
      }
      else if (lux > LUZ_MINIMA_MAX && lux <
LUZ_OPTIMA_MIN) {
        // Adecuada: 80-100%
        return 80 + ((lux - LUZ_MINIMA_MAX) /
(LUZ_OPTIMA_MIN - LUZ_MINIMA_MAX)) * 20;
      }
      else if (lux >= LUZ_OPTIMA_MIN && lux <=
LUZ_OPTIMA_MAX) {
        // Óptima: 100%
        return 100;
      }
      else if (lux > LUZ_OPTIMA_MAX && lux <
LUZ_EXCESIVA) {
        // Alta: 100-120%
        return 100 + ((lux - LUZ_OPTIMA_MAX) /
(LUZ_EXCESIVA - LUZ_OPTIMA_MAX)) * 20;
      }
      else {
        // Excesiva: más de 120%
        return 120 + ((lux - LUZ_EXCESIVA) /
LUZ_EXCESIVA) * 30;
      }
    }
    void mostrarDatos(float lux) {
      Serial.println("====================");
      Serial.print("Luz: ");
```

```
    Serial.print(lux, 0);
    Serial.println(" lux");
    Serial.print("Estado: ");
    Serial.println(obtenerEstado(lux));
    Serial.print("Porcentaje: ");
    Serial.print(calcularPorcentaje(lux), 1);
    Serial.println("%");
    Serial.print("LED: ");
    Serial.println(led_encendido ? "ON" : "OFF");
    // Mostrar información de rangos
    Serial.println("--- RANGOS DE LUZ ---");
    if (lux < LUZ_BAJA_MIN) {
      Serial.println("< 5,000 lux: Luz muy baja");
    }
    else if (lux <= LUZ_BAJA_MAX) {
      Serial.println("5,000 - 10,000 lux: Luz baja");
    }
    else if (lux < LUZ_MINIMA_MIN) {
      Serial.println("10,000 - 15,000 lux: Luz insuficiente");
    }
    else if (lux <= LUZ_MINIMA_MAX) {
      Serial.println("15,000 - 20,000 lux: Luz mínima
aceptable");
    }
    else if (lux < LUZ_OPTIMA_MIN) {
      Serial.println("20,000 - 25,000 lux: Luz adecuada");
    }
    else if (lux <= LUZ_OPTIMA_MAX) {
      Serial.println("25,000 - 50,000 lux: Luz óptima para
floración");
    }
    else if (lux < LUZ_EXCESIVA) {
      Serial.println("50,000 - 55,000 lux: Luz alta");
    }
    else {
      Serial.println("> 55,000 lux: Luz excesiva");
    }
    Serial.println("====================");
  }
  void conectarWiFi() {
    WiFi.begin(ssid, password);
    Serial.print("Conectando WiFi");

    int intentos = 0;
    while (WiFi.status() != WL_CONNECTED && intentos <
20) {
      delay(500);
      Serial.print(".");
      intentos++;
    }
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println();
      Serial.println("WiFi OK");
      Serial.println(WiFi.localIP());
    } else {
      Serial.println("Error WiFi");
    }
  }
  void configurarFirebase() {
    config.api_key = API_KEY;
    config.database_url = DATABASE_URL;

    auth.user.email = USER_EMAIL;
    auth.user.password = USER_PASSWORD;
    config.token_status_callback = tokenStatusCallback;
    Firebase.begin(&config, &auth);
    Firebase.reconnectWiFi(true);
    Serial.println("Firebase configurado");
  }
  void iniciarSensor() {
```

```
  Wire.beginTransmission(BH1750_ADDRESS);
  Wire.write(0x10);
  if (Wire.endTransmission() == 0) {
    Serial.println("Sensor BH1750 OK");
    delay(200);
  } else {
    Serial.println("Error sensor BH1750");
  }
}
void sonar(int veces) {
  for (int i = 0; i < veces; i++) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(200);
    digitalWrite(BUZZER_PIN, LOW);
    delay(100);
  }
}
```

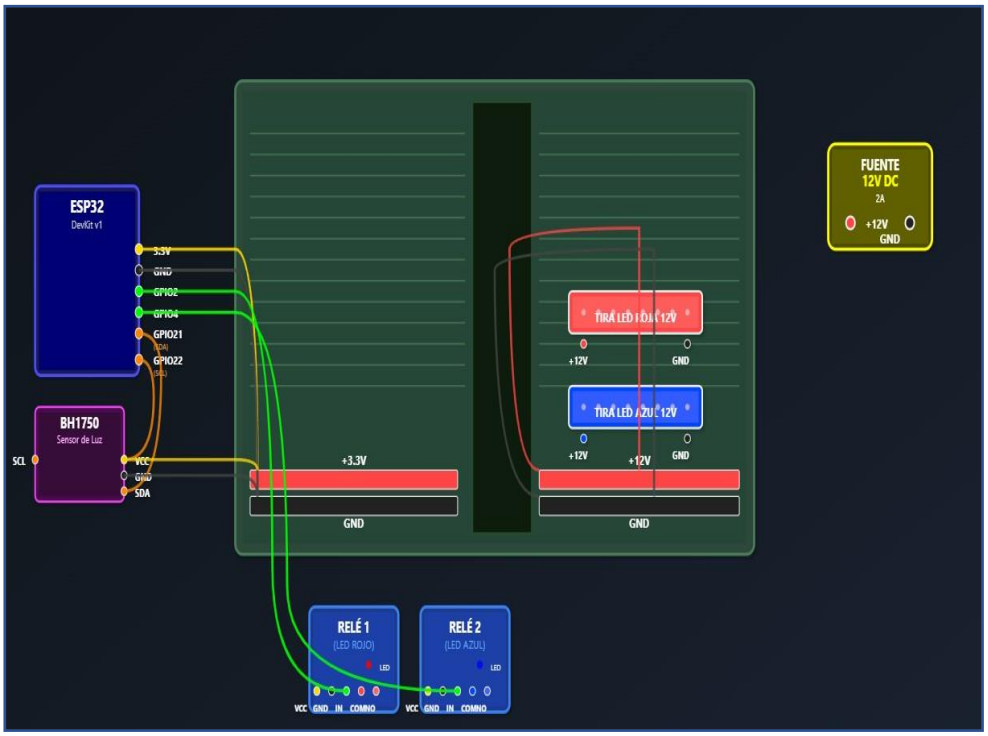

Figura 27 Conexión y programación ESP32 + BH1750 + Tira LED RGB

Fuente : Elaboración propia (2025)

5.6 Conexión y Programación con ESP32, DHT22 y Firebase

El objetivo es programar el ESP32 para que los datos de temperatura y humedad obtenidos del sensor DHT22 se envíen en tiempo real a Firebase (Realtime Database). Además, los datos se guardarán con fecha y hora exacta obtenida desde un servidor NTP, de manera que cada registro quede almacenado de forma histórica y organizada.

Componentes utilizados

1. **1 cable USB a tipo C**
   - Fuente de alimentación para el ESP32.
2. **3 cables Dupont macho-hembra de 10 cm**
   - Para las conexiones entre el ESP32 y el DHT22.
3. **1 ESP32 de 38 Pines con WiFi + Bluetooth**
   - Tipo: Módulo WiFi + Bluetooth.
   - Modelo: ESP32 38 Pines.
   - Voltaje de Alimentación (USB): 5V DC.
   - Voltaje de Entradas/Salidas en pines: 3.3V DC.
   - CPU principal: Tensilica Xtensa 32-bit LX6.
   - Frecuencia de reloj: hasta 240 MHz.
4. **1 DHT22 con LED indicador**
   - Voltaje de operación: 3V – 5V DC.
   - Rango de medición de temperatura: -40 a 80 °C.
   - Precisión de temperatura: ±0.5 °C.
   - Resolución temperatura: 0.1 °C.

- Rango de medición de humedad: 0% a 100% RH.
- Precisión de humedad: ±2% RH.
- Resolución de humedad: 0.1% RH.
- Tiempo de sensado: 2 segundos.

Conexiones

- DATA (DHT22) → GPIO 4 (ESP32)
- GND (DHT22) → GND (ESP32)
- VCC (DHT22) → 5V (ESP32)

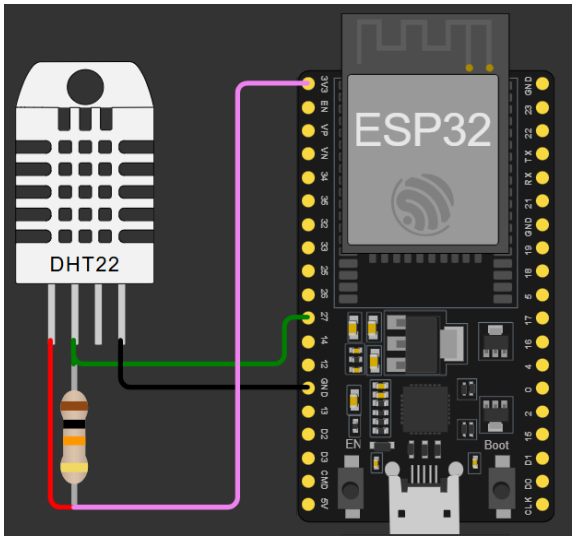

Figura 28 Conexión con ESP32 y DHT22.

Fuente : Elaboración propia (2025)

Código de conexión

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include "DHT.h"
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <time.h>
// Configura tu red WiFi
#define WIFI_SSID "Josue"
#define WIFI_PASSWORD "12341234"
// Configura Firebase
#define API_KEY
"AIzaSyDsEb_YAGcr9S9fKjzmJKuA7RdNuJtQZJg"
#define DATABASE_URL "https://huerto-unandes-default-
rtdb.firebaseio.com/"
// Configura DHT22
#define DHTPIN 4
```

```
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
// Objetos Firebase
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
// NTP Client para hora real
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", -14400, 60000);
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = "josueleonardomanriquez@gmail.com";
  auth.user.password = "12341234";
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
  dht.begin();
  timeClient.begin();
}

void loop() {
  timeClient.update();
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) return;
  unsigned long epochTime = timeClient.getEpochTime();
  struct tm *ptm = localtime((time_t *)&epochTime);
  char dateTime[20];
  sprintf(dateTime, "%02d-%02d-%04d_%02d-%02d-%02d",
        ptm->tm_mday, ptm->tm_mon + 1, ptm->tm_year + 1900,
        ptm->tm_hour, ptm->tm_min, ptm->tm_sec);

  String key = String(dateTime);

  Firebase.RTDB.setFloat(&fbdo, "/DHT22/" + key +
"/temperatura", t);
  Firebase.RTDB.setFloat(&fbdo, "/DHT22/" + key +
"/humedad", h);
  delay(5000);
}
```

