



ACUSE REPORTE FINAL
INGENIERÍA MECATRÓNICA

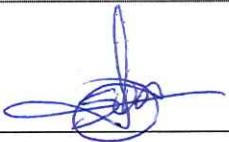
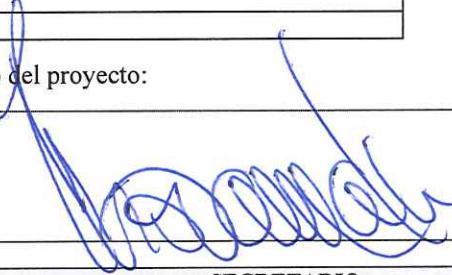
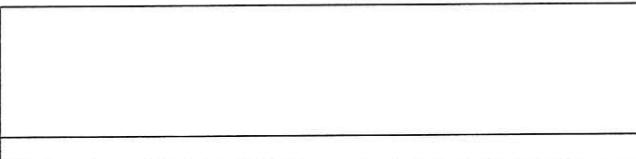
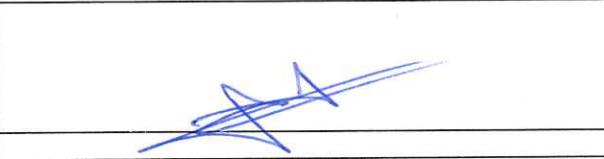
El presente documento hace constar que se ha entregado el Reporte Final de Trabajo Terminal II al Jurado para su defensa en la presentación ETS, del proyecto denominando:

NÚMERO DE REGISTRO (Número asignado en la Constancia de Registro de Protocolo)	TTM-2025/2-3
TÍTULO	
Sistema de monitoreo térmico periódico para un techo verde ligero extensivo	

Proyecto integrado por los siguientes alumnos:

NOMBRE COMPLETO	
Alumno 1	Garduño Jardón Sergio
Alumno 2	Noguerón Soto Hugo de Jesús
Alumno 3	
Alumno 4	

A continuación firman de recibido todos los miembros del Jurado del proyecto:

	
PRESIDENTE M. en C. Sergio Viveros Bretón	SECRETARIO Dr. Rafael Trovamala Landa
	
ASESOR 1 M. en C. Niels Henrik Navarrete Manzanilla	ASESOR 2
Interno <input checked="" type="checkbox"/> Externo <input type="checkbox"/>	Interno <input type="checkbox"/> Externo <input type="checkbox"/>
	
ASESOR 3	SUPLENTE M. en A. Armando Guzmán Álvarez
Interno <input type="checkbox"/> Externo <input type="checkbox"/>	

NOTAS IMPORTANTES:

- El Acuse de Reporte Final deberá entregarse en original al Secretario, este deberá incluir todas las firmas de los miembros del Jurado, incluyendo al Suplente, sin excepciones, en caso contrario el Departamento de Tecnologías Avanzadas no podrá programar la presentación del proyecto.
- Los alumnos tendrán la responsabilidad de notificar a todo el Jurado, incluyendo al Suplente, en caso de que el proyecto no se presentará en la evaluación correspondiente, esto lo deberán realizar con al menos dos días de anticipación a la fecha programada.
- En caso de tener menos de tres asesores, deberá dejar el espacio vacío.



INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN
INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Trabajo Terminal II

**“Sistema de Monitoreo térmico periódico
para un techo verde ligero extensivo”**

Presentan:

Garduño Jardón Sergio
Noguerón Soto Hugo de Jesús

Asesores:

M. en C. Niels Henrik Navarrete Manzanilla
M. en C. Ana Laura Cervantes Nájera

Ciudad de México, Junio 2025

Resumen

El presente documento describe el desarrollo e implementación de un sistema de monitoreo térmico automatizado, diseñado para evaluar el comportamiento térmico de un techo verde experimental instalado sobre una caseta de vigilancia en las instalaciones del CIEMAD-IPN. Este sistema tiene como propósito principal registrar y analizar la temperatura del sustrato en distintas zonas del techo, con el fin de comparar el efecto térmico de la cobertura vegetal frente a áreas sin vegetación.

Para ello, se integraron ocho sensores digitales de temperatura. Siete de estos sensores fueron colocados bajo diferentes especies de suculentas, y uno se ubicó en una zona expuesta sin cobertura vegetal, lo que permite establecer comparativas directas entre ambos tipos de superficie. El registro de datos se realiza de manera automatizada cada determinado tiempo, asegurando una frecuencia de muestreo adecuada para observar variaciones térmicas a lo largo del día.

El sistema es controlado por un microcontrolador FireBeetle ESP32-E, el cual gestiona las lecturas de los sensores mediante una máquina de estados programada para activar la lectura de bloques de sensores de manera secuencial. Los datos obtenidos son almacenados temporalmente en la memoria interna del microcontrolador y transmitidos, siempre que haya conexión disponible, a una hoja de cálculo mediante conectividad Wi-Fi. En caso de pérdida de red, el sistema cuenta con una lógica de respaldo que garantiza la conservación local de los datos hasta su posterior sincronización.

La alimentación del sistema se realiza mediante una celda solar de 5V y una batería recargable LiPo de 3.7V, lo cual permite una operación autónoma y continua las 24 horas del día. Este diseño aprovecha el módulo de carga solar integrado en la propia placa FireBeetle, eliminando la necesidad de componentes adicionales para la gestión energética. Además, se contempla una alternativa de alimentación directa como respaldo, con el objetivo de aumentar la confiabilidad del sistema en escenarios experimentales prolongados.

Este sistema fue desarrollado con fines de investigación aplicada, en colaboración con el CIEMAD, como parte de un esfuerzo por generar evidencia técnica sobre el impacto de los techos verdes en la mitigación del efecto isla de calor urbano.

Palabras clave

Techo verde, monitoreo térmico, cobertura vegetal, Wi-Fi.

Abstract

This document describes the development and implementation of an automated thermal monitoring system, designed to evaluate the thermal behavior of an experimental green roof installed over a guard house at CIEMAD-IPN facilities. The main purpose of this system is to record and analyze the substrate temperature in different areas of the roof, in order to compare the thermal effect of the vegetation cover versus areas without vegetation.

For this purpose, eight DS18B20 digital temperature sensors were integrated, distributed over a 3×2 meter surface. Seven of these sensors were placed under different species of succulents, and one was located in an exposed area without vegetation cover, allowing direct comparisons between the two types of surface. Data recording is automated every 30 minutes, ensuring an adequate sampling frequency to observe thermal variations throughout the day.

The system is controlled by a FireBeetle ESP32-E microcontroller, which manages sensor readings through a state machine programmed to trigger the reading of sensor blocks sequentially. The data obtained is temporarily stored in the microcontroller's internal memory and transmitted, provided a connection is available, to a spreadsheet via Wi-Fi connectivity. In case of network loss, the system has a backup logic that guarantees the local preservation of the data until its subsequent synchronization.

The system is powered by a 5 V solar cell and a 3.7 V LiPo rechargeable battery, allowing autonomous and continuous operation 24 hours a day. This design takes advantage of the solar charging module integrated into the FireBeetle board itself, eliminating the need for additional power management components. In addition, a direct power supply alternative is contemplated as a backup, with the objective of increasing the reliability of the system in prolonged experimental scenarios.

This system was developed for applied research purposes, in collaboration with CIEMAD, as part of an effort to generate technical evidence on the impact of green roofs in mitigating the urban heat island effect.

Key words

Green roof, thermal monitoring, vegetation cover, Wi-Fi.

Índice general

Resumen	3
Abstract	5
Introducción	1
1. Panorama General	3
Planteamiento del Problema	4
1.1. Justificación	4
1.2. Objetivos	5
1.2.1. Objetivo general	5
1.2.2. Objetivos específicos	5
Antecedentes	6
Propuesta de Solución	8
2. Marco Referencial	11
Marco teórico	12
2.0.1. Urbanización, cambio climático y retos térmicos actuales	12
2.0.2. Techos verdes: definición, tipología y beneficios	12
2.0.3. Isla de calor urbana y techos verdes como solución térmica	13
2.0.4. Importancia del monitoreo térmico localizado	13
2.0.5. Fundamentos de IoT	13
2.0.6. Tecnologías para la conectividad	15
2.0.7. Microcontroladores para Sistemas IoT	16
2.0.8. Sensores ambientales	16
2.0.9. Sensor de temperatura	17
2.0.10. Plataformas IoT	17

2.0.11. Tipos de plataformas IoT	18
2.0.12. Ejemplos de plataformas IoT	18
2.0.13. Microcontroladores e IoT en sistemas ambientales	18
2.0.14. Especies suculentas utilizadas y clasificación por sensor	19
Marco Procedimental	19
Definición de la metodología mecatrónica	19
Método de objetivos ponderados	21
3. Diseño del Sistema	23
Necesidades	24
Arquitectura funcional	24
Propuesta de diseño	28
4. Diseño en el Dominio Específico	31
4.1. Componentes mecánicos	32
4.1.1. Análisis de tensiones en el chasis de protección	32
4.1.2. Análisis de desplazamiento estático del chasis	33
4.1.3. Análisis de deformación unitaria estática del chasis	35
4.1.4. Cumplimiento de requisitos ambientales: agua y temperatura	36
4.2. Sistema eléctrico - electrónico	38
4.2.1. Selección de componentes mediante matriz de decisión	38
4.2.2. Diagrama Electrónico y Conexiones	40
4.3. Sistema eléctrico-electrónico y cálculos de dimensionamiento	41
4.3.1. Presupuesto de corriente y potencia	41
4.3.2. Autonomía con batería	41
4.3.3. Dimensionamiento del panel solar	42
4.3.4. Medición analógica de batería y panel	42
4.3.5. Ley de Ohm y divisor de tensión (recordatorio)	43
4.3.6. Energía, consumo por evento y muestreo	43
4.3.7. Accionamiento del relevador y protección	43
4.3.8. Integridad de alimentación y desacoplo	44
4.3.9. Bus I ² C y 1-Wire: resistencias de <i>pull-up</i>	44
4.3.10. Caída de tensión en cables y protección	44
4.3.11. Resumen de cumplimiento eléctrico	44
4.4. Sistema de control basado en máquina de estados	44
4.4.1. Definición formal	45
4.4.2. Guardas y reglas de control	45
4.4.3. Tabla de transiciones (resumen)	46
4.4.4. Propiedades (invariantes y liveness)	46
4.4.5. Temporización del ciclo diurno	46
4.4.6. Pseudocódigo de implementación (C++/ESP32)	46

4.4.7. Plan de verificación	47
4.4.8. Gestión de energía por estado	48
4.4.9. Manejo de fallos	48
4.5. Diseño del sistema de interfaz web (IoT)	48
4.5.1. Objetivos de la HMI web	48
4.5.2. Arquitectura	48
4.5.3. Modelo de datos (JSON)	49
4.5.4. End-points del backend (GAS)	49
4.5.5. Actualización “casi en tiempo real”	49
4.5.6. Diseño de UI (vistas)	49
4.5.7. Accesibilidad y respuesta	49
4.5.8. Métricas y rendimiento	50
4.5.9. Pruebas	50
5. Implementación y Validación	51
5.0.1. Implementación del hardware (esquemáticos y PCB)	52
5.0.2. Bloques implementados en la tarjeta	52
5.0.3. Esquemáticos por bloques	54
5.0.4. Pinout y conectores de la PCB	55
5.0.5. Inclusión de Librerías y Credenciales	55
5.0.6. Estructura de Comunicación y Envío de Datos	56
5.0.7. Definición de Pines, Sensores y Variables Globales	58
5.0.8. Pantalla OLED y dimensiones	58
5.0.9. Entradas del encoder y pulsador	58
5.0.10. Puntos de medición analógicos (panel y batería)	59
5.0.11. Bus 1-Wire de sensores DS18B20	59
5.0.12. Sensores ambientales por I ² C	60
5.0.13. Relevador y umbrales de protección	60
5.0.14. Reloj de tiempo real (RTC) y ventanas día/noche	61
5.0.15. Estado global y persistencia entre deep sleeps	62
5.0.16. Resumen visual del bloque	62
5.1. Funciones Auxiliares y Utilidades	63
5.1.1. Codificación de texto para envío HTTP	63
5.1.2. Conversión de índice del encoder a minutos/horas	63
5.1.3. Lectura de BMP280 (temperatura y presión)	64
5.1.4. Lectura de BH1750 (intensidad lumínica)	64
5.1.5. Persistencia de respaldos (LittleFS)	64
5.2. Manejo de Pantalla OLED	65
5.2.1. Encendido y apagado del display	65
5.2.2. Visualización del tiempo seleccionado	66

5.2.3. Páginas de sensores DS18B20	67
5.2.4. Páginas de sensores ambientales	67
5.2.5. Página de voltajes y estado del relevador	68
5.2.6. Página de envío de datos	69
5.3. Control del Relevador y Gestión Energética	70
5.3.1. Función principal de control	71
5.3.2. Variables de soporte	71
5.3.3. Interacción con el ciclo de trabajo	72
5.3.4. Visualización en OLED	72
5.4. Modo Noche	73
5.4.1. Detección del modo noche	73
5.4.2. Ejecución de la rutina nocturna	73
5.4.3. Explicación de la lógica	74
5.4.4. Visualización en OLED	75
5.5. Modo Día y Ciclo de 6 Partes	76
5.5.1. División en partes	76
5.5.2. Ejecución del ciclo en el <code>loop()</code>	76
5.5.3. Explicación de la lógica	78
5.5.4. Ventanas de visualización OLED	78
5.6. Manejo del Botón y Encoder	79
5.6.1. Lectura del giro del encoder	79
5.6.2. Lectura del botón integrado	79
5.6.3. Confirmación del tiempo seleccionado	80
5.6.4. Visualización en OLED	81
5.7. Integración con Google Apps Script y Google Sheets	81
5.7.1. Flujo de recepción	81
5.7.2. 4.11.2 Estructura del <code>doPost(e)</code>	82
5.7.3. 4.11.3 Respuesta del servicio	84
5.7.4. 4.11.4 Despliegue del Web App	84
5.7.5. 4.11.5 Seguridad y buenas prácticas	84
5.7.6. 4.11.6 Pruebas de integración	85
5.7.7. 4.11.7 Manejo de registros provenientes de respaldo (LittleFS)	85
5.7.8. 4.11.8 Buenas prácticas de hoja de cálculo	85
5.7.9. 4.11.9 Diagnóstico de fallos comunes	86
5.7.10. 4.11.10 Posibles mejoras futuras	86
5.8. 4.12 Interfaz OLED y Navegación por Páginas	86
5.8.1. 4.12.1 Inicialización y control de energía	87
5.8.2. 4.12.2 Navegación por páginas	87
5.8.3. 4.12.3 Página de selección de tiempo	88
5.8.4. 4.12.4 Página de sensores DS18B20	88

5.8.5. 4.12.5 Página de sensores ambientales	89
5.8.6. 4.12.6 Página de voltajes y relevador	89
5.8.7. 4.12.7 Página de envío	90
5.8.8. 4.12.8 Mensajes especiales	90
5.9. 4.13 Manejo del RTC y Control Horario	90
5.9.1. 4.13.1 Inicialización del RTC	91
5.9.2. 4.13.2 Definición de intervalos de día y noche	91
5.9.3. 4.13.3 Transiciones entre modos	91
5.9.4. 4.13.4 Marca temporal en los registros	92
5.9.5. 4.13.5 Ejemplo de cronograma horario	93
5.10. 4.14 Control del Relevador	93
5.10.1. 4.14.1 Configuración básica	93
5.10.2. 4.14.2 Lógica de decisión	94
5.10.3. 4.14.3 Interacción con el ciclo de 6 partes	94
5.10.4. 4.14.4 Visualización en la OLED	94
5.10.5. 4.14.5 Ventajas del diseño	95
5.11. 4.15 Conclusiones del Capítulo de Programación	95
Cronograma seguido para TTII	99
6. Conclusiones Finales	101
6.1. Trabajo Futuro	102
Apéndices	103
Apéndice Código de la lógica ESP32 FireBeetle	104
Apéndice D. Repositorios de GitHub del sistema Web IoT	132