5.7 Integración de los componentes

En las siguientes imágenes se observa la estructura física del prototipo, conformada por un marco de madera, el área de cultivo con sustrato y la instalación de la tira LED para iluminación artificial. Asimismo, se aprecia la conexión de los sensores y actuadores al ESP32, junto con las pruebas de funcionamiento realizadas mediante la computadora.

Figura 29 Instalación de la tira LED

En la siguiente imagen se observa el proceso de montaje de la tarjeta de control (ESP32), sensores y conexiones eléctricas dentro de una caja de resguardo hecha de cartón.

Función de la caja: proteger el protoboard, cables y módulos de factores externos (humedad, polvo, contacto directo con la tierra del huerto).

Proceso mostrado:

- Se organizan los cables Dupont y conexiones del protoboard.
- Los sensores y actuadores quedan conectados al ESP32.
- Se asegura la caja con silicona caliente, como se ve en la imagen, para mantener firme la estructura y evitar que los cables se suelten.

Importancia: este encapsulado garantiza que el sistema electrónico tenga orden, estabilidad mecánica y seguridad eléctrica, facilitando el mantenimiento y evitando cortocircuitos.

Figura 30 Instalación y ensamblaje de los componentes electrónicos

5.8 Diagrama de conexión general del sistema automatizado

En la figura se observa el esquema completo de conexión del sistema de riego y control climático automatizado. Este diagrama integra todos los sensores, actuadores y módulos alrededor del ESP32, que actúa como unidad central de procesamiento y comunicación con Firebase.

Sensores conectados:

- DHT22, encargados de medir la temperatura y humedad relativa del aire en diferentes sectores del huerto.
- Sensores de humedad de suelo (FC-28), que permiten monitorear el nivel de humedad en distintas macetas o compartimientos de cultivo.

Actuadores controlados mediante relés:

- Ventiladores, instalados en la parte superior para la ventilación y control de la temperatura interna.
- Una tira LED RGB de 12V, que se enciende automáticamente cuando los niveles de luz natural son insuficientes, garantizando la iluminación óptima para el crecimiento de las plantas.
- Bomba de agua / electroválvula, que regula el riego de acuerdo con los valores obtenidos por los sensores de humedad de suelo.

Alimentación del sistema:

- El ESP32 se alimenta mediante conexión USB o fuente regulada de 5V.
- Los relés y actuadores de mayor consumo se alimentan con fuentes externas (batería de 9V y packs de pilas AA/AAA), compartiendo la misma referencia de GND para mantener estabilidad en las señales de control.

Importancia del diagrama:

Este esquema resume la arquitectura práctica del prototipo, mostrando cómo los distintos sensores transmiten datos al microcontrolador, y cómo los actuadores son gestionados de forma automática para mantener un microclima estable (temperatura, humedad, iluminación y riego). Además, evidencia la modularidad del diseño, lo que permite futuras expansiones o reemplazos de componentes de manera sencilla.

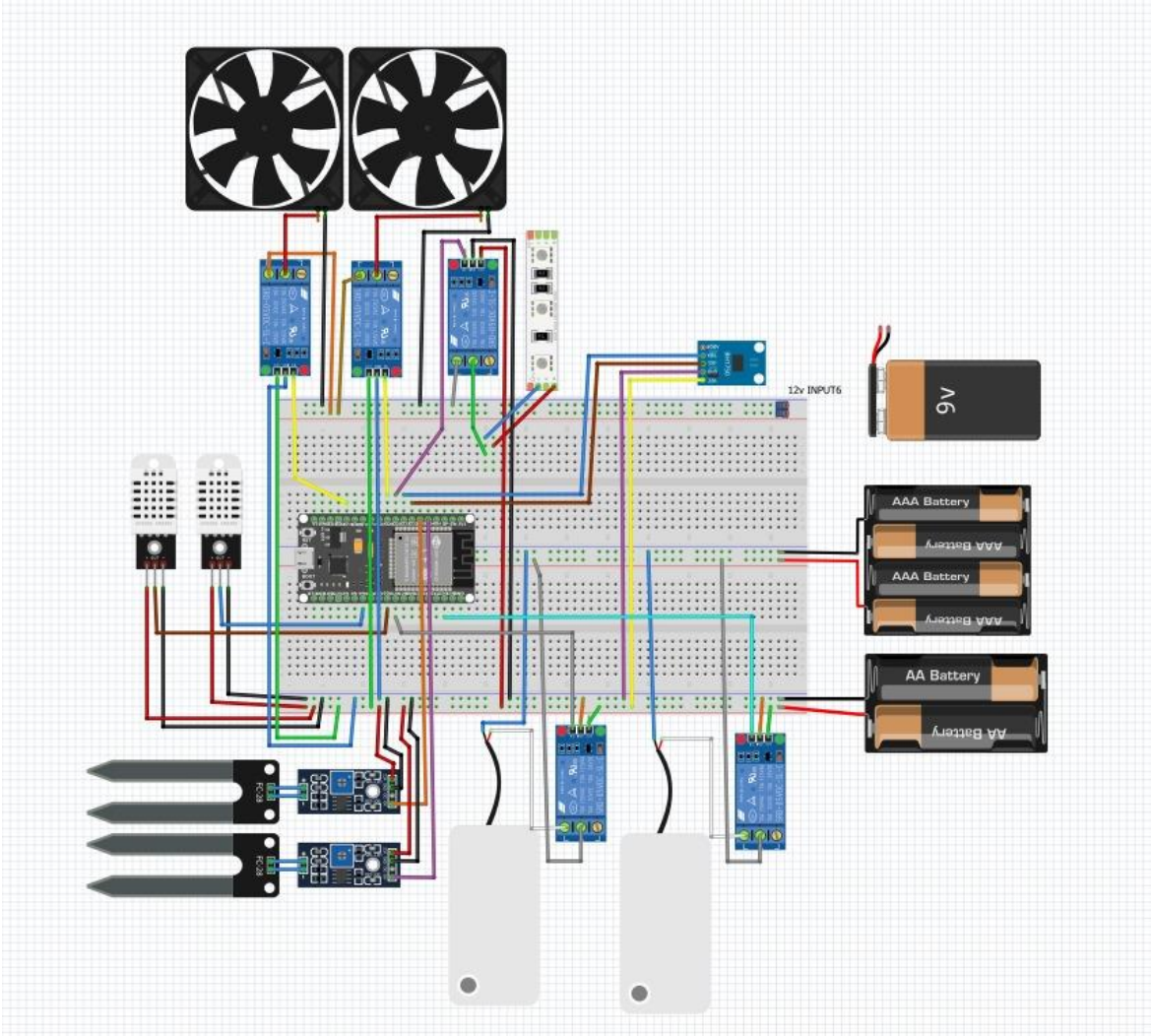

Figura 31 Diagrama de conexión general del sistema automatizado

5.9 Prototipo final del sistema automatizado en funcionamiento

En la figura se aprecia la finalización del montaje físico del sistema multifunción para el control de microclima agrícola. La estructura de madera y plástico transparente permite mantener un ambiente controlado, mientras que la instalación interna de los sensores y actuadores está organizada en una caja lateral de protección.

- La tira LED RGB proporciona iluminación artificial regulada, como se observa en el tono púrpura dentro del huerto.

- Los sensores de humedad de suelo y de temperatura/humedad del aire están distribuidos en la superficie de cultivo para obtener datos en tiempo real.
- El ESP32 y el conjunto de relés están instalados en el costado derecho, dentro de una caja protectora, con el cableado organizado para evitar interferencias y facilitar el mantenimiento.
- El sustrato agrícola ya está dispuesto dentro de la estructura, lo que permite realizar pruebas reales de crecimiento en condiciones controladas.

Este armado constituye la integración completa de los módulos de hardware desarrollados en el proyecto: sensado ambiental, control de iluminación, riego automático y ventilación. Además, demuestra la funcionalidad real del sistema, capaz de recrear un microclima óptimo para cultivos en condiciones frías de la ciudad de La Paz.

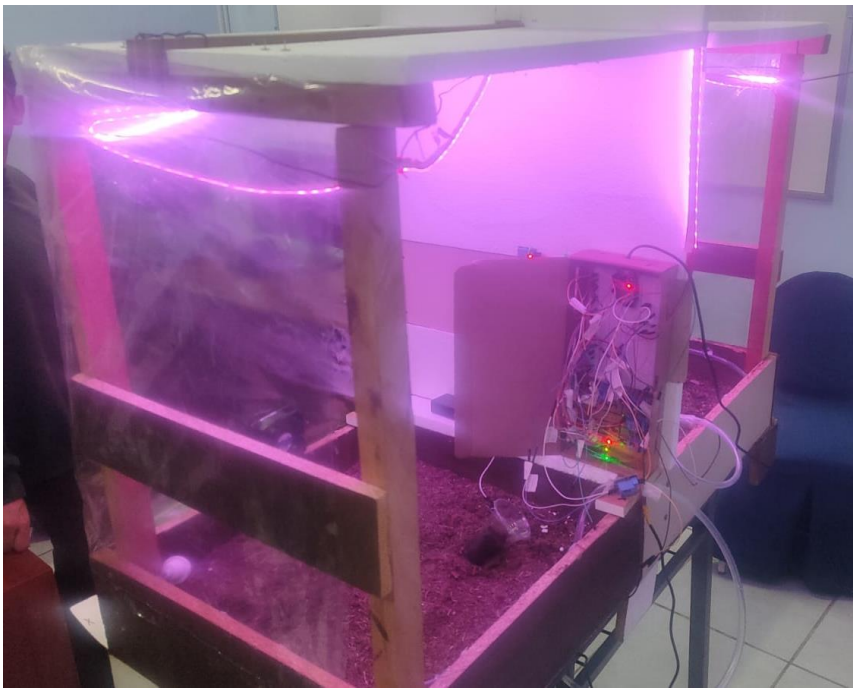

Figura 32 Prototipo final del sistema automatizado en funcionamiento

5.10 Interfaz del usuario

La interfaz gráfica del sistema fue desarrollada en formato dashboard web, accesible desde cualquier navegador. Permite al usuario visualizar en tiempo real las condiciones ambientales del huerto y controlar los actuadores principales.

Características principales:

1. **Distribución en dos lados (Side A y Side B):**
   - Side A (DHT11): monitorea temperatura, humedad del aire, humedad del suelo y luz para cultivos como la zanahoria.
   - Side B (DHT22): realiza las mismas mediciones, pero con mayor precisión, orientado a cultivos como el tomate.
2. **Indicadores de variables ambientales:**
   - Temperatura (°C).
   - Humedad del aire (%).
   - Humedad del suelo (%).
   - Iluminación (lux).
3. **Perfiles de cultivos configurables:**
   - El usuario selecciona la planta a cultivar (ejemplo: zanahoria o tomate).
   - El sistema carga automáticamente los valores objetivos (temperatura mínima, humedad del suelo, etc.) para ese cultivo.
4. **Panel de actuadores:**
   - Botones de control manual para ventilador (Fan), calefactor (Heater), bomba de agua (Pump) y luces (Light).
   - Estado mostrado en tiempo real: ON/OFF.

5. **Visualización gráfica:**

Cada variable tiene su gráfico dinámico donde se representan los datos captados por los sensores en tiempo real.

Esto permite al usuario identificar rápidamente si las condiciones están dentro del rango óptimo.

Importancia: La interfaz centraliza el control del sistema, combinando monitorización, configuración de cultivos y gestión de actuadores en un solo espacio. De esta forma, el agricultor puede mantener un microclima óptimo de manera sencilla e intuitiva.

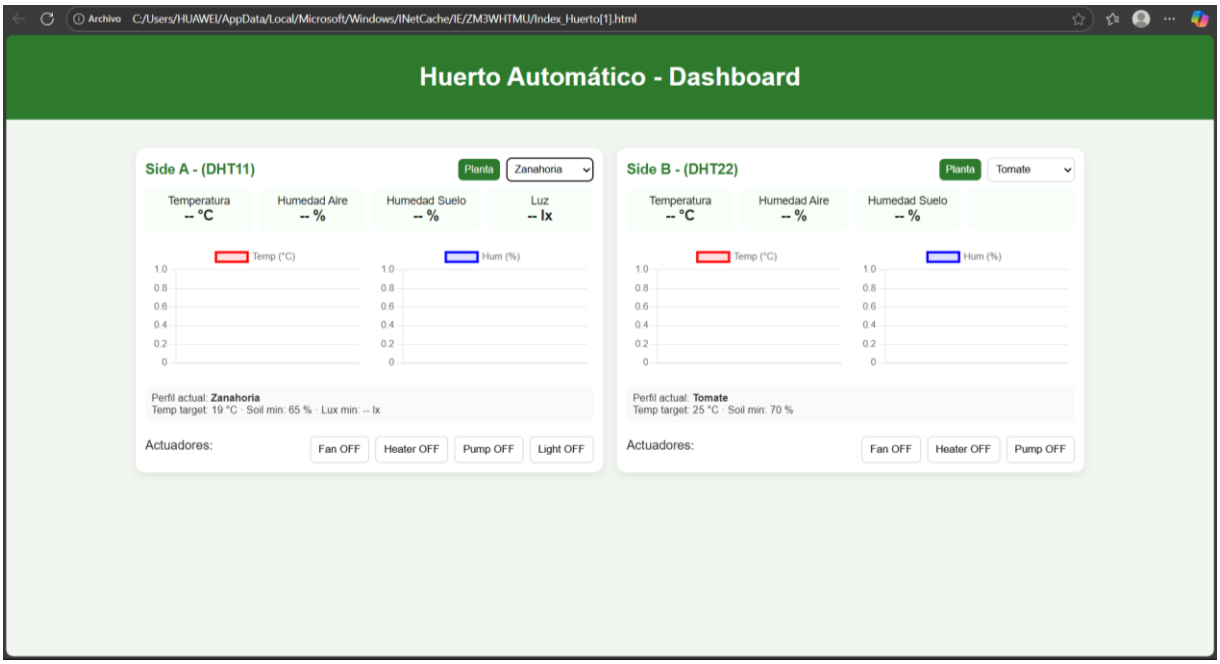

Figura 33 Interfaz web del sistema

5.11 Diagrama de flujo del modelo de negocio alternativo

Con el fin de representar de manera clara el funcionamiento general del sistema de riego multifunción, se elaboró un diagrama de flujo que muestra el recorrido de la información desde la interacción del agricultor con la interfaz del sistema, hasta la toma de decisiones

automáticas mediante el microcontrolador ESP32 y el almacenamiento en la nube con Firebase.

El modelo alternativo se basa en que los sensores ambientales (temperatura, humedad del aire y del suelo, y nivel de luz) envían los datos en tiempo real al ESP32, el cual procesa la información y decide si se requiere activar los actuadores: riego automático, ventilación, calefacción o iluminación LED. Toda esta información se registra en la base de datos de Firebase, lo que permite generar notificaciones y reportes en tiempo real al agricultor a través de la aplicación o dashboard.

Este flujo garantiza que el agricultor tenga un control continuo del estado de los cultivos, con alertas inmediatas y la posibilidad de intervenir manualmente si lo desea.