Índice de figuras

1.1. Principios básicos del urbanismo verde	6
1.2. Valores promedio mensuales, temperatura máxima y mínima del aire y radiación solar promedio.	7
1.3. Diseño electrónico del sistema del bioreactor.	7
2.1. Comparativa térmica entre un techo convencional y un techo verde.	12
2.2. Arquitectura de tres capas para aplicaciones IoT.	14
2.3. Principales elementos en un sistema IoT.	15
2.4. Metodología V para diseño en ingeniería mecatrónica	20
3.1. Diagrama FBS del sistema de monitoreo térmico en techo verde	26
3.2. Arquitectura física del sistema	28
3.3. Diseño Conceptual	29
4.1. Análisis estático: Tensión Von Mises	33
4.2. Análisis estático: Desplazamiento estático	34
4.3. Análisis estático: Deformación unitaria	35
5.1. Vista superior de la PCB final (100 mm × 100 mm). Se aprecian: FireBeetle ESP32 (centro), cabeceras para BMP280 y BH1750 (superior-izquierda), cabecera <i>RELOJ RTC</i> (vertical al centro-izquierda), cabecera del <i>encoder</i> (superior-derecha), zona de <i>relé</i> y su driver (derecha) y ocho conectores S1–S8 para sondas DS18B20 (lados izquierdo y derecho).	52
5.2. Esquemático	54
5.3. PCB	54
5.4. Esquemáticos.	54
5.5. Comunicación ESP32 → WiFi → Google Apps Script → Google Sheets . .	56

5.6. Captura de Google Sheets mostrando datos enviados correctamente desde el ESP32.	58
5.7. Divisor de tensión y entrada ADC.	59
5.8. Bus I ² C compartido para AHT10, BMP280, BH1750 y OLED.	60
5.9. Decisión del relevador con umbrales V _{BAT} y V _{PANEL}	61
5.10. División en 6 partes del ciclo diurno y ventanas de visualización OLED.	62
5.11. Rango de tiempos posibles según posición del encoder.	64
5.12. Flujo de respaldo y reintento: Normal → Sin WiFi → LittleFS → Reintento → Envío exitoso.	65
5.13. OLED mostrando el menú de selección de tiempo.	66
5.14. Pantalla OLED mostrando lecturas ambientales.	68
5.15. Voltajes del prototipo en funcionamiento.	69
5.16. Decisión del relevador: si es modo día, $V_{bat} < 3,6$ V y $V_{panel} > 3,0$ V → Rele ON; en otro caso → Rele OFF.	72
5.17. Diagrama de flujo del modo noche: Lectura → Envío/Respaldo → Deep Sleep.	75
5.18. Las 6 partes del ciclo diurno, ventanas OLED al inicio y al final, y acción principal de cada parte.	78
5.19. Mensaje de “Respaldo borrado” mostrado en la pantalla OLED tras un hold largo.	81
5.20. Diagrama de flujo ESP32 -> WEB APP (GAS).	82
5.21. Hoja ”4.0” mostrando encabezados A-V	83
5.22. Implementación del Web App.	84
5.23. Captura de filtros y gráficos propuestos.	86
5.24. Línea de tiempo del sistema. Transiciones día-noche y acciones clave.	93
5.25. Diagrama de decisión. Tipo Árbol	95

Índice de tablas

1.1. Comparación de estudios previos relacionados con techos verdes y monitoreo ambiental.	8
2.1. Clasificación de especies suculentas monitoreadas por sensor	19
3.1. Requerimientos del sistema de monitoreo térmico en techo verde	24
4.1. Propiedades mecánicas típicas del ABS (Acrilonitrilo Butadieno Estireno).	32
4.2. Matriz de decisión: microcontrolador (pesos entre paréntesis).	38
4.3. Matriz de decisión: sensor de temperatura puntual.	38
4.4. Matriz de decisión: interfaz de visualización.	39
4.5. Matriz de decisión: actuador de potencia.	39
4.6. Matriz de decisión: cargador Li-ion.	40
4.7. Matriz de decisión: regulador a 3.3 V.	40
4.8. Transiciones principales de la FSM.	46
5.1. Pinout resumido de conectores principales.	55
5.2.	83
5.3. Cronograma de actividades para TTII	100

Introducción

El crecimiento acelerado de las ciudades ha generado una transformación significativa del entorno natural, dando lugar a un aumento progresivo de las temperaturas en áreas urbanas, fenómeno conocido como isla de calor urbana (ICU). Este efecto es ocasionado por la reducción de superficies vegetadas y la predominancia de materiales constructivos con alta capacidad de absorción térmica. Como consecuencia, se eleva el consumo energético, se deteriora la calidad del aire y se reducen las condiciones de confort térmico en las zonas densamente habitadas [1].

Ante esta problemática, los techos verdes han sido identificados como una solución sustentable capaz de mitigar el efecto de la ICU. Estos sistemas reducen la temperatura superficial mediante aislamiento térmico y evapotranspiración, llegando a disminuir entre 0.3°C y 3°C la temperatura ambiental inmediata, e incluso hasta 15°C en interiores [2] [3] [4]. Adicionalmente, brindan beneficios ambientales como la reducción de contaminantes, mejora de la calidad del aire, manejo del agua pluvial y aumento de la biodiversidad urbana [5] [6]

Para evaluar con precisión estos beneficios térmicos, es indispensable contar con sistemas de monitoreo que proporcionen datos confiables bajo condiciones reales. El presente proyecto propone el diseño de un sistema de monitoreo térmico ambiental basado en el FireBeetle ESP32-E y sensores DS18B20, instalados en un techo verde sobre una caseta de policía, como parte de una estrategia de evaluación térmica local. Este sistema, permitirá la adquisición, almacenamiento y transmisión de datos térmicos, apoyando la validación del impacto de techos verdes en contextos urbanos específicos [7] [8].

CAPÍTULO 1

Panorama General

Planteamiento del Problema

El incremento de las temperaturas en entornos urbanos, consecuencia directa del reemplazo de áreas verdes por superficies construidas, ha intensificado el fenómeno conocido como isla de calor urbana (ICU). Este efecto térmico genera impactos significativos en la salud pública, la calidad del aire y el consumo energético en zonas densamente edificadas [1],[9]

Si bien los techos verdes se han propuesto como una solución viable para mitigar este fenómeno, su desempeño térmico real puede variar ampliamente dependiendo del tipo de vegetación, las condiciones climáticas locales, la orientación del edificio y la calidad del sustrato [2] [6]. En muchos casos, no se cuenta con datos empíricos suficientes para validar su efectividad bajo condiciones específicas, lo cual dificulta su integración sistemática en políticas urbanas o estrategias de diseño arquitectónico.

Diversas investigaciones han señalado la carencia de datos térmicos localizados y continuos como una de las principales limitaciones para optimizar la implementación de techos verdes [10] [11]. Esta necesidad se vuelve aún más crítica en contextos reales, como instalaciones públicas o comunitarias, donde la toma de decisiones debe estar respaldada por evidencia cuantificable.

Ante esta situación, surge la oportunidad de implementar un sistema de monitoreo térmico basado en tecnologías de bajo costo y fácil despliegue, como el ESP32 y sensores DS18B20, capaces de registrar temperaturas en tiempo real con alta precisión. El presente proyecto plantea su instalación en una caseta de policía ubicada dentro del Centro Interdisciplinario de Investigaciones y Estudios sobre Medio Ambiente y Desarrollo (CIEMAD) del IPN, lo que permite estudiar su comportamiento térmico bajo condiciones reales y controladas de exposición solar.

El sistema busca generar datos objetivos que contribuyan a la validación térmica del techo verde y, a su vez, sentar las bases para el desarrollo de soluciones replicables en entornos urbanos, aprovechando tecnologías abiertas y energías renovables como parte de una estrategia de infraestructura verde más inteligente.

1.1. Justificación

[10] [12]. En este sentido, el desarrollo de sistemas de monitoreo térmico autónomos, accesibles y escalables se vuelve prioritario para respaldar decisiones de infraestructura verde.

En el presente proyecto, se propone un sistema de instrumentación electrónica basado en FireBeetle ESP32-E y sensores DS18B20, alimentado por energía solar, para registrar la temperatura en zonas con y sin cobertura vegetal dentro de un techo verde experimental. Este se encuentra instalado sobre una caseta de policía urbana, lo que otorga al proyecto un valor adicional al permitir observar su comportamiento en un entorno real, de acceso comunitario.

Además, ya se han realizado mediciones preliminares de radiación solar con un medidor SM206-Solar en distintas especies suculentas distribuidas en la superficie del techo verde, como Sedum pachyphyllum, S. nussbamerianum y S. dendroideum, lo que aporta un contexto técnico sólido al sistema de monitoreo planteado. Dichos datos serán integrados en el apartado de anexos para respaldar las decisiones de diseño y la selección de puntos de muestreo térmico.

El sistema propuesto representa una herramienta de bajo costo, autónoma y de fácil implementación, útil tanto para fines educativos como para investigación aplicada, con potencial de replicabilidad en otras infraestructuras urbanas de características similares.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar e implementar un sistema de monitoreo térmico periódico para un techo verde ligero extensivo que permita registrar la temperatura del sustrato en secciones asociadas a siete especies distintas de suculentas, enviando la información recolectada a una base de datos para su consulta desde cualquier dispositivo con conexión a internet.

1.2.2. Objetivos específicos

- Seleccionar un chasis que contenga los componentes electrónicos del sistema, exceptuando los sensores, con resistencia a las precipitaciones y temperaturas elevadas.
- Diseñar una estructura de montaje que permita el anclaje del chasis y la distribución de los sensores en los puntos a medir del techo verde ligero extensivo.
- Validar que el chasis diseñado permita el correcto funcionamiento de los componentes sin que se vean interferidos por precipitaciones o temperatura elevada.
- Implementar una máquina de estados que controle la lectura secuencial de los sensores de temperatura.

- Incorporar lógica de temporización adaptable que permita modificar el intervalo de muestreo térmico según las necesidades del usuario.
- Seleccionar los sensores de temperatura y los módulos electrónicos necesarios para la adquisición de los datos de temperatura en cada uno de los sustratos de las siete especies de suculentas.
- Diseñar un sistema de alimentación energética, tomando la red de distribución eléctrica o incorporando una batería recargable con panel solar, asegurando la autonomía operativa del sistema en entornos exteriores.
- Implementar un sistema de comunicación y/o almacenamiento de datos, que permita enviar los registros térmicos a una base de datos en línea accesible desde cualquier dispositivo con conexión a internet.
- Diseñar una interfaz gráfica para el usuario que permita visualizar los datos de las temperaturas en los distintos sustratos del techo verde ligero extensivo.

Antecedentes

Durante los últimos años, la implementación de techos verdes como solución para mitigar el efecto de isla de calor urbana ha sido ampliamente estudiada. Investigaciones como las de [3] y [4] han demostrado reducciones significativas de temperatura superficial, así como beneficios complementarios en confort térmico y eficiencia energética.



Figura 1.1: Principios básicos del urbanismo verde
Obtenida de [3].

Para evaluar estos efectos, diferentes estudios han empleado sensores térmicos y sistemas de monitoreo ambiental. Por ejemplo, [10] documentaron variaciones térmicas utilizando sensores distribuidos sobre techos verdes en clima mediterráneo. No obstante,

muchos de estos trabajos utilizan equipos de alto costo o requieren infraestructura compleja.

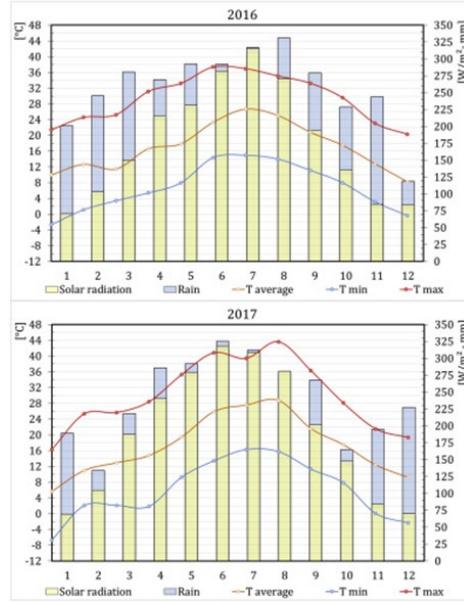


Figura 1.2: Valores promedio mensuales, temperatura máxima y mínima del aire y radiación solar promedio.

Obtenida de [10].

Con el avance de la tecnología, han surgido alternativas más accesibles. Estudios como los de [7] y [8] emplean microcontroladores ESP32 y sensores DS18B20 para la recolección de datos ambientales, destacando por su bajo consumo energético, conectividad Wi-Fi y facilidad de integración. Estos sistemas han sido aplicados principalmente en laboratorios, bioprocessos o monitoreo de calidad del agua.

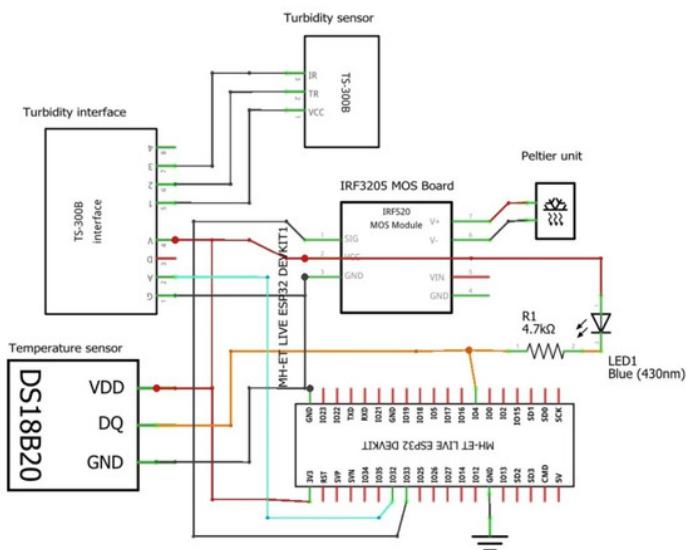


Figura 1.3: Diseño electrónico del sistema del bioreactor.

Obtenida de [7].

A pesar de estos avances, aún existen pocos desarrollos que integren un sistema totalmente autónomo, alimentado por energía solar, de bajo costo, y orientado a comparar térmicamente zonas vegetadas y no vegetadas en techos verdes de pequeña escala. Tampoco es frecuente encontrar aplicaciones implementadas en entornos urbanos reales.

La tabla 1.1 presenta una comparativa entre los distintos productos y proyectos antes presentados.

Tabla 1.1: Comparación de estudios previos relacionados con techos verdes y monitoreo ambiental.