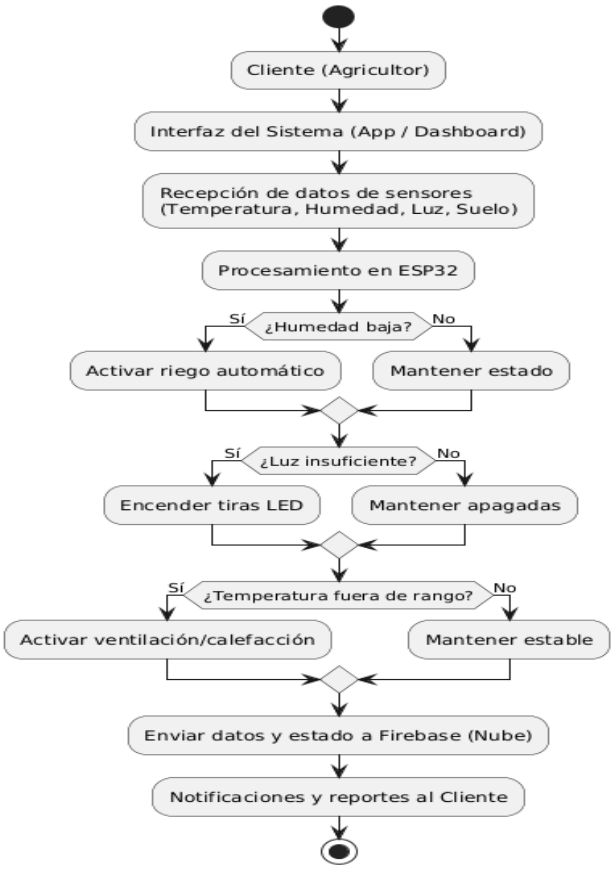

Figura 34 Diagrama de flujo del modelo de negocio alternativo

5.12 Diagrama del Sistema de riego multifunción Automatizado

Este diagrama muestra la finalización del proyecto y representa de manera clara cómo funciona el sistema: el usuario interactúa con la aplicación web/móvil, la cual se conecta con Firebase en la nube; el ESP32 recibe y procesa los datos de los sensores (temperatura, humedad, luz y humedad del suelo) y, en tiempo real, controla los actuadores como la bomba de agua, la tira LED y el ventilador, garantizando el correcto funcionamiento del huerto automatizado.

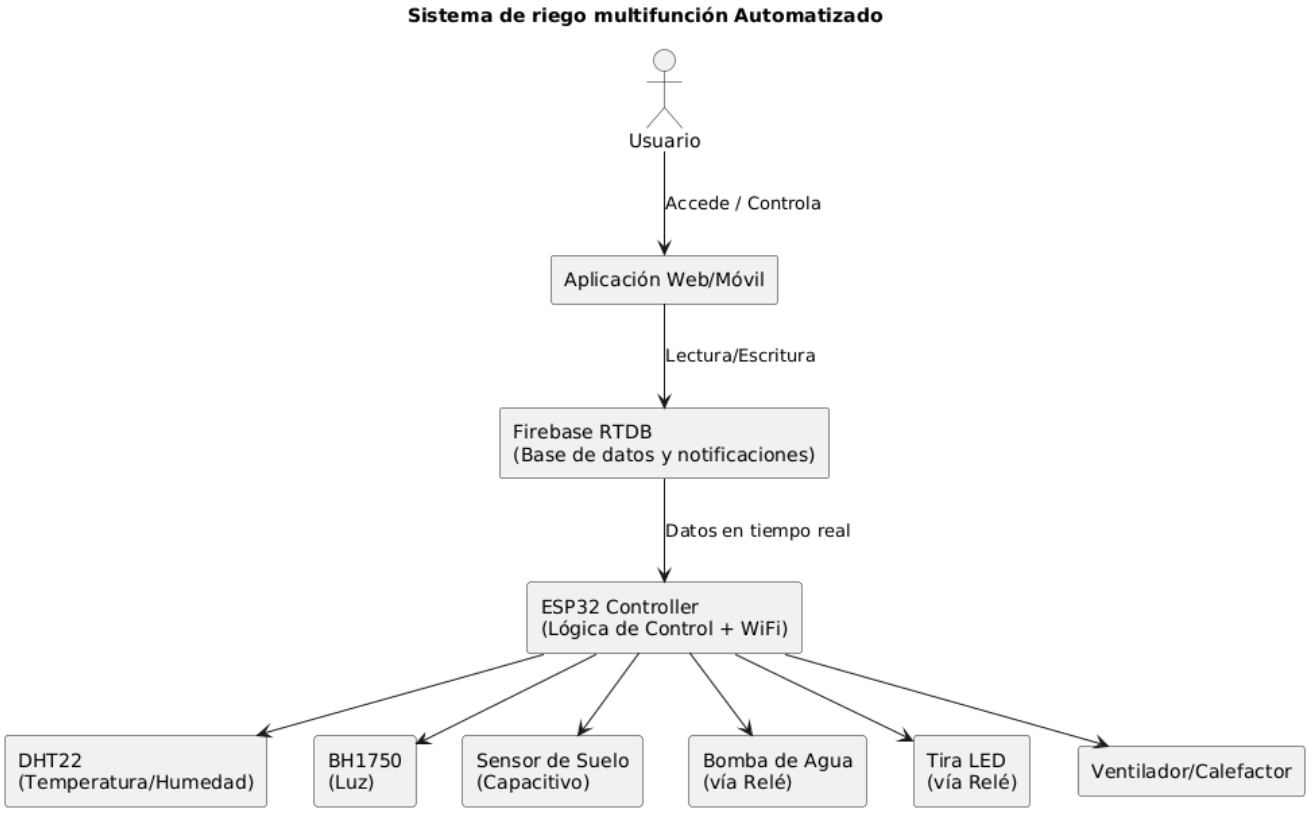

Figura 35 Diagrama del Sistema de riego multifunción Automatizado

VI. METODOLOGÍA DE TRABAJO — Scrum con Jira

- Para la gestión de nuestro proyecto del huerto automatizado, elegimos utilizar la metodología ágil Scrum, ya que nos permite organizar de manera eficiente las tareas, distribuir responsabilidades y adaptarnos a cambios durante el desarrollo. Scrum se basa en ciclos de trabajo cortos llamados sprints, reuniones periódicas de seguimiento y la entrega continua de avances parciales.

Contamos con un equipo de 8 integrantes, lo que permite dividir el trabajo en roles claros: algunos miembros se enfocan en el desarrollo de software (programación de sensores, control de luces y registro de datos), otros en el diseño e integración de hardware (sensores, protoboard, tiras de LED), y algunos en pruebas y documentación del proyecto.

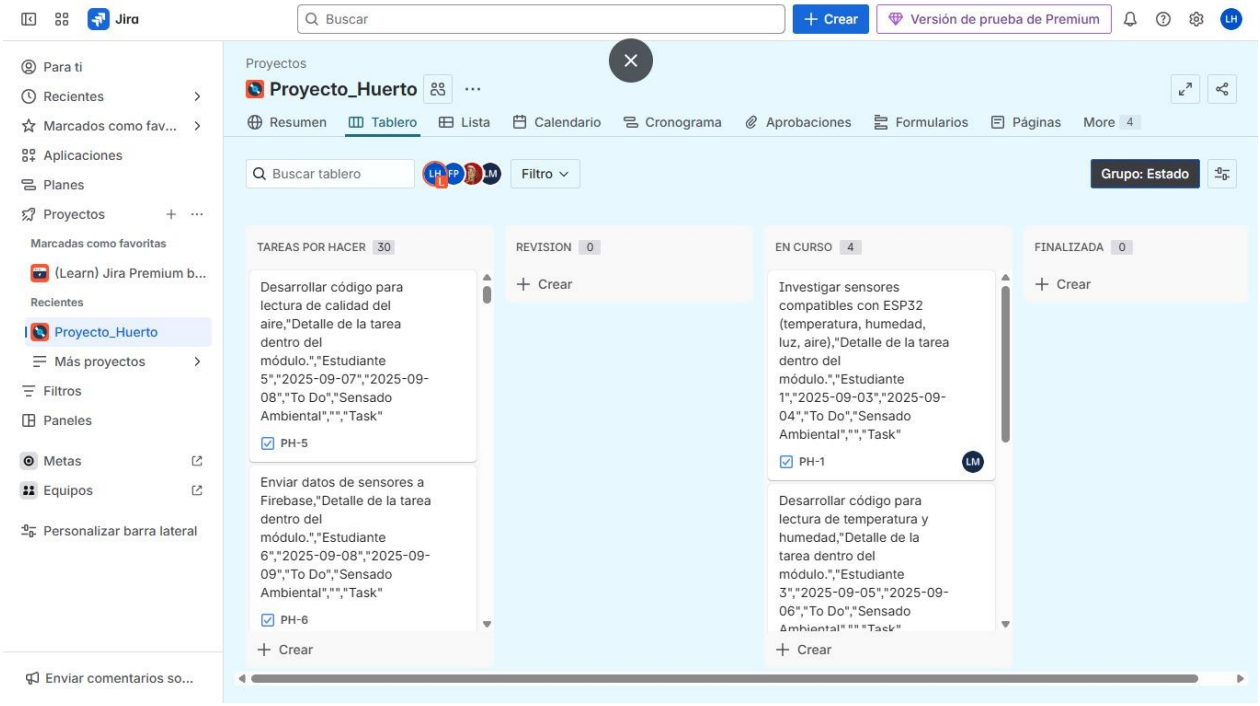

Figura 36 Interfaz del proyecto en JIra

Fuente: Elaboración propia (2025)

En la Figura 37 se puede observar la vista en lista de tareas:

- Tipo / Clave (PH-1, PH-2, PH-3, etc.): son identificadores únicos de cada tarea del proyecto, con el prefijo “PH” que corresponde a Proyecto Huerto.
- Resumen: describe brevemente la tarea. Ejemplos:
  - Investigar sensores compatibles con ESP32.
  - Desarrollar código para lectura de temperatura y humedad.
  - Configurar conexiones y pines en ESP32.
  - Enviar datos de sensores a Firebase.

- Estado: indica el progreso de cada tarea según el flujo de trabajo definido:
  - En curso → tareas que los integrantes están desarrollando.
  - Tareas por hacer → pendientes de iniciar.
- Comentarios: espacio colaborativo para que los integrantes del equipo añadan observaciones o avances.
- Persona asignada: miembro responsable de completar la tarea (ejemplo: Laura, Josue, Fernando).

De esta manera, el tablero en Jira nos permitió dividir el trabajo en sprints, asignar responsabilidades a cada integrante y dar visibilidad del progreso del equipo.

Figura 37 Lista de tareas en Jira

Fuente: Elaboración propia (2025)

6.1 Ejemplo de Tarea en Jira

La tarea PH-6: "Enviar datos de sensores a Firebase" muestra cómo se documentan y gestionan las actividades del proyecto en Jira.

Elementos de la tarea:

- Clave (PH-6): identificador único de la tarea dentro del backlog del proyecto.
- Resumen: breve descripción de la actividad, en este caso, el desarrollo de la conexión entre los sensores y la base de datos en la nube (Firebase).

- Descripción: espacio para detallar el alcance de la tarea, pasos a seguir o especificaciones técnicas.
- Actividades vinculadas: permite relacionar esta tarea con otras dependientes o asociadas (ej. primero programar el sensor, luego enviar datos a Firebase).
- Comentarios: los integrantes del equipo pueden dejar avances, dudas o decisiones tomadas.
- Detalles (panel derecho):
    - Informador: quién creó la tarea (ejemplo: Alejandro Cortes).
    - Persona asignada: miembro responsable de completarla (en este caso aún está “sin asignar”).
    - Estado: indica la fase en que se encuentra la tarea (“Tareas por hacer”, “En curso” o “Completado”).
    - Fechas: inicio y vencimiento de la tarea para controlar tiempos dentro del sprint.
    - Etiquetas: se pueden usar para clasificar (ej. Sensado Ambiental).

Este formato permite que cada integrante tenga claridad sobre qué debe hacer, cuándo y con qué prioridad, facilitando la gestión en los sprints y evitando confusiones dentro del equipo.

Figura 38 Tarea en jira

Fuente: Elaboración propia (2025)

De esta manera, Jira nos permitió dar un seguimiento ágil, colaborativo y transparente a cada actividad del proyecto, asegurando que los entregables de cada sprint fueran completados en tiempo y forma.

6.2 Explicación de los Scripts en Scrum:

6.2.1 Sprint 1 – Sensado Ambiental (DHT11)

- **Product Backlog asociado:**
  - Monitorear la temperatura y humedad del aire.
  - Enviar los datos a Firebase.
  - Visualizarlos en el dashboard del usuario.
- **Historia de usuario:** *"Como agricultor quiero conocer la temperatura y humedad del aire en tiempo real, para asegurar condiciones óptimas en el cultivo."*
- **Tarea en Jira (PH-2):** "Desarrollar código para lectura de temperatura y humedad (DHT11) y enviar datos a Firebase."

- **Sprint asignado:** Sprint 1 → Enfoque en sensado ambiental básico.
- **Responsable:** Leandro
- **Sprint Backlog:**
  - Conectar sensor DHT11 al ESP32.
  - Programar script de lectura de temperatura y humedad.
  - Configurar envío de datos a Firebase.
  - Visualizar información en el dashboard.
- **Interacción:** El script se conecta al ESP32 para obtener datos y los envía a Firebase, donde luego el Usuario puede visualizarlos en el dashboard. Además, alimenta al módulo de alertas que genera notificaciones cuando los valores salen del rango óptimo.
- **Entregable (Incremento):**

  El sistema es capaz de mostrar en tiempo real la temperatura y humedad del ambiente.

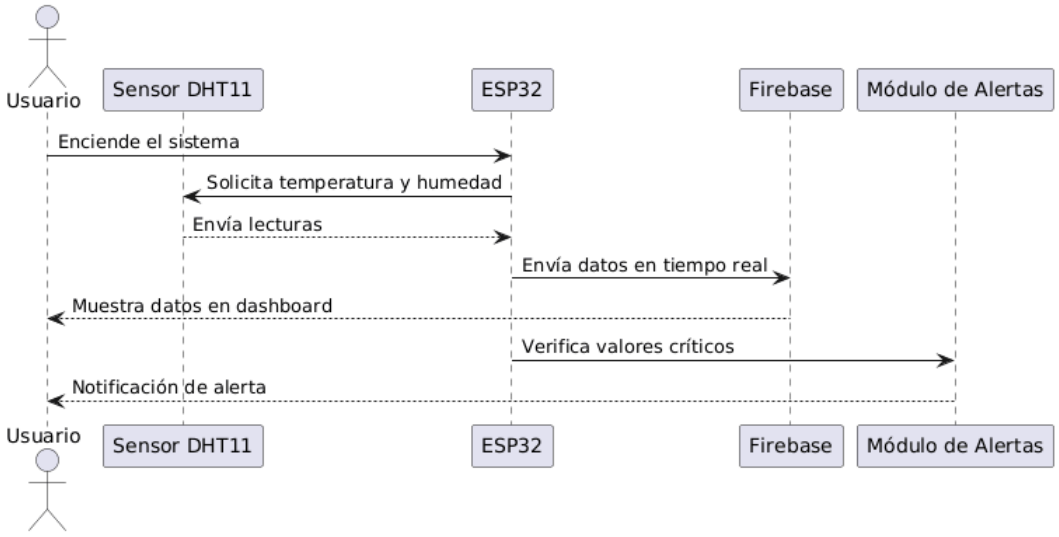

Figura 39 Diagrama de secuencia Sprint 1

6.2.2 Sprint 2 – Humedad del Suelo y Riego

- **Product Backlog asociado:**
  - Medir la humedad del suelo con el sensor FC-28.
  - Activar el riego automáticamente cuando el nivel sea bajo.
  - Permitir al usuario forzar el riego manualmente.
  - Calibrar los sensores de suelo.
- **Historia de usuario:**

"Como agricultor quiero saber el nivel de humedad del suelo, para que el sistema decida cuándo activar el riego automáticamente."

"Como agricultor quiero que los sensores estén calibrados, para obtener datos confiables del suelo."

- **Tarea en Jira (PH-3 y PH-7)** :
  - PH-3: Desarrollar código para lectura de humedad (FC-28) y envío a Firebase.
  - PH-7: Pruebas y calibración de los sensores FC-28.
- **Responsable:** Alejandro y Josue
- **Sprint Backlog:**
  - Conectar sensor FC-28 al ESP32.
  - Programar script de lectura y conversión a porcentaje.
  - Integrar con la bomba de agua para riego automático.
  - Ajustar rangos de calibración de humedad.
  - Visualizar datos en el dashboard.

- **Interacción:** Este script mide la humedad y la compara con umbrales definidos. La lectura se guarda en Firebase, lo que permite que el módulo de riego (bomba de agua) se active en interacción con este script. Además, el usuario puede forzar el riego manual desde la interfaz si es necesario.
- **Entregable (Incremento):**

  El sistema puede medir la humedad del suelo, activar el riego automático según condiciones y permitir calibración.