No.	Autor y año	Enfoque del estudio	Tecnología usada	Enlace
1	Cuce et al. (2025)	Reducción térmica y beneficios ambientales de techos verdes	Evaluación térmica con sensores	https://www.mdpi.com/2071-1050/17/3/1303
2	Asadi et al. (2020)	Simulación de techos verdes para mitigación del efecto isla de calor	Red neuronal + simulación térmica	https://www.sciencedirect.com/science/article/abs/pii/S0273117720304737?via%3Dihub
3	Bevilacqua et al. (2020)	Medición térmica en techos verdes en clima mediterráneo	Sensores térmicos distribuidos	https://www.sciencedirect.com/science/article/abs/pii/S096014812030104X?via%3Dihub
4	Baicu et al. (2024)	Monitoreo ambiental con ESP32 en biorreactores	ESP32 + sensores DS18B20	https://www.mdpi.com/1424-8220/24/20/6587
5	Padma et al. (2023)	Monitoreo ambiental en calidad del agua	ESP32 + sensores integrados	https://www.ijraset.com/best-journal/iot-based-water-qual-monit-sys
6	Akhie & Joksimovic (2023)	Monitoreo de techo productivo con sensores de bajo costo	Red de sensores económicos	https://www.mdpi.com/1424-8220/23/24/9788
7	Lambarki et al. (2024)	Evaluación térmica y económica con GIS	Ánálisis geoespacial	https://www.sciencedirect.com/science/article/abs/pii/S2325426224000688
8	Li et al. (2024)	Predicción térmica con aprendizaje profundo híbrido	IA basada en grafos	https://www.sciencedirect.com/science/article/abs/pii/S2352710224001220?via%3Dihub
9	Wang et al. (2024)	Predicción térmica avanzada en techos verdes	CNN-LSTM + SSA	https://www.sciencedirect.com/science/article/abs/pii/S0378778824008612?via%3Dihub
10	Masali et al. (2024)	Sistemas solares autónomos con ESP32	ESP32 + Energía solar	https://ojs.ukscip.com/index.php/neea/article/view/274

Propuesta de Solución

Con el fin de abordar las limitaciones actuales en la recolección y análisis de datos térmicos en techos verdes extensivos, se propone el diseño e implementación de un sistema mecatrónico de monitoreo térmico automatizado y autónomo, capaz de registrar, almacenar y transmitir datos de temperatura del sustrato

en intervalos regulares.

El sistema estará compuesto por un conjunto de sensores digitales de temperatura tipo DS18B20, seleccionados por su precisión, bajo consumo energético y facilidad de integración mediante comunicación digital. Estos sensores se distribuirán estratégicamente en las secciones del techo verde, cada una correspondiente a una especie distinta de suculenta, con el fin de permitir comparaciones térmicas entre las diferentes coberturas vegetales.

La adquisición de datos será gestionada por una unidad de procesamiento central basada en una placa FireBeetle ESP32-E, que cuenta con conectividad WiFi y capacidad de bajo consumo, lo cual la hace ideal para aplicaciones energéticamente eficientes en ambientes exteriores. El sistema se conectará a través de un multiplexor digital 74HC4067, lo que permitirá la lectura secuencial de múltiples sensores usando un número reducido de pines de entrada/salida del microcontrolador.

Para asegurar la operación continua del sistema sin intervención constante, se integrará un módulo de alimentación solar con una celda fotovoltaica, una batería recargable LiPo y un circuito de carga interna provisto por la propia FireBeetle, eliminando la necesidad de módulos adicionales como el TP4056. Esta solución energética garantizará la autonomía del sistema incluso en condiciones variables de radiación solar.

La información recolectada será enviada a una base de datos en la nube (como Google Sheets o un servidor web con backend personalizado), accesible desde cualquier dispositivo con conexión a internet. Además, se desarrollará una interfaz gráfica de usuario (GUI) para facilitar la visualización de datos históricos y en tiempo real, utilizando herramientas como HTML, JavaScript y librerías de visualización gráfica (por ejemplo, Chart.js o Grafana).

Para garantizar la confiabilidad del sistema, se llevará a cabo una validación experimental mediante comparaciones con instrumentos de medición de referencia, así como pruebas de desempeño en condiciones reales de operación sobre el techo verde. También se implementarán funciones de respaldo de datos y alertas automáticas en caso de detección de valores atípicos de temperatura o fallos de comunicación.

Con esta solución se busca no solo generar un conjunto de datos térmicos útiles para la caracterización de techos verdes con suculentas, sino también ofrecer una plataforma replicable, escalable y sustentable para estudios ambientales y aplicaciones urbanas de infraestructura verde inteligente.

CAPÍTULO 2

Marco Referencial

Marco teórico

2.0.1. Urbanización, cambio climático y retos térmicos actuales

El crecimiento acelerado de las zonas urbanas ha transformado radicalmente el entorno natural, alterando los ciclos térmicos, hidrológicos y ecológicos. La alta densidad de edificaciones, el uso extensivo de concreto y asfalto, y la eliminación de áreas verdes contribuyen a un aumento significativo en la temperatura superficial de las ciudades, exacerbando fenómenos como la isla de calor urbana (ICU) y elevando el consumo energético para climatización [1].

La situación se agrava con los efectos del cambio climático, que amplifican las variaciones térmicas extremas y generan mayores riesgos para la salud, infraestructura y calidad de vida urbana. En este contexto, se requieren soluciones integrales que permitan mitigar estos impactos térmicos sin aumentar la huella ecológica de las ciudades.

2.0.2. Techos verdes: definición, tipología y beneficios

Los techos verdes son sistemas constructivos que incorporan vegetación sobre cubiertas de edificios. Existen tres tipos principales:

- **Extensivos:** ligeros, de bajo mantenimiento, con vegetación rastrera.
- **Intensivos:** requieren mayor carga estructural y cuidados frecuentes, permiten arbustos o pequeños árboles.
- **Semi-intensivos:** intermedios en peso y mantenimiento.

Sus componentes típicos incluyen una membrana impermeable, una capa drenante, un sustrato especializado y plantas adaptadas. Estos sistemas contribuyen a:

- Reducir la temperatura superficial del edificio.
- Aumentar la eficiencia energética.
- Retener agua de lluvia.
- Capturar partículas contaminantes.
- Fomentar la biodiversidad urbana [5].

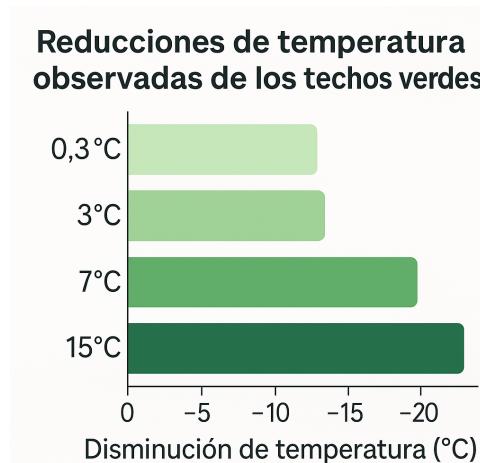


Figura 2.1: Comparativa térmica entre un techo convencional y un techo verde.

2.0.3. Isla de calor urbana y techos verdes como solución térmica

El fenómeno de la isla de calor urbana (ICU) se refiere al incremento de temperatura en zonas urbanas en comparación con sus alrededores rurales, como consecuencia de la alta densidad de edificaciones, la pavimentación extensiva y la pérdida de cobertura vegetal [9]. Este efecto genera un entorno más cálido y seco, elevando la demanda energética para climatización, afectando la salud pública y deteriorando el confort térmico en las ciudades [1].

Frente a esta problemática, los techos verdes se consideran una estrategia eficaz para mitigar la ICU. Se ha documentado que reducen entre 0.3°C y 3°C la temperatura ambiental inmediata, y hasta 15°C en el interior de los edificios donde se aplican, especialmente si se combinan con sustratos de alta retención hídrica o sistemas de riego [4] [6].

2.0.4. Importancia del monitoreo térmico localizado

La mayoría de los estudios sobre techos verdes utilizan modelos de simulación generalizados. Sin embargo, variables como el clima local, la orientación del edificio, la especie vegetal y el mantenimiento influyen de manera crítica en el comportamiento térmico real [10] [11].

Diversos autores coinciden en que la ausencia de datos localizados e in situ limita la optimización y escalabilidad de estas soluciones [12]. Por ello, se requiere un sistema confiable, de bajo costo, que recopile datos en tiempo real desde el propio techo verde para estudiar su comportamiento térmico dinámico.

2.0.5. Fundamentos de IoT

El Internet de las Cosas es la interconexión de dispositivos físicos a través de redes digitales [13]; estos dispositivos físicos contienen en su diseño interno la tecnología necesaria para recopilar y compartir información con otros dispositivos dentro o fuera de la misma red local.

En el contexto del presente proyecto, se usa el IoT como la base tecnológica para implementar el monitoreo y control remoto, la recolección de datos ambientales y la toma de decisiones automatizada para el control de un techo inteligente, como una alternativa y mejora del sistema implementado actualmente.

Arquitectura de un sistema IoT

Una aplicación de IoT está conformada por distintos elementos, como sensores, actuadores, protocolos de comunicación, servicios en la nube, etc. Además, existen capas dentro de una aplicación IoT que son representativas de este tipo de arquitecturas, como lo son los gateways.

Debido a la complejidad de un sistema de IoT, existen diversas arquitecturas que se pueden utilizar. Los dos principales enfoques que representan como una solución IoT puede ser modelada son [14]:

- Arquitectura de tres capas
- Arquitectura de cinco capas

La arquitectura de tres capas ilustra de una manera sencilla la topología de un sistema IoT. Esta simplificación hace que se pierdan algunos aspectos importantes como por ejemplo el funcionamiento interno de la red en la que los dispositivos interactúan. Incluso, puede no ser tan claro el procesamiento que se lleva a cabo en los datos recolectados en la aplicación y el valor de los mismos.

Sin embargo, la arquitectura de tres capas es la más usada en etapas iniciales del desarrollo de soluciones IoT, a continuación se muestran las capas que componen esta arquitectura.

- **Nivel de percepción:** donde se encuentran los sensores y actuadores encargados de interactuar con el entorno físico.
- **Nivel de red:** que gestiona la transmisión de datos entre dispositivos y hacia plataformas externas.
- **Nivel de aplicación:** que permite a los usuarios finales visualizar información y controlar el sistema.

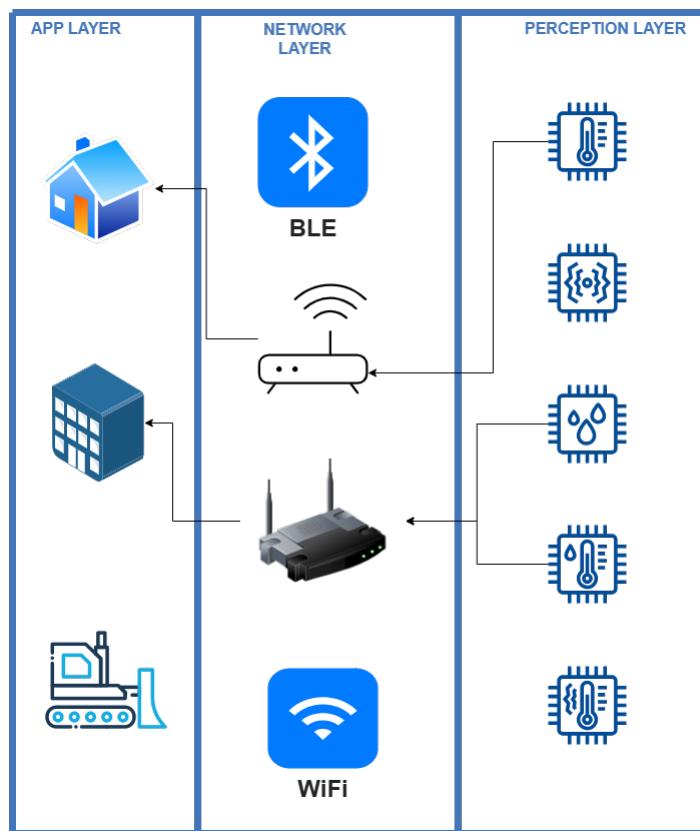


Figura 2.2: Arquitectura de tres capas para aplicaciones IoT.

Principales componentes de un sistema IoT

Un sistema basado en el Internet de las Cosas se compone de diversos elementos que permiten la adquisición de datos del entorno, su procesamiento, la transmisión hacia otros sistemas o plataformas y, en muchos casos, la ejecución de acciones de forma autónoma o remota. A continuación, se describen los principales componentes de una arquitectura IoT:

1. **Dispositivos de sensado:** Son los elementos encargados de captar información del entorno físico.
2. **Unidad de procesamiento local:** Es el núcleo de control del sistema. Generalmente está compuesto por microcontroladores o sistemas embebidos capaces de ejecutar lógica de control, gestionar las comunicaciones y tomar decisiones.
3. **Módulos de comunicación:** Permiten la transmisión y recepción de datos entre el dispositivo IoT y otros sistemas o plataformas remotas.
4. **Plataformas de almacenamiento y procesamiento remoto (cloud):** Son servicios que reciben y procesan la información enviada desde los dispositivos.

5. **Interfaces de usuario:** Son las herramientas mediante las cuales los usuarios pueden monitorear y controlar el sistema IoT. Pueden ser aplicaciones móviles, dashboards web, asistentes virtuales, entre otros.

Cada uno de estos componentes trabaja en conjunto para conformar un sistema IoT funcional, escalable y adaptable a diferentes aplicaciones.

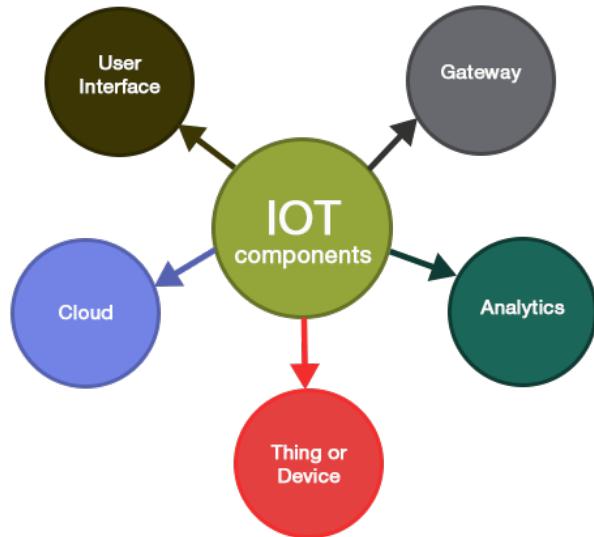


Figura 2.3: Principales elementos en un sistema IoT.

Protocolos de comunicación

La comunicación eficiente es clave en un sistema IoT, existen múltiples protocolos utilizados según los requerimientos del sistema. Algunos de los más comunes son:

- **MQTT (Message Queuing Telemetry Transport):** protocolo liviano de mensajería ideal para dispositivos con recursos limitados.
- **HTTP/HTTPS:** ampliamente utilizados para comunicación con servicios web.
- **CoAP (Constrained Application Protocol):** diseñado específicamente para sistemas embebidos con capacidades reducidas.

2.0.6. Tecnologías para la conectividad

La conectividad es un elemento fundamental en los sistemas de Internet de las Cosas (IoT), ya que permite la comunicación entre dispositivos físicos y servicios en la nube. Esta conectividad se basa en la integración de múltiples tecnologías que operan en distintas capas del modelo OSI, y su selección depende del tipo de aplicación, requisitos energéticos, alcance y topología de red.

Desde la capa física hasta la de aplicación, la arquitectura IoT incluye tecnologías inalámbricas y protocolos diseñados para entornos de bajo consumo y alto número de nodos. A nivel de red y transporte, se utilizan protocolos ligeros como MQTT y CoAP, que permiten comunicación eficiente incluso en dispositivos con recursos limitados [15]. Otro aspecto clave es la interoperabilidad, es decir, la capacidad de que dispositivos de diferentes fabricantes se comuniquen entre sí. Esto ha llevado al desarrollo de estándares comunes que aseguran la compatibilidad y la escalabilidad del sistema [16].

Las principales tecnologías usadas en sistemas IoT son:

- **Wi-Fi**
- **Bluetooth / BLE**
- **Zigbee**
- **LoRa / LoRaWAN**
- **Sigfox**
- **NB-IoT**
- **LTE-M**

2.0.7. Microcontroladores para Sistemas IoT

Los microcontroladores son el núcleo de procesamiento en los sistemas IoT, ya que gestionan la adquisición de datos, el control de dispositivos y la comunicación con plataformas en la nube [17]. La selección de un microcontrolador adecuado depende de factores clave como consumo energético, capacidad de procesamiento, conectividad y entorno de desarrollo.

A continuación, se presentan algunas de las principales plataformas utilizadas en aplicaciones IoT, que abarcan desde dispositivos de bajo consumo hasta soluciones con mayor capacidad de procesamiento:

- **ESP32:** Un microcontrolador con Wi-Fi y Bluetooth integrado, ideal para proyectos de automatización y control remoto [18].
- **Raspberry Pi:** Una SBC más potente, adecuada para aplicaciones de procesamiento intensivo como visión artificial y edge computing [19].
- **STM32:** Basado en núcleos ARM Cortex-M, cubre una amplia gama de aplicaciones, desde sistemas simples hasta soluciones en tiempo real [20].
- **Arduino:** Plataforma popular para prototipos rápidos, con un enfoque educativo y una extensa comunidad de desarrollo [21].

Consideraciones para la selección

Al seleccionar un microcontrolador, es fundamental considerar los siguientes aspectos:

- **Requisitos de procesamiento:** Necesidades de procesamiento básico o avanzado.
- **Consumo energético:** Importante para sistemas alimentados por batería.
- **Interfaces de comunicación:** Compatibilidad con tecnologías como Wi-Fi, BLE, LoRa, y otros protocolos.
- **Facilidad de desarrollo:** Disponibilidad de herramientas, documentación y comunidad de soporte.

Una elección adecuada asegura la eficiencia y escalabilidad del sistema IoT [22].

2.0.8. Sensores ambientales

Los sensores ambientales son elementos fundamentales en los sistemas de automatización basados en IoT, ya que permiten la recolección de información clave del entorno. Esta información es procesada por el sistema de control para tomar decisiones autónomas que mejoren la funcionalidad, eficiencia y comodidad del entorno controlado. En el presente proyecto, los sensores ambientales seleccionados permiten automatizar el movimiento de un techo inteligente según las condiciones del clima.

2.0.9. Sensor de temperatura

El sensor de temperatura permite detectar la cantidad de calor presente en el ambiente. Esta variable es útil para evitar condiciones de sobrecalentamiento bajo el techo, así como para definir umbrales de confort térmico. Comúnmente se utilizan sensores como el DHT22, LM35 o sensores digitales como el DS18B20, que proporcionan datos precisos y fáciles de interpretar por microcontroladores.

Termistor

Un termistor es un elemento de detección de temperatura compuesto por material semiconductor sinterizado que presenta un gran cambio en la resistencia en proporción a un cambio pequeño en la temperatura. En general, los termistores tienen coeficientes de temperatura negativos, lo que significa que la resistencia del termistor disminuye a medida que aumenta la temperatura.

Termopar

Un termopar es un tipo de sensor de temperatura formado por dos hilos o aleaciones de metales diferentes unidos por un extremo para formar una unión de medición. Produce una tensión proporcional a la diferencia de temperatura entre la unión de medición y una unión de referencia (a menudo a una temperatura conocida) cuando se somete a un gradiente de temperatura.

Sensor de temperatura resistivo (RTD)

Los sensores de temperatura resitivos, también denominados RTD, Resistance Temperature Detector en inglés, se basan en la variación de la resistencia de un conductor con la temperatura [23].

2.0.10. Plataformas IoT

Las plataformas IoT son una parte crucial en un sistema IoT, esta conecta y coordina todos los dispositivos inteligentes. No solo permiten que los sensores y actuadores se comuniquen, sino que también gestionan los datos, la seguridad y las aplicaciones que hacen que todo funcione de manera correcta.

¿Qué es una plataforma IoT?

Una plataforma IoT es un conjunto de herramientas y servicios que facilitan la conexión, gestión y análisis de dispositivos inteligentes. Actúa como un intermediario entre el hardware (sensores, actuadores) y las aplicaciones que utilizan los datos recopilados. Estas plataformas permiten que diferentes dispositivos, que a menudo utilizan distintos protocolos y tecnologías, trabajen juntos de manera armoniosa.

Características principales

Las plataformas IoT suelen ofrecer una variedad de funcionalidades clave:

- **Conectividad y compatibilidad:** Soportan múltiples protocolos de comunicación como MQTT, HTTP, CoAP, entre otros, lo que permite integrar una amplia gama de dispositivos [24].
- **Gestión de dispositivos:** Facilitan la administración de dispositivos, incluyendo su registro, configuración, monitoreo y actualización remota.

- **Procesamiento y almacenamiento de datos:** Recopilan y almacenan grandes volúmenes de datos generados por los dispositivos, y ofrecen herramientas para su análisis y visualización.
- **Seguridad:** Implementan medidas de seguridad como autenticación, autorización y cifrado para proteger los datos y dispositivos conectados.
- **Escalabilidad:** Están diseñadas para manejar desde unos pocos dispositivos hasta millones, adaptándose al crecimiento de la red IoT.

2.0.11. Tipos de plataformas IoT

Dependiendo de su enfoque y características, las plataformas IoT pueden clasificarse en:

- **Plataformas comerciales:** Ofrecidas por grandes empresas tecnológicas, suelen ser soluciones completas y de fácil implementación, aunque pueden tener costos asociados.
- **Plataformas de código abierto:** Permiten mayor personalización y control, siendo ideales para proyectos específicos o académicos.
- **Plataformas especializadas:** Diseñadas para sectores específicos como la salud, la industria o la agricultura, ofreciendo funcionalidades adaptadas a esas áreas.

2.0.12. Ejemplos de plataformas IoT

Existen diversas plataformas IoT en el mercado, cada una con sus propias características y enfoques. Algunas de las más conocidas incluyen:

- **AWS IoT Core**
- **Losant**
- **ThingsBoard**
- **Ubidots**
- **Blynk**
- **ThingSpeak**

Cada una de estas plataformas ofrece diferentes herramientas y servicios, por lo que la elección dependerá de las necesidades específicas del proyecto.

2.0.13. Microcontroladores e IoT en sistemas ambientales

La electrónica embebida y el Internet de las Cosas (IoT) han permitido el desarrollo de sistemas de monitoreo ambiental compactos y eficientes. Los microcontroladores como Arduino o ESP32 permiten adquirir, procesar y transmitir datos en tiempo real desde sensores conectados, favoreciendo el análisis continuo y remoto del entorno [7].

Los sensores digitales modernos ofrecen alta precisión con bajo consumo, y al integrarse con módulos de comunicación inalámbrica y fuentes de energía autónomas, es posible implementar estaciones de monitoreo completamente descentralizadas.

2.0.14. Especies suculentas utilizadas y clasificación por sensor

Para este estudio se emplearon siete especies distintas de suculentas. Cada sección está equipada con un sensor de temperatura, permitiendo registrar los valores térmicos del sustrato cubierto por cada planta. Además, se incluyó una sección sin vegetación como referencia de control.

Tabla 2.1: Clasificación de especies suculentas monitoreadas por sensor

Sensor	Nombre común	Nombre científico	Color identificador
1	<i>Sedum pachyphyllum</i> (Dedos de Dios)	<i>Sedum pachyphyllum</i>	Naranja
2	<i>Sedum nussbaumerianum</i> (Sedo dorado)	<i>Sedum nussbaumerianum</i>	Azul fuerte
3	<i>Sedum dendroideum</i> (Lágrima de María)	<i>Sedum dendroideum</i>	Rojo
4	<i>Sedum rubrotinctum</i> (Dedo de niño)	<i>Sedum rubrotinctum</i>	Morado
5	<i>Graptopetalum paraguayense</i> (Planta madre perla / fantasma)	<i>Graptopetalum paraguayense</i>	Amarillo
6	<i>Echeveria agavoides</i> (Echeveria oscura)	<i>Echeveria agavoides</i>	Verde limón
7	<i>Echeveria elegans</i> (Rosa de alabastro)	<i>Echeveria elegans</i>	Azul aqua
8	Sin vegetación	—	Blanco

Marco Procedimental

Definición de la metodología mecatrónica

Para llevar a cabo todas las tareas requeridas para el desarrollo de la máquina expuesta en el presente protocolo, es necesario crear un plan de trabajo coherente, dividido en áreas funcionales debido a que nuestra carrera es multidisciplinaria, esto para garantizar la obtención final del producto esperado. Se hará uso de la metodología de trabajo basada en el modelo V **Isermann**, como se muestra a en la figura 2.4:

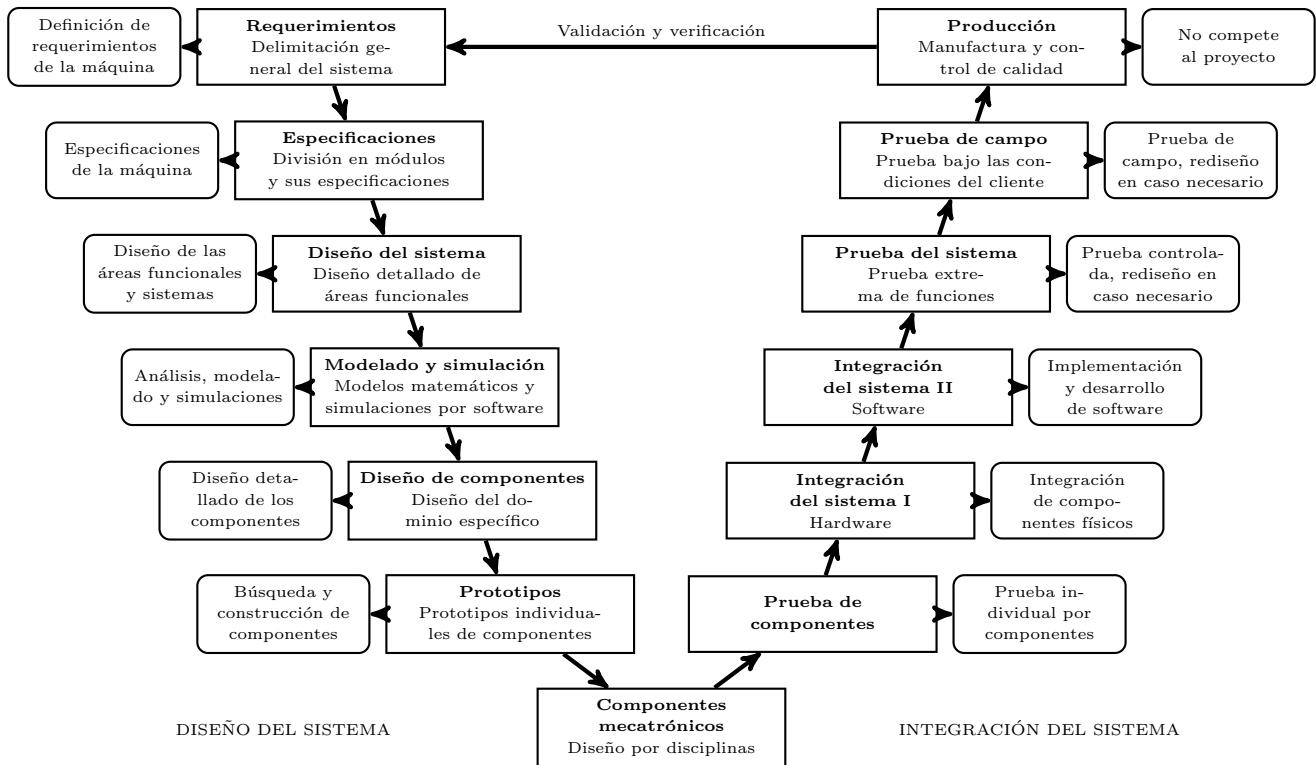


Figura 2.4: Metodología V para diseño en ingeniería mecatrónica

Requerimientos y especificaciones Definición de los requerimientos particulares obtenidos del planteamiento del problema y los cuales se especifican en la tabla 3.1 (apartado 7.2.1. del presente protocolo).

Diseño del sistema División del sistema en áreas funcionales

Diseño de componentes Selección de componentes de acuerdo a las validaciones realizadas, y cuyas características satisfagan las demandas de los mismos.

Prototipos y prueba de componentes Prototipos de los sistemas individuales una vez identificados sus componentes, así como pruebas en cada uno de ellos.

Integración del sistema I y II Integración sinérgica de sistemas y subsistemas de software y hardware en un solo prototipo.

Prueba del sistema - Prueba de campo Sometimiento del sistema a pruebas, en un primer escenario, con condiciones controladas, para posteriormente evaluar su funcionamiento en situaciones reales de operación

Producción Se refiere a la producción en cadena del prototipo llevando un control de calidad, aunque adecuado al presente protocolo, se pretende producir un único prototipo funcional por lo que este último punto no compete del todo al proyecto.

Método de objetivos ponderados

Se utiliza el método de objetivos ponderados para llevar a cabo la selección de materiales y componentes de la máquina.

Este método es ampliamente utilizado para elegir una opción entre diversas alternativas basado en la importancia de los objetivos buscados que bien pueden ser cualitativos, cuantitativos o ambos. El procedimiento a seguir es el siguiente:

1. Determinar los objetivos que debe cubrir la selección.
2. Enfrentar uno a uno los objetivos para determinar la importancia entre ellos.
3. Determinar un peso para cada objetivo de acuerdo a su importancia.
4. Generar alternativas.
5. Enfrentar las alternativas con los objetivos asignándole una calificación en cuanto al desempeño ante dicho objetivo. La calificación debe comprenderse entre un mismo rango para todos los casos (ejemplo, 0-10, 0-100, etc.).
6. Obtener la calificación global de cada alternativa aplicando la ecuación 2.1.
7. Hacer una recomendación basada en la alternativa que haya obtenido un mayor puntaje en la calificación global.

$$C = \sum W_i \cdot P_i \quad (2.1)$$

Donde:

C - calificación global.

W_i - peso del objetivo i.

P_i - calificación parcial i de la alternativa para el objetivo i.

CAPÍTULO 3

Diseño del Sistema

Necesidades

Requerimientos del sistema

Adecuando lo propuesto en el modelo en “V” acerca de los requerimientos del sistema descritos por el cliente, para este proyecto se considerarán las necesidades descritas en los antecedentes previamente mencionados y se enlistan los requerimientos en la tabla 3.1.

Tabla 3.1: Requerimientos del sistema de monitoreo térmico en techo verde

R1 Variables a monitorear.
R1.1 Temperatura del sustrato en zonas con siete especies diferentes de suculentas del techo verde.
R1.2 Temperatura de control en zona sin vegetación.
R2 Adquisición de datos.
R2.1 Lectura de sensores DS18B20 cada 30 minutos.
R2.2 Activación secuencial de sensores.
R3 Almacenamiento y envío de información.
R3.1 Envío de datos a una base de datos mediante conexión Wi-Fi.
R3.2 Visualización de datos a través de página web accesible con URL propia.
R3.3 En caso de pérdida de red Wi-Fi, almacenamiento local temporal en memoria de la ESP32.
R4 Energía del sistema.
R4.1 Alimentación primaria mediante conexión a la red eléctrica.
R4.2 Alternativa de alimentación por panel solar y batería recargable LiPo.
R5 Estructura del sistema.
R5.1 Soporte rígido para montaje de sensores y electrónica sobre el techo verde.
R5.2 Protección de componentes frente a condiciones ambientales.
R6 Funcionalidad operativa.
R6.1 Sistema capaz de operar de forma continua durante 24 horas.
R6.2 Máximo margen de error aceptado en lecturas: ± 0.5 °C.
R7 Fines académicos.
R7.1 El sistema debe generar datos válidos para análisis comparativo entre vegetación y superficie descubierta.
R7.2 Su diseño debe permitir modificaciones para estudios futuros del CIEMAD.

Arquitectura funcional

Ahora se pasa a describir las funciones y subfunciones que debe tener el sistema para satisfacer los requerimientos del monitoreo térmico del techo verde:

F1. Gestión de energía.

F1.1. Almacenar energía en batería recargable.

F1.2. Acondicionar energía para operación de bajo consumo.

F1.3. Seleccionar fuente energética (solar o directa).

F2. Gestión de adquisición de datos.

F2.1. Activar el sistema en intervalos definidos.