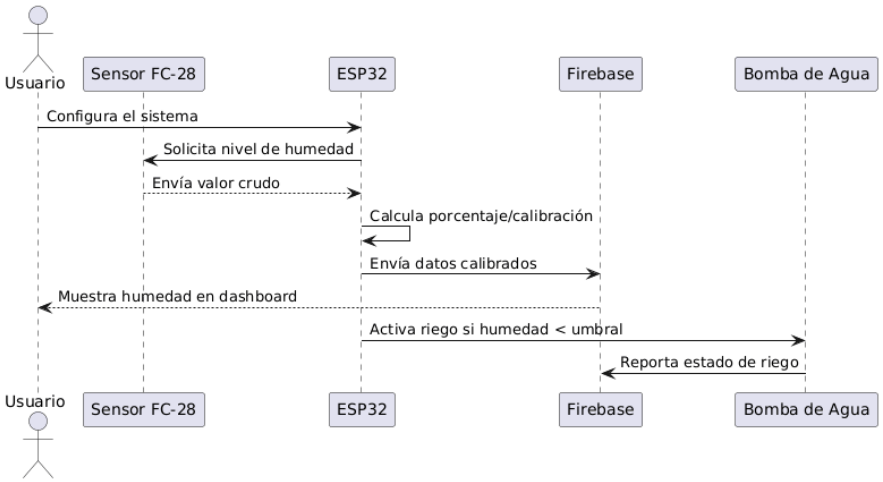

Figura 40 Diagrama de secuencia Sprint 2 (PH-3 + PH-7)

Fuente: Elaboración propia (2025)

6.2.3 Sprint 3 – Control de Iluminación

- **Product Backlog asociado:**
  - Medir la intensidad de luz con el sensor BH1750.
  - Encender o apagar tiras LED automáticamente según necesidad.
  - Enviar datos de luz a Firebase.
  - Notificar al usuario en caso de exceso o falta de luz.

- **Historia de usuario:** *"Como agricultor quiero que el sistema controle automáticamente la iluminación, para garantizar que mis plantas reciban la cantidad de luz adecuada."*
- **Tarea en Jira (PH-4):** "Desarrollar código para lectura de luz (BH1750) y control automático de tiras LED."
- **Sprint asignado:** Sprint 3 → Control del microclima (luz).
- **Responsable:** Maribel Y Luis Fernando
- **Sprint Backlog:**
  - Conectar el sensor BH1750 al ESP32.
  - Programar script de lectura de lux.
  - Configurar control de tiras LED según umbral.
  - Enviar estado y valores a Firebase.
  - Generar notificaciones de alerta.
- **Interacción:** El script lee el sensor BH1750 y enciende o apaga las tiras LED según los lux detectados. Envía el estado a Firebase, lo que actualiza el dashboard y permite al usuario ver si la iluminación está encendida. Interactúa con el módulo de notificaciones para avisar si hay exceso o falta de luz.
- **Entregable (Incremento):**

  El sistema controla la iluminación de forma automática y muestra en el dashboard los niveles de luz.

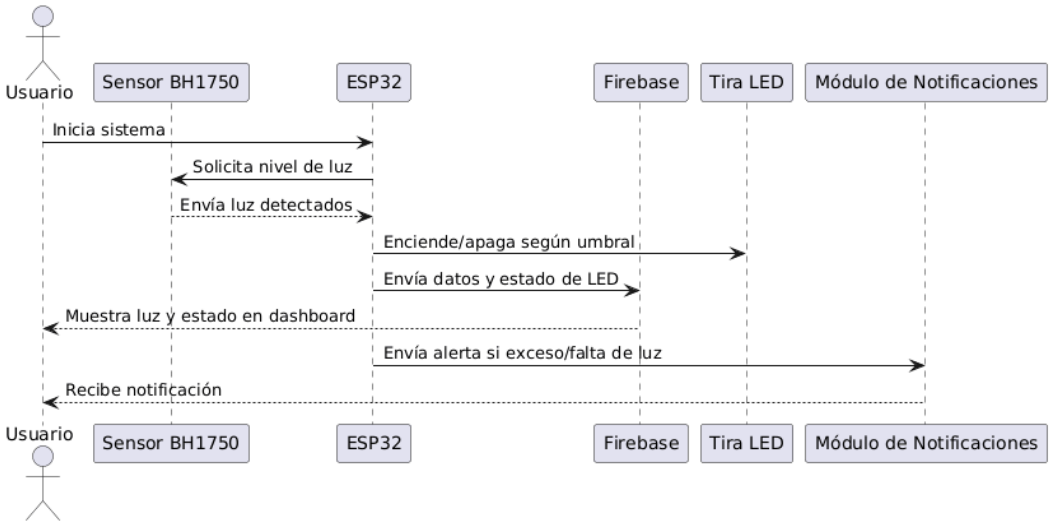

Figura 41 Diagrama de secuencia Sprint 3 (PH-4)

VII. DIAGRAMAS UML

- En este apartado se mostrara los diagramas UML usados así explicando cada proceso que realiza cada componente.

7.1 Diagrama de casos de uso (Interacción)

El diagrama de casos de uso representa las principales interacciones entre el Usuario **(**agricultor**)** y el *Sistema de Riego Multifunción*. Se identifican los actores externos (sensores, actuadores y servicio en la nube) y las funcionalidades clave que ofrece el sistema:

- **Monitoreo:** lectura de temperatura y humedad del aire (DHT11), humedad del suelo (FC-28) y nivel de luz (BH1750).
- **Acciones automáticas:** control del riego mediante la bomba de agua y regulación de iluminación con tiras LED.
- **Interfaz con el usuario:** visualización de datos en el dashboard y envío de notificaciones cuando las condiciones salen del rango óptimo.
- **Gestión en la nube:** los datos son registrados en Firebase**,** lo que permite la consulta remota y la integración con otros módulos del sistema.

Este diagrama permite comprender de manera global cómo se relacionan los distintos elementos del proyecto y cuáles son las funciones principales que garantizan el monitoreo y control automático del cultivo.

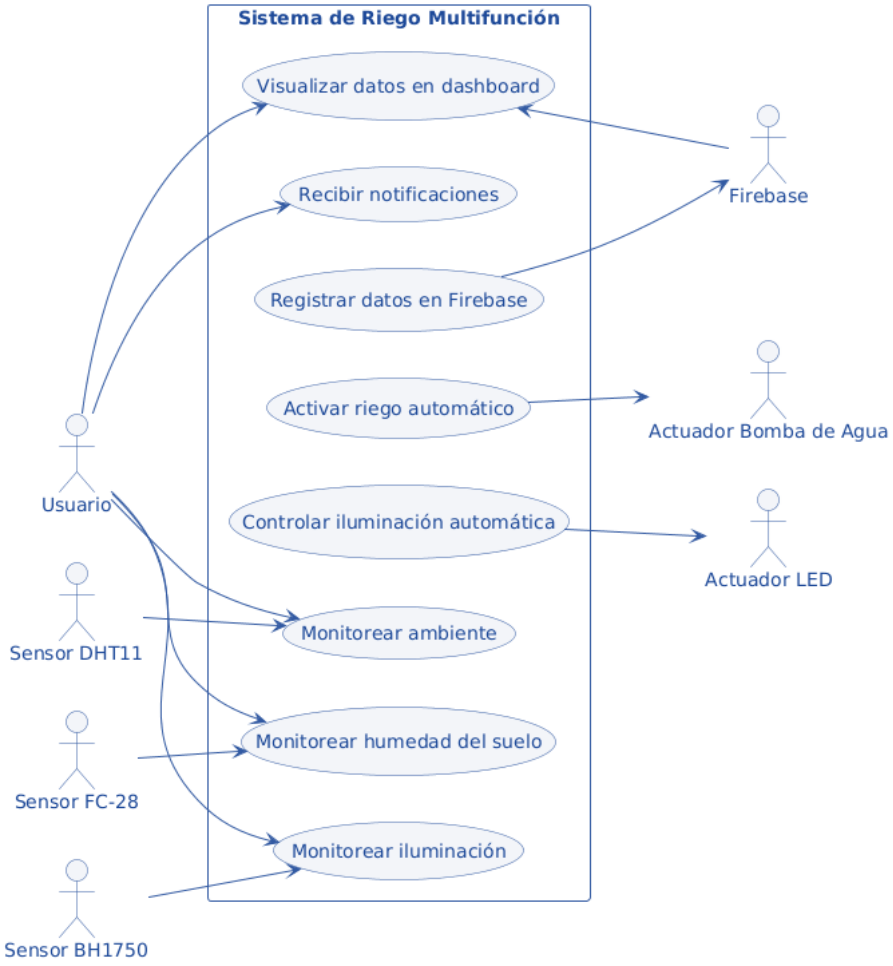

Figura 42 Diagrama de casos de Uso

7.2 Diagrama de clases (Estructura lógica)

El diagrama de clases del sistema de riego multifunción muestra la estructura lógica del proyecto, organizando los sensores, actuadores, controlador, servicio en la nube y usuario.

La clase Sensor funciona como superclase de los dispositivos de medición: DHT11 (temperatura y humedad del aire), FC-28 (humedad del suelo) y BH1750 (intensidad lumínica). De manera análoga, la clase Actuador generaliza los dispositivos de control, representados por la Bomba de Agua para el riego automático y el LED para la iluminación artificial.

En el centro se encuentra la clase ESP32Controller, que actúa como el núcleo del sistema. Sus funciones principales son establecer la conexión WiFi, leer datos de los sensores, enviarlos a la nube y activar los actuadores según las condiciones registradas.

El componente de nube está representado por la clase FirebaseService, encargada de registrar y proveer datos en tiempo real. Finalmente, la clase Usuario representa al agricultor, quien puede visualizar la información desde un dashboard y forzar manualmente acciones como el riego.

Este diagrama permite comprender la arquitectura lógica del sistema, mostrando la jerarquía de clases, las herencias y las relaciones de comunicación entre los elementos clave.

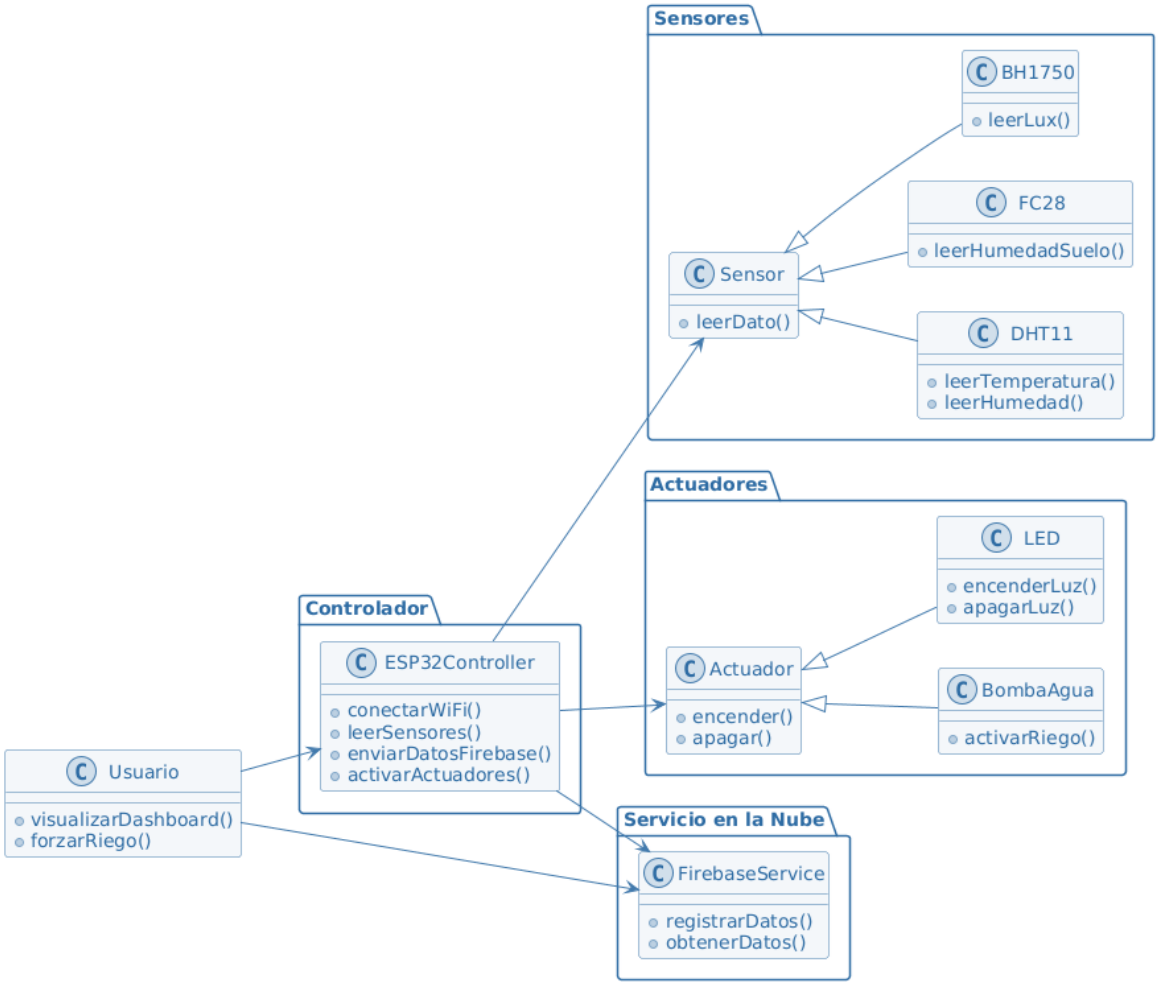

Figura 43 Diagrama de Clases

7.3 Diagrama de secuencia (Flujo de mensajes)

El diagrama de secuencia representa el flujo de mensajes paso a paso entre los distintos elementos del sistema.

En este caso se muestra el proceso de monitoreo de la humedad del suelo con el sensor FC-28. El sensor envía los datos al ESP32, que los transmite a Firebase. El agricultor puede visualizar los valores en el dashboard, y si el nivel de humedad es bajo, el ESP32 activa automáticamente la bomba de agua para realizar el riego. Finalmente, el estado actualizado se refleja en la nube para mantener informado al usuario.

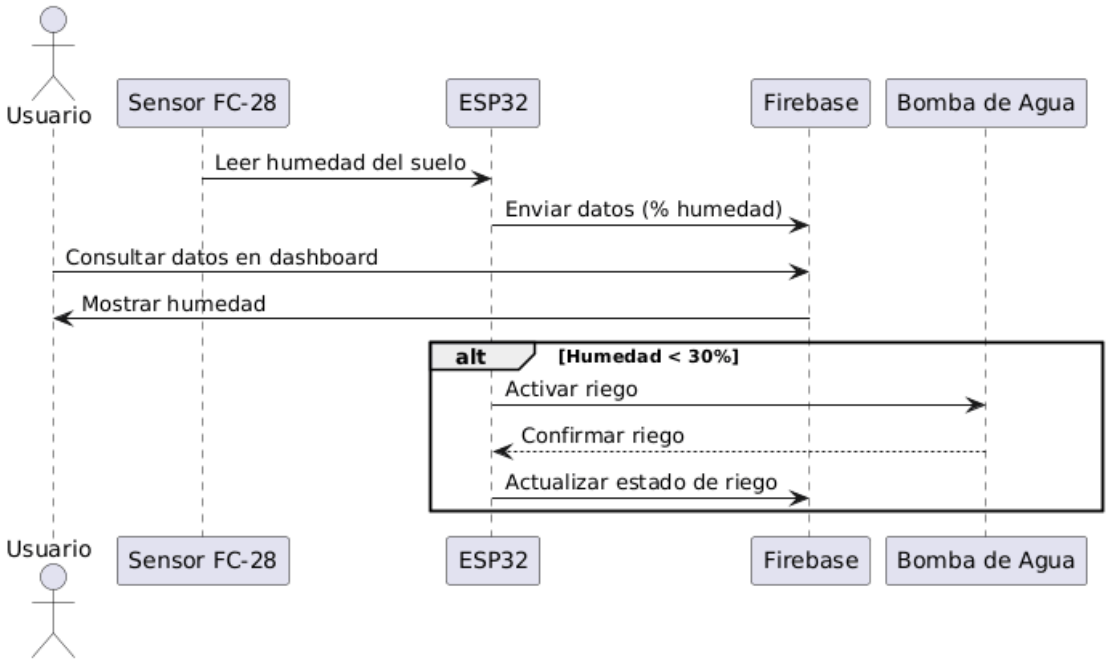

Figura 44 Diagrama de secuencian (Flujo de mensajes)

7.4 Diagrama de Actividades (Flujo de decicisiones)

El diagrama de actividades muestra el flujo lógico y las decisiones que toma el sistema en función de las condiciones detectadas por los sensores.

El proceso inicia con la lectura de los sensores (DHT11, FC-28 y BH1750). Los datos son enviados a Firebase y, dependiendo de si los valores se encuentran dentro o fuera de los rangos óptimos, el sistema activa los actuadores correspondientes: riego, iluminación o ventilación/calefacción. Si todo se encuentra en parámetros adecuados, el sistema continúa en modo de monitoreo.

Este flujo garantiza que las acciones automáticas se realicen solo cuando son necesarias, optimizando recursos y manteniendo la estabilidad del microclima.