F2.2. Ejecutar la máquina de estados para lectura secuencial de sensores.

F2.3. Capturar temperatura desde sensores DS18B20.

F2.4. Almacenar datos temporalmente en memoria local.

F3. Procesamiento y almacenamiento.

F3.1. Identificar zona del sensor correspondiente (especie de suculenta o control sin vegetación).

F3.2. Registrar fecha, hora y temperatura por sensor.

F4. Comunicación y envío de información.

F4.1. Verificar disponibilidad de red WiFi.

F4.2. Enviar datos a servidor web para visualización.

F4.3. Mantener los datos en almacenamiento local en caso de pérdida de conexión.

F5. Interfaz y visualización.

F5.1. Mostrar valores actuales mediante servidor web local.

F5.2. Facilitar acceso remoto a datos históricos.

F6. Mantenimiento y control.

F6.1. Permitir reinicio manual del sistema.

F6.2. Actualizar el sistema mediante conexión USB o inalámbrica.

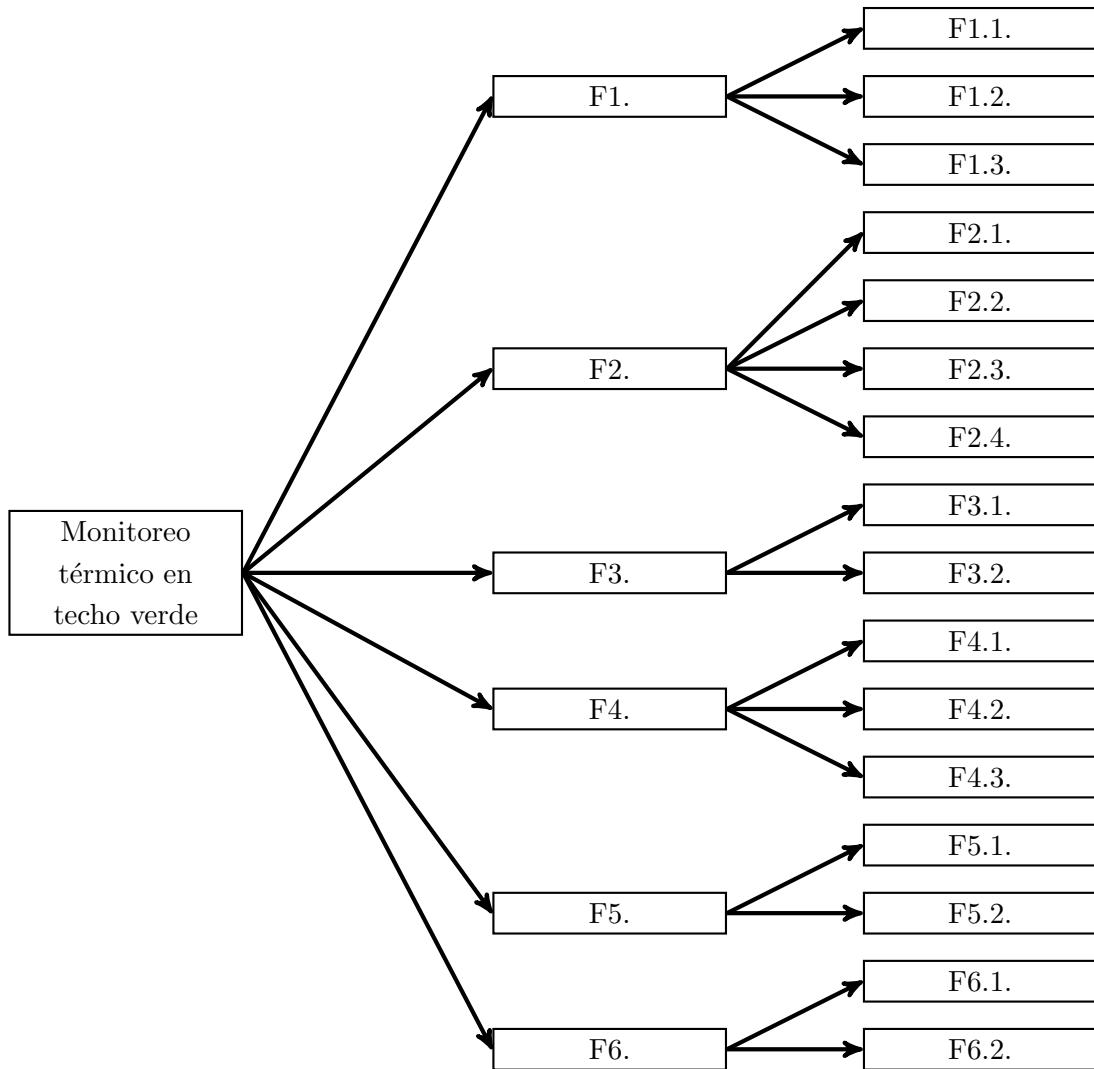


Figura 3.1: Diagrama FBS del sistema de monitoreo térmico en techo verde

Explicación de los sistemas a desarrollar A continuación se describen los sistemas propuestos para el monitoreo térmico en techo verde, con base en el diagrama IDEF0 desarrollado:

A1. Sistema de Adquisición de Datos

Este sistema será el responsable de recopilar información de temperatura del suelo mediante sensores DS18B20. Para ello, se implementará una máquina de estados y una lógica de lectura secuencial que gestione adecuadamente la toma de muestras desde múltiples sensores.

A2. Sistema de Almacenamiento

Este sistema garantizará que los datos capturados puedan ser almacenados de forma temporal en la memoria interna del microcontrolador, especialmente útil en casos donde no se cuente con conexión de red. Su función es mantener un respaldo local de las mediciones térmicas.

A3. Sistema de Transmisión

Este sistema permitirá el envío de los datos térmicos capturados y/o almacenados hacia una hoja de cálculo remota o sitio web mediante conexión WiFi. Para ello, se hará uso de un enlace inalámbrico y un protocolo de transmisión a la red disponible.

A4. Sistema de Visualización

Este sistema permitirá al usuario visualizar de forma remota los datos procesados. La interfaz incluirá tanto la consulta a través de una hoja de cálculo (Excel/Google Sheets) como una página web alojada en el servidor, permitiendo acceso desde distintos dispositivos.

Alimentación del sistema

Todos los sistemas serán alimentados mediante una fuente solar y/o red eléctrica.

Arquitectura física

Resumiendo la explicación anterior en los módulos que conforman cada sistema:

A1. Sistema de Adquisición de Datos

- M1. Sensado térmico (sensores DS18B20).
- M2. Máquina de estados.
- M3. Lectura secuencial.

A2. Sistema de Almacenamiento

- M6. Memoria interna.

A3. Sistema de Transmisión

- M4. Enlace WiFi.
- M5. Protocolo de transmisión.

A4. Sistema de Visualización

- M7. Hoja de cálculo (Excel).
- M8. Página web.

Alimentación

Fuente solar o red eléctrica para todo el sistema.

En la figura 3.2 se muestra el diagrama de bloques representativo de la arquitectura física del sistema.

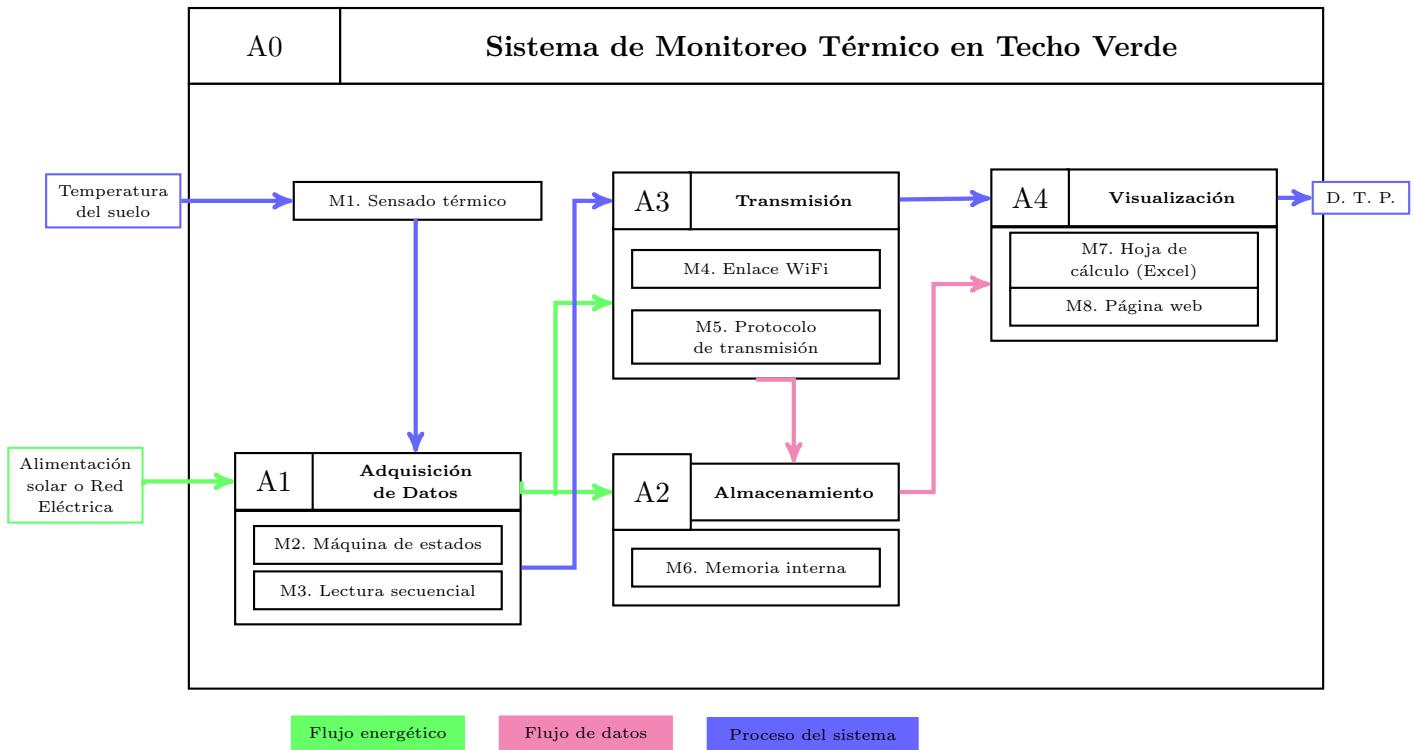


Figura 3.2: Arquitectura física del sistema

Propuesta de diseño

El diseño conceptual del sistema de monitoreo térmico para un techo verde parte de la necesidad de generar datos confiables sobre la temperatura del sustrato en diferentes zonas con y sin cobertura vegetal, con el fin de evaluar su comportamiento térmico y su potencial impacto en la eficiencia ambiental de estas infraestructuras urbanas.

Para lograr este propósito, se plantea una arquitectura basada en un microcontrolador FireBeetle ESP32-E, elegido por su bajo consumo energético, conectividad Wi-Fi integrada y compatibilidad con fuentes de alimentación solar. Este componente central coordina la lectura de ocho sensores de temperatura DS18B20, los cuales están distribuidos estratégicamente: siete de ellos ubicados en distintas secciones del techo verde, cada una con una especie suculenta diferente, y uno adicional en una zona sin vegetación, lo cual permite establecer una línea base para la comparación térmica.

Con el fin de optimizar el consumo energético y reducir la cantidad de pines necesarios, se incorpora un multiplexor 74HC4067, que permite realizar lecturas secuenciales de los sensores utilizando una lógica de máquina de estados. Esta arquitectura asegura un funcionamiento eficiente durante las 24 horas del día, realizando mediciones cada 30 minutos.

El sistema está diseñado con una doble opción de alimentación: puede operar tanto con una batería LiPo recargable mediante energía solar como mediante una conexión directa a la red eléctrica. Además, se contempla una estrategia de almacenamiento local de datos en la memoria del ESP32 en caso de pérdida de conexión Wi-Fi, garantizando la integridad de la información recopilada.

La estructura mecánica que soporta el sistema está diseñada para ser robusta, estable y resistente a la intemperie, permitiendo proteger los componentes electrónicos sin interferir con el proceso de medición. El diseño modular facilita el mantenimiento, la calibración de sensores y la reconfiguración del sistema en caso de futuras ampliaciones o ajustes experimentales.

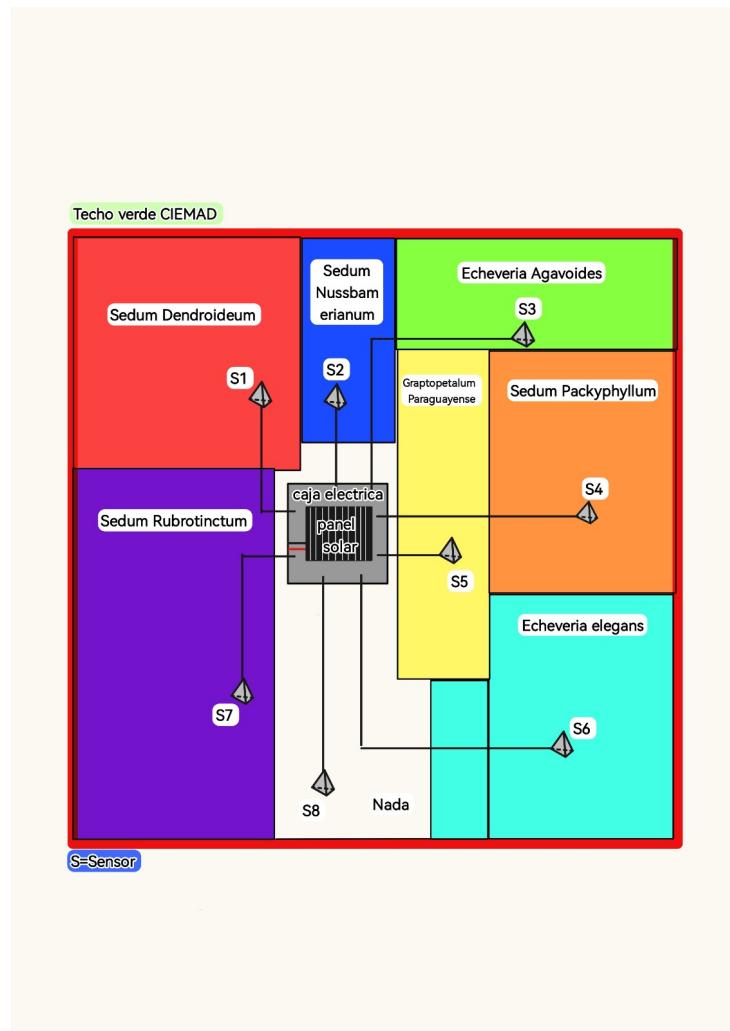


Figura 3.3: Diseño Conceptual

CAPÍTULO 4

Diseño en el Dominio Específico

4.1. Componentes mecánicos

4.1.1. Análisis de tensiones en el chasis de protección

Para validar el desempeño mecánico del chasis de protección del sistema embebido se realizó un análisis estático en *SolidWorks Simulation*, considerando el material ABS y una carga equivalente al peso de los componentes internos del sistema.

Antes de presentar el análisis, se muestra la siguiente tabla, la cual muestar las propiedades mecánicas típicas del ABS

Tabla 4.1: Propiedades mecánicas típicas del ABS (Acrilonitrilo Butadieno Estireno).

Propiedad	Símbolo / Unidad	Valor típico
Módulo de elasticidad (Young)	E [GPa]	2,0 – 2,5
Límite de fluencia	σ_y [MPa]	35 – 45
Resistencia a la tracción	σ_t [MPa]	40 – 50
Coeficiente de Poisson	ν [-]	0,35 – 0,40
Densidad	ρ [kg/m ³]	1040 – 1080
Alargamiento a la rotura	ε_f [%]	10 – 50

La carga total se estimó en $m = 0,250$ kg, correspondiente a $F = mg$, donde g es la aceleración de la gravedad. Así, la fuerza aplicada se calculó como:

$$F = m \cdot g = 0,250 \text{ kg} \cdot 9,81 \text{ m/s}^2 \approx 2,45 \text{ N} \quad (4.1)$$

Dicha carga se distribuyó uniformemente en la base del chasis, mientras que las esquinas superiores se restringieron para simular los puntos de fijación. El criterio de evaluación fue la tensión equivalente de Von Mises, que se expresa como:

$$\sigma_{\text{vm}} = \sqrt{\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2}} \quad (4.2)$$

donde $\sigma_1, \sigma_2, \sigma_3$ son las tensiones principales en el punto analizado.