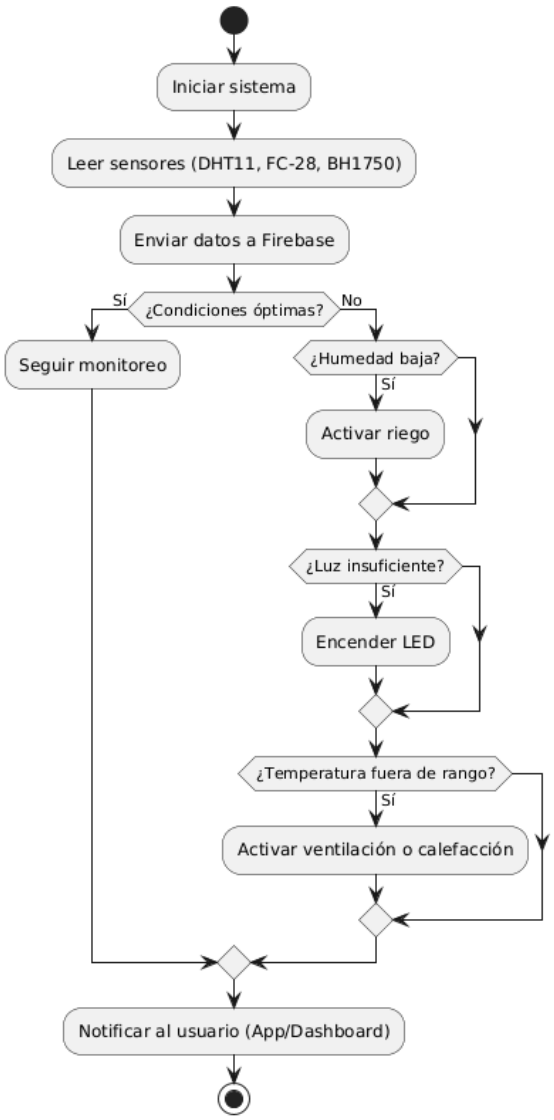

Figura 45 Diagrama de actividades (flujo de decisiones)

7.5 Diagrama de Componentes (Elementos físicos)

El diagrama de componentes describe cómo se relacionan los elementos físicos y lógicos del sistema de riego multifunción.

El ESP32 es el núcleo central que conecta los sensores (DHT11, FC-28, BH1750) y los actuadores (bomba de agua y tira LED RGB). A través de su módulo WiFi, se comunica con Firebase, que funciona como almacenamiento en la nube y centro de gestión de datos.

El dashboard web/móvil accede a Firebase para presentar al agricultor la información en tiempo real, permitiéndole monitorear el estado de los cultivos y si lo desea activar manualmente los actuadores.

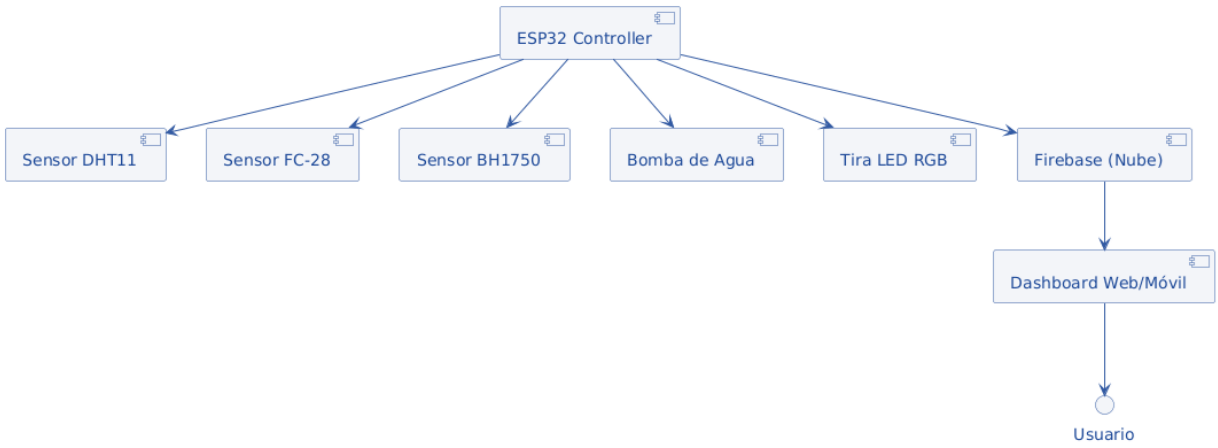

Figura 46 Diagrama de componentes (Elementos físicos)

7.6 Diagrama de Despliegue (Entorno del sistema)

El diagrama de despliegue muestra en qué entorno se ejecuta cada parte del sistema.

- En el dispositivo IoT (ESP32) se ejecutan los scripts en C++, que procesan los datos de los sensores y controlan los actuadores.
- En la nube de Firebase se encuentran la base de datos en tiempo real y el servicio de notificaciones, que almacenan la información y permiten la comunicación entre los distintos nodos.

- En el nodo del usuario (celular o PC) se despliega el dashboard web/móvil, desde donde el agricultor puede visualizar información, recibir alertas y tomar decisiones manuales.

Este modelo permite que el sistema sea distribuido, con cada parte trabajando en un entorno diferente, pero conectado en tiempo real mediante la nube.

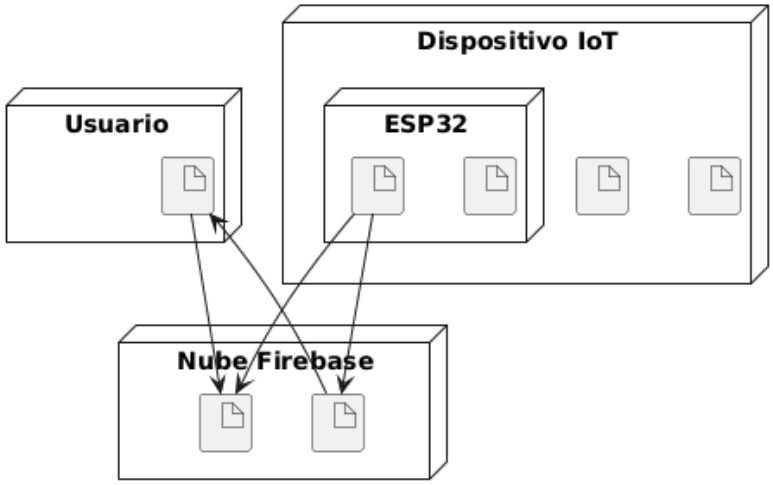

Figura 47 Diagrama de Despliegue (Entorno del sistema)

VIII. COSTOS

- En este apartado se tomara los gastos para el desarrollo del sistema se consideraron los gastos principales en materiales electrónicos, infraestructura básica y plantas. En la Tabla se detallan los costos estimados de cada componente.

Componente/Material	Cantidad	Precio Unitario (Bs)	Subtotal (Bs)
Modulo Relay (1 salida)	2	15,00	30,00
PC817	4	3,00	12,00
Modulo sensor de luz (BH1750)	1	30,00	30,00
Modulo DHT22 de humedad y temperatura	1	47,00	47,00
Modulo DHT11 de humedad y temperatura	1	20,00	20,00
Sensor de humedad de suelo FC-28	1	15,00	15,00
Relay	2	7,50	15,00
Modulo sensor DHT11 de temperatura y humedad	1	24,00	24,00
Mini bomba de agua sumergible	2	23,00	46,00
Modulo relay 1 salida	1	14,00	14,00
Cable	1	24,00	24,00
Ventilador	2	5,00	10,00
Modulo relay 1CH 5v	1	13,00	13,00
Bolsas de semilla	2	4,00	8,00
Dupons	varios	3,50	7,00
Plastaformo	2	10,00	20,00
Silicona	1	5,00	5,00
Docena de tornillos	12	5,00	5,00
Abono	2	10,00	10,00
Madera	2 de 90 cm, 4 de 70 cm, 6 de 40 cm	120,00	120,00
TOTAL:			= 475,OO

IX. CONCLUSIONES

Y RECOMENDACIONES

9.1 Conclusión

El sistema de riego multifunción propuesto logra automatizar y mantener un microclima estable en un huerto cerrado, integrando sensado en tiempo real (ESP32 ,Firebase), control de actuadores (riego, iluminación, ventilación) y una interfaz para supervisión y control manual. La métrica de "tiempo de clima" (horas en rango) y el uso de GDD permiten cuantificar y optimizar el desarrollo de cultivos sensibles al frío de La Paz. Con calibración adecuada de sensores y perfiles por cultivo (apio, perejil, hierba buena), el sistema mejora la estabilidad productiva y reduce pérdidas por variabilidad climática.

9.2 Recomendaciones

1) Uso de sensores precisos: Se recomienda emplear el sensor DHT22 en lugar del DHT11 por su mayor exactitud en la medición de temperatura y humedad.
2) Humedad del suelo: Utilizar sensores capacitivos en vez de resistivos, ya que son más duraderos y menos propensos a la corrosión.
3) Control del riego: Implementar un sistema de histeresis y temporización, evitando el encendido y apagado constante de la bomba.
4) Seguridad eléctrica: Incorporar protecciones básicas como fusibles y fuentes reguladas para evitar daños en los componentes.
5) Notificaciones y alertas: Configurar avisos en tiempo real (por ejemplo, vía Firebase) cuando la temperatura, humedad o humedad del suelo estén fuera de rango.
6) Pruebas previas: Realizar un periodo de calibración y pruebas de campo, para ajustar los umbrales de cada cultivo (apio, perejil y hierbabuena).
7) Escalabilidad: Considerar la posibilidad de ampliar el sistema con más sensores o módulos, para cubrir diferentes tipos de cultivos en el futuro.