En el caso de esfuerzos planos ($\sigma_z = 0$), la expresión se reduce a:

$$\sigma_{\text{vm}} = \sqrt{\sigma_x^2 - \sigma_x \sigma_y + \sigma_y^2 + 3\tau_{xy}^2} \quad (4.3)$$

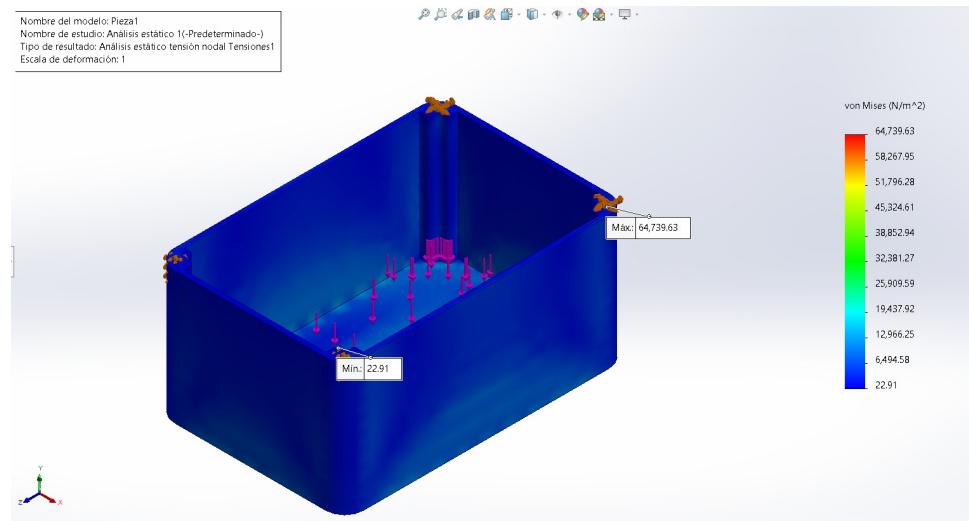


Figura 4.1: Análisis estático: Tensión Von Mises

El análisis arrojó un valor máximo de tensión equivalente de:

$$\sigma_{vm,max} \approx 0,0647 \text{ MPa} \quad (4.4)$$

Comparando este resultado con el límite de fluencia del ABS, tomado de manera conservadora como $\sigma_y = 35 \text{ MPa}$, se obtiene el factor de seguridad:

$$FS = \frac{\sigma_y}{\sigma_{vm,max}} = \frac{35 \text{ MPa}}{0,0647 \text{ MPa}} \approx 540 \quad (4.5)$$

El factor de seguridad obtenido es significativamente mayor al valor recomendado en diseño estático para materiales plásticos ($FS \approx 2,0-3,0$). Esto indica que el chasis fabricado en ABS es capaz de soportar la carga estática asociada al peso de los componentes electrónicos con un amplio margen de seguridad.

Además, las máximas concentraciones de esfuerzo se localizaron en las esquinas fijadas, lo cual es coherente con las condiciones de frontera del modelo. No obstante, dichas tensiones están muy por debajo del límite de fluencia del material, garantizando que la caja cumple satisfactoriamente con su función estructural.

4.1.2. Análisis de desplazamiento estático del chasis

Con el fin de verificar la rigidez del chasis de ABS frente a la carga funcional del sistema embebido, se realizó un análisis estático lineal en *SolidWorks Simulation* y se evaluó el *desplazamiento resultante* (URES). Se emplearon las mismas condiciones del estudio de tensiones:

- **Material:** ABS (inyección), propiedades típicas como en la Tabla 4.1.
- **Carga:** masa total $m = 0,250 \text{ kg}$ aplicada como presión distribuida en la base.

$$F = mg = 0,250 \times 9,81 \approx 2,45 \text{ N} \quad (4.6)$$

- **Restricciones:** fijaciones en las esquinas superiores (puntos de atornillado).
- **Escala de deformación:** 1 (sin amplificación).

Formulación (estático lineal)

Bajo el supuesto de pequeñas deformaciones, el problema se resuelve con el sistema de ecuaciones del MEF:

$$\mathbf{K} \mathbf{u} = \mathbf{f}, \quad (4.7)$$

donde \mathbf{K} es la matriz global de rigidez, \mathbf{u} el vector de desplazamientos nodales y \mathbf{f} el vector de cargas equivalentes. El campo de deformaciones y tensiones queda dado por

$$\boldsymbol{\varepsilon} = \mathbf{B} \mathbf{u}, \quad \boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}, \quad (4.8)$$

siendo \mathbf{B} la matriz cinemática y \mathbf{D} la matriz constitutiva (ley de Hooke para material isótropo).

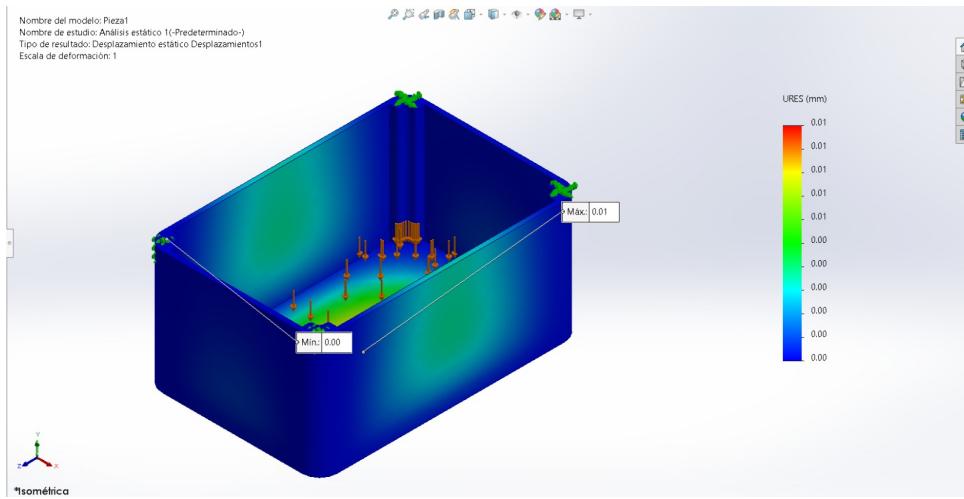


Figura 4.2: Análisis estático: Desplazamiento estático

Resultado de desplazamientos

La figura de *desplazamientos estáticos* (URES) reporta:

$$U_{\max} \approx 0,01 \text{ mm}, \quad U_{\min} \approx 0,00 \text{ mm}. \quad (4.9)$$

El máximo se localiza próximo al borde libre opuesto a una de las fijaciones, coherente con la distribución de rigidez y las condiciones de frontera. La deformación visual corresponde a la real (escala 1:1).

Comprobación de servicio (deflexión)

Para elementos de cascarón/caja es común verificar una *limitación de flecha* tipo

$$\delta_{\text{adm}} = \frac{L}{\eta}, \quad (4.10)$$

donde L es una longitud característica (p. ej., el claro mayor de la base) y η es un coeficiente de servicio; un valor conservador ampliamente utilizado es $\eta = 200$. Aun para L en el rango [100, 200] mm,

$$\delta_{\text{adm}} = \frac{L}{200} \in [0,50, 1,00] \text{ mm}, \quad (4.11)$$

mientras que el resultado numérico es

$$U_{\max} = 0,01 \text{ mm} \ll \delta_{\text{adm}}. \quad (4.12)$$

Por tanto, la deflexión es entre 50 y 100 veces menor que la admisible, cumpliendo holgadamente el criterio de rigidez para servicio.

Con la carga operativa de 2,45 N, el chasis de ABS presenta un desplazamiento máximo de únicamente 0,01 mm, valor muy inferior a límites de servicio típicos y coherente con las bajas tensiones obtenidas en el análisis de Von Mises. Se concluye que la **rigidez estructural** de la caja es adecuada para su función prevista, garantizando estabilidad geométrica y correcta protección del sistema embebido bajo carga estática.

4.1.3. Análisis de deformación unitaria estática del chasis

Con el mismo montaje del estudio estático (material ABS, fijaciones en las cuatro esquinas superiores y carga equivalente al peso del conjunto electrónico distribuida en la base), se evaluó la *deformación unitaria* (*strain*) mediante *SolidWorks Simulation*. La masa total considerada fue $m = 0,250$ kg, por lo que la fuerza aplicada resulta:

$$F = mg = 0,250 \times 9,81 \approx 2,45 \text{ N.} \quad (4.13)$$

Formulación teórica

Para pequeñas deformaciones, el campo de desplazamientos \mathbf{u} y la deformación unitaria (tensor de Cauchy–Green lineal) se relacionan como:

$$\boldsymbol{\varepsilon} = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T). \quad (4.14)$$

En material elástico lineal, isótropo, la ley de Hooke se escribe:

$$\boldsymbol{\sigma} = \lambda \text{tr}(\boldsymbol{\varepsilon}) \mathbf{I} + 2G \boldsymbol{\varepsilon}, \quad (4.15)$$

con $G = \frac{E}{2(1 + \nu)}$ y $\lambda = \frac{E\nu}{(1 + \nu)(1 - 2\nu)}$, donde E es el módulo de Young y ν el coeficiente de Poisson.

El posprocesado reporta la *deformación unitaria equivalente* (tipo von Mises), definida a partir de las deformaciones principales ($\varepsilon_1, \varepsilon_2, \varepsilon_3$) como:

$$\varepsilon_{\text{eq}} = \sqrt{\frac{2}{3} [(\varepsilon_1 - \varepsilon_2)^2 + (\varepsilon_2 - \varepsilon_3)^2 + (\varepsilon_3 - \varepsilon_1)^2]}. \quad (4.16)$$

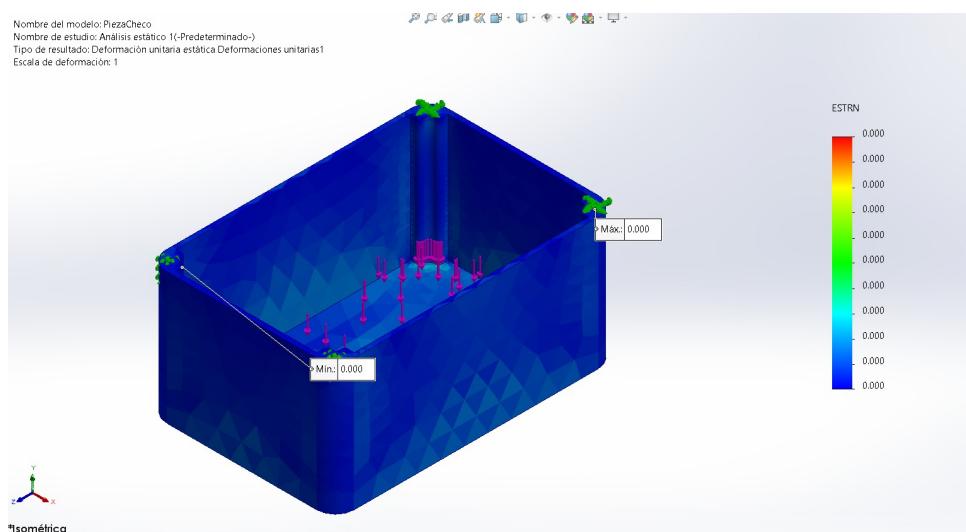


Figura 4.3: Análisis estático: Deformación unitaria

Resultados y estimaciones

De acuerdo con el estudio de tensiones presentado en la Sección correspondiente, la tensión equivalente máxima fue

$$\sigma_{\text{vm, max}} \approx 0,0647 \text{ MPa.}$$

Tomando un valor típico para ABS de $E = 2,2 \text{ GPa}$, una cota superior razonable de la deformación unitaria elástica máxima se obtiene por relación uniaxial:

$$\varepsilon_{\text{máx}} \approx \frac{\sigma_{\text{vm, max}}}{E} = \frac{0,0647}{2200} \approx 2,94 \times 10^{-5} \text{ } (\approx 30 \text{ } \mu\varepsilon \text{ o } 0,0029 \text{ \%}). \quad (4.17)$$

Este orden de magnitud explica que, en la figura de contornos, la barra de escala (con redondeo a tres decimales) muestre valores cercanos a 0,000.

De forma consistente, si se considera la relación *media* deformación–flecha,

$$\bar{\varepsilon} \approx \frac{\Delta L}{L}, \quad (4.18)$$

con el desplazamiento máximo medido $U_{\text{máx}} \approx 0,01 \text{ mm}$ y una longitud característica $L \geq 100 \text{ mm}$ del panel, se obtiene $\bar{\varepsilon} \leq 1,0 \times 10^{-4}$, del mismo orden (o mayor que) la estimación elástica anterior, por lo que los resultados son coherentes.

Verificación frente a fluencia (versión en deformaciones)

El límite de fluencia conservador adoptado para ABS es $\sigma_y = 35 \text{ MPa}$, lo que implica una deformación de fluencia elástica aproximada:

$$\varepsilon_y \approx \frac{\sigma_y}{E} = \frac{35}{2200} \approx 1,59 \times 10^{-2} \text{ (1,59 \%)} \quad (4.19)$$

El *factor de seguridad en deformación* resulta:

$$FS_{\varepsilon} = \frac{\varepsilon_y}{\varepsilon_{\text{máx}}} \approx \frac{1,59 \times 10^{-2}}{2,94 \times 10^{-5}} \approx 540, \quad (4.20)$$

en concordancia con el factor de seguridad calculado por tensiones.

La deformación unitaria equivalente en el chasis de ABS bajo la carga funcional de 2,45 N es del orden de 10^{-5} (decenas de microdeformación), muy inferior a la asociada a la fluencia del material y a cualquier criterio de servicio usual. En consecuencia, el **comportamiento deformacional** de la caja es adecuado: las variaciones geométricas son despreciables para el uso previsto y la rigidez estructural es suficiente para proteger el sistema embebido.

4.1.4. Cumplimiento de requisitos ambientales: agua y temperatura

Además del análisis estructural (tensiones, desplazamientos y deformaciones unitarias), se verificó el cumplimiento de requisitos ambientales para el chasis de ABS, considerando **resistencia al agua (salpicaduras)** y **resistencia térmica** dentro del rango operativo del sistema embebido.

Hermeticidad frente a agua (salpicaduras, IPX4)

Se adoptaron los siguientes *criterios de diseño* y verificación:

- **Geometría de tapa:** solape/laberinto perimetral que protege el sello del chorro directo y del escurrimiento.
- **Rigidez:** desplazamiento máximo bajo carga funcional $U_{\max} = 0,01$ mm (Sección de desplazamientos), muy inferior a la compresión de la junta (0,4–0,6 mm), por lo que *no* se pierde contacto de sellado.

Como referencia, la presión hidrostática de una lámina de agua de altura h es

$$p_h = \rho gh. \quad (4.21)$$

Resistencia a temperatura (operación)

Para ABS de uso general se consideran propiedades típicas (Tabla 4.1): módulo $E \simeq 2,0\text{--}2,5$ GPa, coeficiente de Poisson $\nu \simeq 0,35\text{--}0,40$, transición vítrea $T_g \approx 105^\circ\text{C}$. Se especifica un **rango operativo** del equipo de

$$T_{\text{amb}} \in [-10^\circ\text{C}, 60^\circ\text{C}],$$

compatible con el ABS y el hardware electrónico.

Se comprueba que las **dilataciones térmicas** no comprometen el sellado:

$$\Delta L = \alpha L \Delta T, \quad (4.22)$$

donde α es el coeficiente de dilatación lineal del ABS ($\alpha \approx 80\text{--}100 \times 10^{-6} \text{ K}^{-1}$), L una longitud característica del perímetro de tapa y ΔT la variación térmica. Para $L = 150$ mm y $\Delta T = 40$ K se obtiene

$$\Delta L \leq 0,60 \text{ mm} \quad (\alpha = 100 \times 10^{-6} \text{ K}^{-1}),$$

valor del mismo orden que la compresión máxima de la junta ($\approx 0,6$ mm a 30 %). Dado que tapa y base dilatan de forma *congruente* y el sello trabaja en compresión, se mantiene la estanqueidad en el rango térmico especificado. Además, las tensiones por carga funcional son muy bajas ($\sigma_{\text{vm,max}} \approx 0,0647$ MPa $\ll \sigma_y$), por lo que no existe riesgo de fluencia por efecto combinado *mecánico+térmico* en operación.

Resumen de cumplimiento

Requisito	Criterio / Evidencia	Veredicto
Resistencia a agua (salpicaduras)	Junta EPDM/silicona 2 mm, compresión 20–30 %; tapa con solape; $U_{\max} = 0,01$ mm \ll compresión del sello; $p_h(0,5 \text{ m}) \approx 4,9 \text{ kPa}$	Cumple (IPX4 por diseño)
Resistencia térmica	Rango de operación -10 a 60°C ; $\Delta L = \alpha L \Delta T \leq 0,60$ mm para $L = 150$ mm y $\Delta T = 40$ K; compresión del sello 0,4–0,6 mm; $\sigma_{\text{vm,max}} \ll \sigma_y$	Cumple

Bajo las condiciones de diseño establecidas (sello comprimido 20–30 %, geometría con solape y rango térmico de -10 a 60°C), el chasis de ABS **cumple** con los requisitos de **resistencia al agua (salpicaduras)** y **resistencia a temperatura**. La baja deformación estructural ($U_{\max} = 0,01$ mm) garantiza la integridad del sello en servicio, y las tensiones se mantienen muy por debajo del límite de fluencia del material, asegurando desempeño confiable en el entorno de operación previsto.

4.2. Sistema eléctrico - electrónico

4.2.1. Selección de componentes mediante matriz de decisión

Se aplicó un esquema de evaluación multicriterio (MCDA) con ponderaciones w_i ($\sum w_i = 1$) y calificaciones $r_{i,j} \in [1, 5]$ para cada alternativa j en el criterio i . El puntaje total se calcula como:

Las ponderaciones reflejan los objetivos del proyecto (bajo consumo, disponibilidad, costo, facilidad de integración y adecuación funcional).

Microcontrolador

Candidatos: ESP32, ESP8266, ATmega328P, STM32F103.

graphicx booktabs makecell array caption

Tabla 4.2: Matriz de decisión: microcontrolador (pesos entre paréntesis).

Alternativa	Conect.	I/O	Costo	Potencia		S
	(0.20)	DeepSleep (0.15)	(0.10)	Facilidad (0.15)	Seguridad (0.10)	
ESP32	5	4	5	4	5	5
ESP8266	4	3	3	5	4	3
ATmega328P	1	4	3	5	5	2
STM32F103	2	4	4	4	3	4

Selección: ESP32 por mejor conectividad (WiFi+BLE), potencia, seguridad (TLS/HTTPS) y ecosistema.

Sensores de temperatura puntuales

Candidatos: DS18B20, NTC 10k, LM35.

Tabla 4.3: Matriz de decisión: sensor de temperatura puntual.

Alternativa	Calibr.		Costo	Interfaz		S
	Precisión (0.25)	Cableado (0.20)	(0.15)	Consumo (0.10)	(0.15)	
DS18B20	4	5	4	4	5	4
NTC 10k	3	3	2	5	5	3
LM35	4	3	3	3	4	3

Selección: DS18B20 (bus 1-Wire, cables largos y lectura digital estable).

Pantalla de usuario

Candidatos: OLED SSD1306, LCD 1602 I²C, TFT ST7735.

Tabla 4.4: Matriz de decisión: interfaz de visualización.

Alternativa					Dispon. (0.10)	<i>S</i>		
	Legib.		Costo	Facilidad (0.10)				
	Consumo (0.25)	Cap. info. (0.20)	(0.15)					
OLED SSD1306	5	4	3	4	5	5 4.25		
LCD 1602 I ² C	3	3	2	5	4	5 3.40		
TFT ST7735	2	3	5	3	3	4 3.25		

Selección: OLED SSD1306 (bajo consumo y rápida integración).

Actuador de potencia (conmutación de carga DC)

Candidatos: relé mecánico, MOSFET canal N, SSR DC. La carga a conmutar es DC de baja tensión, con necesidad de *contacto seco* y simplicidad de servicio.

Tabla 4.5: Matriz de decisión: actuador de potencia.

Alternativa	Pérd.		Costo	Durab. (0.05)	Caída V (0.05)	Adecuación (0.20)
	(0.20)	Aislamiento (0.15)	Control 3.3V (0.10)			
	(0.25)	(0.25)	(0.10)			
Relé mecánico	2	5	4	4	3	5 5
MOSFET N	5	2	4	4	5	3 4
SSR DC	4	5	2	5	5	2 3

Selección: relé mecánico por requerir contacto seco, excelente aislamiento y simple mantenimiento.

Cargador de batería Li-ion

Candidatos: TP4056 (módulo con protección), MCP73831, CN3065 (con MPPT básico).

Tabla 4.6: Matriz de decisión: cargador Li-ion.

Alternativa	I _{carga}	Dispon.	Costo	S Efic./Térm. (0.15)
	Protección (0.20)	(0.20)	(0.20)	
TP4056	5	5	5	3 4.70
MCP73831	3	4	4	3 3.60
CN3065	4	4	3	4 3.60

Selección: TP4056 por protección integrada, alta disponibilidad y bajo costo.

Regulación a 3.3 V (versión integrada)

Candidatos: LDO baja I_q (MCP1700), LDO AMS1117 (devkit), Buck MP1584.

Tabla 4.7: Matriz de decisión: regulador a 3.3 V.

Alternativa	Efic.	I_q	Costo	Dispon.	Ruido	Facilidad (0.05)
	(0.30)	(0.25)	(0.15)	(0.15)	(0.10)	
LDO MCP1700	3	5	4	4	5	5
LDO AMS1117	2	1	5	5	4	5
Buck MP1584	5	2	3	4	3	3

Selección recomendada (versión de baja potencia): LDO MCP1700 por bajísimo I_q en *deep sleep*. *Nota:* En prototipo con placa ESP32 se usa el LDO integrado (AMS1117) por disponibilidad; en versión final se sugiere *Buck + LDO* si la entrada supera 5 V o hay disipación apreciable.

4.2.2. Diagrama Electrónico y Conexiones

El diseño del sistema se basa en un microcontrolador ESP32, al que se conectan distintos sensores, una pantalla OLED, un relevador y los módulos de energía (panel solar y batería).

- Los sensores **DS18B20** se conectan mediante el bus 1-Wire en el pin GPIO 25, permitiendo medir la temperatura de hasta ocho puntos diferentes.
- Los sensores ambientales se comunican a través de la interfaz I²C compartida (SDA y SCL).
- La pantalla OLED **SSD1306** también utiliza la misma interfaz I²C.
- El relevador se controla mediante el pin GPIO 12, permitiendo habilitar o deshabilitar la carga a partir de los voltajes medidos.

- El voltaje del panel solar y de la batería se mide a través de los pines analógicos GPIO 34 y GPIO 35, respectivamente, con divisores de tensión para adaptar los niveles al rango máximo del ADC del ESP32.
- El encoder rotativo se conecta a los pines GPIO 16 (CLK), GPIO 17 (DT) y GPIO 4 (SW), permitiendo la selección del intervalo de tiempo para los ciclos de monitoreo.

4.3. Sistema eléctrico-electrónico y cálculos de dimensionamiento

Además de la integración de sensores, pantalla y comunicaciones, el desempeño del sistema depende de un correcto *presupuesto de potencia*, del *dimensionamiento de la batería y del panel*, del tratamiento de *mediciones analógico-digitales* y del *accionamiento del relevador*. En esta sección se consolidan los cálculos clave.

4.3.1. Presupuesto de corriente y potencia

Para cada bloque se define corriente y potencia típicas¹:

Bloque	Corriente	Potencia a 3.3 V
ESP32 activo (CPU+WiFi, promedio)	$I_{\text{ESP,act}} \approx 80\text{--}150 \text{ mA}$	$P \approx 0,26\text{--}0,50 \text{ W}$
ESP32 <i>deep sleep</i>	$I_{\text{DS}} \approx 5\text{--}20 \text{ }\mu\text{A}$	$\approx 0 \text{ W}$
OLED SSD1306 (0.96")	$I_{\text{OLED}} \approx 10\text{--}20 \text{ mA}$	$P \approx 33\text{--}66 \text{ mW}$
AHT10 + BMP280 + BH1750	$I_{\text{I2C}} \approx 1,5\text{--}2,0 \text{ mA}$	$P \approx 5\text{--}7 \text{ mW}$
DS18B20 (conversión)	$I_{\text{1W}} \approx 1,5 \text{ mA /sensor}$	$P \approx 5 \text{ mW/sensor}$
Relevador (bobina DC)	$I_{\text{coil}} \approx 30\text{--}80 \text{ mA}$	$P = V_{\text{coil}} I_{\text{coil}}$

La **corriente media** del sistema depende del ciclo de trabajo de cada bloque:

$$I_{\text{avg}} = \alpha_{\text{act}} I_{\text{ESP,act}} + \alpha_{\text{OLED}} I_{\text{OLED}} + \alpha_{\text{coil}} I_{\text{coil}} + I_{\text{I2C}} + \alpha_{\text{1W}} N_{\text{1W}} I_{\text{1W}} + (1 - \alpha_{\text{act}}) I_{\text{DS}}, \quad (4.23)$$

donde α es la fracción de tiempo activa de cada bloque y N_{1W} el número de sensores DS18B20. En tu sistema diurno la OLED está encendida el 20% del intervalo de cada parte ($\alpha_{\text{OLED}}=0,2$), y por la noche $I \approx I_{\text{DS}}$ tras el envío.

4.3.2. Autonomía con batería

Dados la capacidad C de la batería y el bus de 3.3 V, la autonomía ideal (sin convertidor) es:

$$t_{\text{h}} \approx \frac{C_{\text{mAh}}}{I_{\text{avg,mA}}}. \quad (4.24)$$

¹Valores representativos de hojas de datos usuales: deben sustituirse por los del modelo específico utilizado.

Si existe convertidor (buck/boost) con eficiencia η , la corriente equivalente en el lado de batería de tensión V_{bat} es:

$$I_{\text{bat}} \approx \frac{V_{3,3}}{V_{\text{bat}}} \frac{I_{\text{avg}}}{\eta}, \quad t_h \approx \frac{C}{I_{\text{bat}}}. \quad (4.25)$$

Considerando una **profundidad de descarga** admisible DoD:

$$t_{\text{h,DoD}} = t_h \cdot \text{DoD}. \quad (4.26)$$

Ejemplo ilustrativo. Supón $C = 2200 \text{ mAh}$, $V_{\text{bat}} = 3,7 \text{ V}$, $\eta = 0,9$, $I_{\text{avg}} = 30 \text{ mA}$ (resultado de (4.23)) $\Rightarrow I_{\text{bat}} \approx (3,3/3,7)(30/0,9) \approx 30 \text{ mA}$ y $t \approx 2200/30 \approx 73 \text{ h} (\sim 3 \text{ d})$. Con DoD = 0,8, $t \approx 58 \text{ h}$.

4.3.3. Dimensionamiento del panel solar

La energía diaria requerida es

$$E_{\text{día}} = V_{3,3} I_{\text{avg}} 24 \text{ [Wh]}, \quad (4.27)$$

y la energía disponible de un panel de potencia pico P_{Wp} es

$$E_{\text{PV}} = P_{\text{Wp}} H_{\text{sol}} \eta_{\text{sist}}, \quad (4.28)$$

donde H_{sol} son las *horas sol pico* del sitio y η_{sist} agrupa pérdidas (orientación, temperatura, regulador, cableado). Se requiere:

$$E_{\text{PV}} \geq E_{\text{día}} + E_{\text{recarga}}, \quad (4.29)$$

con E_{recarga} la energía para reponer la descarga nocturna.

Ejemplo. Si $I_{\text{avg}} = 30 \text{ mA}$ a $3,3 \text{ V} \Rightarrow E_{\text{día}} \approx 2,4 \text{ Wh}$. Para $H_{\text{sol}} = 5 \text{ h}$ y $\eta_{\text{sist}} = 0,7$, un panel de $P_{\text{Wp}} \approx \frac{2,4}{5 \cdot 0,7} \approx 0,69 \text{ W}$ (se selecciona por encima, p. ej. 1–3 W para holgura y recarga).

4.3.4. Medición analógica de batería y panel

Divisor resistivo y escala

Para adaptar V_{in} al ADC ($V_{\text{ADC,max}}$), se usa:

$$V_{\text{ADC}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}, \quad k = \frac{R_1 + R_2}{R_2}, \quad V_{\text{in}} = k V_{\text{ADC}}. \quad (4.30)$$

Con $R_1 = R_2 = 100 \text{ k}\Omega$ y $V_{\text{in,max}} = 4,2 \text{ V} \Rightarrow V_{\text{ADC,max}} = 2,1 \text{ V}$ (seguro para el ESP32). La corriente por el divisor es

$$I_{\text{div}} = \frac{V_{\text{in}}}{R_1 + R_2} \approx \frac{4,2}{200 \text{ k}} \approx 21 \text{ }\mu\text{A}. \quad (4.31)$$

Conversión ADC y calibración

Con resolución $N = 12$ bits y referencia efectiva V_{ref} :

$$V_{\text{ADC}} = \frac{\text{counts}}{2^N - 1} V_{\text{ref}}, \quad V_{\text{in}} = k V_{\text{ADC}}. \quad (4.32)$$

Dado que el ADC del ESP32 presenta dispersión, se recomienda **calibración a dos puntos**:

$$V_{\text{in}} \approx a \cdot \text{counts} + b, \quad (4.33)$$

siendo a, b obtenidos con dos valores de referencia $\{\text{counts}, V_{\text{in}}\}$ medidos.

Filtro anti-alias y anti-ruido

El divisor con un condensador C forma un RC con resistencia equivalente $R_{\text{eq}} = R_1 \parallel R_2$:

$$f_c = \frac{1}{2\pi R_{\text{eq}} C}. \quad (4.34)$$

Con $R_1 = R_2 = 100 \text{ k}\Omega \Rightarrow R_{\text{eq}} = 50 \text{ k}\Omega$. Para $C = 100 \text{ nF}$, $f_c \approx 31,8 \text{ Hz}$; suficiente para promediar ruido de alta frecuencia sin afectar lecturas a baja tasa.

Incertidumbre por tolerancia

La ganancia k tiene error relativo aproximado (pequeñas tolerancias):

$$\frac{\Delta k}{k} \approx \sqrt{\left(\frac{\Delta R_1}{R_1}\right)^2 + \left(\frac{\Delta R_2}{R_2}\right)^2}. \quad (4.35)$$

Con resistencias 1 %, $\Delta k/k \approx 1,4 \%$. Usar 0.1–0.5 % mejora la exactitud.

4.3.5. Ley de Ohm y divisor de tensión (recordatorio)

$$V = I R, \quad V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}. \quad (4.36)$$

4.3.6. Energía, consumo por evento y muestreo

La energía consumida en un evento de transmisión WiFi de duración t_{tx} con corriente I_{tx} :

$$E_{\text{tx}} = V_{3,3} I_{\text{tx}} t_{\text{tx}}. \quad (4.37)$$

El promedio temporal a lo largo de un intervalo T con n eventos:

$$I_{\text{avg,tx}} = \frac{n I_{\text{tx}} t_{\text{tx}}}{T}. \quad (4.38)$$

Para señales ambientales de baja frecuencia, el criterio de Nyquist se satisface con periodos de muestreo del orden de minutos:

$$f_s \geq 2f_{\text{máx}}, \quad f_{\text{máx}} \approx 0,1 \text{ Hz}. \quad (4.39)$$

4.3.7. Accionamiento del relevador y protección

Con BJT (ejemplo). Para una bobina de I_{coil} y caída base-emisor $V_{BE} \approx 0,7 \text{ V}$, se busca saturación con $\beta_{\text{sat}} \approx 10$:

$$I_B \geq \frac{I_{\text{coil}}}{\beta_{\text{sat}}}, \quad R_B \approx \frac{V_{\text{GPIO}} - V_{BE}}{I_B}. \quad (4.40)$$

Ej.: $I_{\text{coil}} = 70 \text{ mA}$, $V_{\text{GPIO}} = 3,3 \text{ V} \Rightarrow I_B \geq 7 \text{ mA}$, $R_B \approx (3,3 - 0,7)/0,007 \approx 371 \Omega$ (usar 330–470 Ω). Se coloca un **diodo de rueda libre** en antiparalelo con la bobina para limitar sobrepicos:

$$V_{\text{rev}} \approx V_f, \quad E_{\text{bobina}} = \frac{1}{2} L I_{\text{coil}}^2. \quad (4.41)$$

Con MOSFET canal N. Se selecciona $R_{\text{DS(on)}}$ bajo y $V_{GS,th}$ compatible con 3.3 V. La disipación:

$$P_{\text{MOSFET}} \approx I_{\text{coil}}^2 R_{\text{DS(on)}}. \quad (4.42)$$

Contactos y cargas inductivas. Para cargas inductivas en los contactos, un *snubber* RC reduce el dv/dt . Una elección típica inicial es $R \approx 100 \Omega$, $C \approx 100 \text{nF}$ (ajustar por ensayo).

4.3.8. Integridad de alimentación y desacoplo

Se recomiendan condensadores de **desacoplo** de 100nF en cada CI y **banco** de $10\text{--}47 \mu\text{F}$ cerca del ESP32/regulador. La caída permitida ΔV ante un pulso de corriente ΔI durante Δt requiere:

$$C \geq \frac{\Delta I \Delta t}{\Delta V}. \quad (4.43)$$

Para picos WiFi rápidos conviene un bulk de $220\text{--}470 \mu\text{F}$ en el bus de 3.3 V y asegurar que el regulador soporte la demanda instantánea.

4.3.9. Bus I²C y 1-Wire: resistencias de *pull-up*

El tiempo de subida en I²C con resistencia R_P y capacitancia de bus C_b es

$$t_r \approx 0,8473 R_P C_b. \quad (4.44)$$

Para modo estándar (100 kHz) se requiere $t_r \lesssim 1 \mu\text{s}$. Con $R_P = 4,7 \text{k}\Omega$ y $C_b = 200 \text{ pF} \Rightarrow t_r \approx 0,8 \mu\text{s}$ ✓. Para 1-Wire, una *pull-up* de $4.7 \text{k}\Omega$ es habitual con uno a pocos DS18B20 y cable corto.

4.3.10. Caída de tensión en cables y protección

La caída en un tramo de resistencia R_{line} :

$$\Delta V = I R_{\text{line}}, \quad R_{\text{line}} = \rho \frac{L}{A}. \quad (4.45)$$

Para proteger la entrada de alimentación se recomienda diodo *Schottky* o PFET de inversión contra polaridad. La disipación en diodo:

$$P_D = I V_f. \quad (4.46)$$

4.3.11. Resumen de cumplimiento eléctrico

Con el ciclo de operación implementado (lectura y visualización proporcional, envío y *deep sleep*), el consumo promedio se mantiene bajo y permite autonomías de varios días con baterías del orden de $2\text{--}3 \text{ Ah}$. El dimensionamiento con *paneles de 1--3 W* es suficiente para reponer la energía diaria con holgura en climas de $H_{\text{sol}} \geq 5 \text{ h}$, y las mediciones de batería/panel cumplen con el rango y precisión requeridos gracias al divisor de tensión, filtro RC y calibración de ADC. El accionamiento del relevador con transistor/MOSFET y diodo de rueda libre garantiza conmutación segura, y el desacoplo adecuado mantiene la estabilidad del bus de 3.3 V ante picos de transmisión WiFi.

4.4. Sistema de control basado en máquina de estados

El comportamiento del sistema se formaliza mediante una máquina de estados finitos (FSM) con persistencia entre reinicios por *deep sleep*. Esta abstracción garantiza trazabilidad entre requisitos (energía, envío de datos, interfaz de usuario) y la implementación en el ESP32.

4.4.1. Definición formal

Sea la FSM de tipo Mealy

$$\mathcal{M} = (\mathcal{S}, \Sigma, \Lambda, \delta, \lambda, s_0),$$

donde \mathcal{S} es el conjunto de estados, Σ el conjunto de eventos, Λ el conjunto de acciones/salidas, $\delta : \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$ la función de transición, $\lambda : \mathcal{S} \times \Sigma \rightarrow \Lambda$ la función de salida y s_0 el estado inicial.

Estados (\mathcal{S}).

$$\mathcal{S} = \{\text{INIT, SEL_TIEMPO, DIA_INI, DIA_PARTE_RUN, DIA_PARTE_SLEEP, NOCHE_PREP, NOCHE_ENVIO, NOCHE_SLEEP, RESPALDOS, ERROR}\}.$$

Eventos (Σ).

$$\Sigma = \{\text{RTC_DIA, RTC_NOCHE, BTN_SHORT, BTN_LONG, T_PARTE_EXP, WIFI_OK, WIFI_FAIL, ACK_WEBAPP, ACK_FAIL, UMBRAL_RELE, LOW_BATT, PENDIENTES_OK}\}.$$

Variables y banderas persistentes.

$$\text{RTC_DATA_ATTR} \{\text{parteCiclo, tInicioParteUnix, intervaloParteUnix, envioNocturnoHecho, tiempoConfirmado, estabaEnModoNoche}\}.$$

4.4.2. Guardas y reglas de control

División de ciclo.

$$t_{\text{parte}} = \frac{t_{\text{ciclo}}}{6}, \quad \text{OLED}_{\text{on}} = 0,2 t_{\text{parte}} \text{ (al inicio y al final).}$$

Umbrales del relé (con histéresis).

$$\begin{aligned} \text{ON si } \text{modoDía} \wedge (V_{\text{bat}} < 3,60) \wedge (V_{\text{panel}} > 3,00), \\ \text{OFF si } \neg \text{modoDía} \vee (V_{\text{bat}} > 3,65) \vee (V_{\text{panel}} < 2,95). \end{aligned}$$

La histéresis $\Delta V \approx 0,05 \text{ V}$ evita conmutaciones por ruido.

Filtrado de medida (media móvil).

$$\bar{x}[k] = \alpha x[k] + (1 - \alpha) \bar{x}[k - 1], \quad \alpha \in [0,2, 0,4].$$

4.4.3. Tabla de transiciones (resumen)

Tabla 4.8: Transiciones principales de la FSM.

Estado	Evento/Guarda	Acción λ	Siguiente
INIT	RTC_DIA	init_sensores(), cargar.persistencia()	DIA_INI
INIT	RTC_NOCHE	init_sensores(), cargar.persistencia()	NOCHE_PREP
SEL_TIEMPO	BTN_SHORT	confirmar_tiempo(), calc_tparte()	DIA_INI
SEL_TIEMPO	BTN_LONG (≥ 5 s)	habilitar_edición()	SEL_TIEMPO
DIA_INI	true	mostrar_OLED(20 %), leer_sensores(), evaluar_rele()	DIA_PARTE_RUN
DIA_PARTE_RUN	T_PARTE_EXP	mostrar_OLED(20 %), guardar_resumen()	DIA_PARTE_SLEEP
DIA_PARTE_RUN	RTC_NOCHE	cerrar_rele(), resumen_OLED()	NOCHE_PREP
DIA_PARTE_SLEEP	RTC_DIA & parte < 6	programar_deep_sleep(t_resto)	DIA_PARTE_RUN
DIA_PARTE_SLEEP	parte = 6	reiniciar_ciclo()	DIA_INI
NOCHE_PREP	\neg envioNocturnoHecho	leer_sensores()	NOCHE_ENVIO
NOCHE_ENVIO	WIFI_OK & ACK_WEBAPP	set(envioNocturnoHecho), resumen_OLED()	NOCHE_SLEEP
NOCHE_ENVIO	(WIFI_FAIL \vee ACK_FAIL)	guardar_backup(), resumen_OLED()	NOCHE_SLEEP
NOCHE_SLEEP	true	deep_sleep(noche_restante)	INIT
RESPALDOS	PENDIENTES_OK	limpiar_backups()	estado_previo

4.4.4. Propiedades (invariantes y liveness)

- **Seguridad del actuador:** $\forall t$, modoNoche(t) \Rightarrow Relé = OFF.
- **Energía:** durante **DIA_PARTE_SLEEP** y **NOCHE_SLEEP** el ESP32 permanece en *deep sleep*; la OLED está apagada fuera de las ventanas del 20 %.
- **Idempotencia de envío nocturno:** si `envioNocturnoHecho` = true, no se repite el envío hasta el siguiente cambio noche \rightarrow día.
- **Progreso (liveness):** desde **DIA_PARTE_RUN** se alcanza **DIA_PARTE_SLEEP** en a lo más t_{parte} .

4.4.5. Temporización del ciclo diurno

Sea t_{ciclo} el tiempo confirmado por el usuario mediante el encoder. Entonces

$$t_{parte} = \frac{t_{ciclo}}{6}, \quad t_{OLED} = 0,2 t_{parte}, \quad t_{sleep,parte} = t_{parte} - t_{ejecución}.$$

Con persistencia de `parteCiclo` y `tInicioParteUnix` se evita reiniciar el ciclo al despertar.

4.4.6. Pseudocódigo de implementación (C++/ESP32)

```
enum Estado { INIT, SEL_TIEMPO, DIA_INI, DIA_PARTE_RUN, DIA_PARTE_SLEEP,
    NOCHE_PREP, NOCHE_ENVIO, NOCHE_SLEEP, RESPALDOS, ERROR };

void fsm_step(Event e){
    switch(estado){
        case INIT:
            if (esModoDia()) estado=DIA_INI;
            else estado=NOCHE_PREP;
```

```

break;

case SEL_TIEMPO:
    if (e==BTN_SHORT) { confirmarTiempo(); calcTparte(); estado=DIA_INI; }
    else if (e==BTN_LONG) editarTiempo();
    break;

case DIA_INI:
    ventanaOLED(); leerSensores(); evaluarRele();
    estado=DIA_PARTE_RUN; programarAlarma(tInicioParte+ tParte);
    break;

case DIA_PARTE_RUN:
    if (eventoRTCParte()) { ventanaOLED(); guardarResumen();
        estado=DIA_PARTE_SLEEP; }
    else if (esModoNoche()) { cerrarRele(); resumenOLED(); estado=NOCHE_PREP; }
    break;

case DIA_PARTE_SLEEP:
    if (++parteCiclo<=6) deepSleep(hastaSiguienteParte());
    else { reiniciarCiclo(); estado=DIA_INI; }
    break;

case NOCHE_PREP:
    if (!envioNocturnoHecho) { leerSensores(); estado=NOCHE_ENVIO; }
    else estado=NOCHE_SLEEP;
    break;

case NOCHE_ENVIO:
    if (postWebApp()) { envioNocturnoHecho=true; resumenOLED(); }
    else guardarBackup();
    estado=NOCHE_SLEEP;
    break;

case NOCHE_SLEEP:
    deepSleep(hastaAmanecer());
    break;
}
}

```

4.4.7. Plan de verificación

Asserts en firmware.

```

assert(¬modoNoche ⇒ tparte > 0),      assert(OLED_on ⇒ t ≤ 0,2 tparte),
assert(envioNocturnoHecho ⇒ no_reintentos).

```

Casos de prueba (extracto).

1. Transición día→noche con $\text{parteCiclo} \in \{1, \dots, 6\}$: relé en OFF, envío único y *deep sleep*.
2. Pérdida Wi-Fi en **NOCHE_ENVIO**: creación de respaldo y *flag* de pendientes.
3. Pulsación larga (≥ 5 s) restablece edición de tiempo tras despertar.
4. Histéresis: con V_{bat} oscilando en $[3,58, 3,62]$ V el relé no oscila.

4.4.8. Gestión de energía por estado

La corriente media aproximada por estado es

$$I_{\text{avg}} = \sum_{s \in \mathcal{S}} \alpha_s I_s, \quad \sum_s \alpha_s = 1,$$

donde α_s es la fracción de tiempo en el estado s e I_s su corriente típica (activo vs *deep sleep*). El diseño prioriza $\alpha_{\text{DIA_PARTE_SLEEP}}$ y $\alpha_{\text{NOCHE_SLEEP}}$ para maximizar la autonomía.

4.4.9. Manejo de fallos

- **Red**: reintento exponencial y **RESPALDOS** con reenvío en la siguiente oportunidad.
- **Sensores**: si falla un sensor crítico, colocar **ERROR** con relé en OFF y log de diagnóstico.
- **Integridad de datos**: incluir **id_ciclo** y sello horario RTC en el payload para idempotencia del backend.

4.5. Diseño del sistema de interfaz web (IoT)

La interfaz web permite visualizar el estado del sistema embebido, consultar históricos y, opcionalmente, emitir órdenes de control (p. ej., forzar el relevador). La solución se diseña ligera (HTML/CSS/JS), portable y de bajo costo, aprovechando el backend existente en Google Apps Script (GAS) y Google Sheets.

4.5.1. Objetivos de la HMI web

- **Observabilidad**: panel de control con variables clave (Temperatura, V_{bat} , V_{panel}).
- **Histórico**: consulta por rango de fechas
- **Comparativa**: Visualización de distintos gráfico a la vez.

4.5.2. Arquitectura

1. **ESP32** envía mediciones por HTTP POST a GAS → Google Sheets (*telemetría*).
2. **Interfaz web** (HTML/JS) consume HTTP GET (JSON) de GAS para mostrar: `/latest`, `/timeseries`.
3. **Control remoto (opcional)**: la web publica una orden vía POST `/cmd`. GAS la almacena en una *cola* (hoja “`cmd_queue`”), y el ESP32 la consulta periódicamente vía GET `/cmd_next?device=...`. Tras ejecutar, el ESP32 devuelve ACK (hoja “`cmd_log`”), garantizando idempotencia.

4.5.3. Modelo de datos (JSON)

La telemetría se serializa en JSON compacto:

```
{
  "ts_srv": "2025-08-15T11:23:40Z",    // timestamp del servidor (Apps Script)
  "ts_rtc": "2025-08-15T05:23:38-06:00", // tiempo RTC del ESP32
  "mode": "day|night",
  "part": 1..6,
  "T": 24.7, "H": 52.1, "P": 879.4, "Lux": 315.0,
  "Vbat": 3.81, "Vpv": 4.95,
  "relay": "ON|OFF",
  "retry": 0|1,           // si provino de respaldo
  "id_ciclo": "2025-08-15_05:00_abc123"
}
```

4.5.4. End-points del backend (GAS)

- **GET /latest?device={id}** → JSON con la última muestra válida.
- **GET /timeseries?device={id}&start={ISO}&end={ISO}&step={min}**: devuelve arreglo JSON con muestras agregadas (media/mín/máx) por ventana **step**.
- **POST /cmd** (opcional): {device, cmd, args, expires} → encola orden.
- **GET /cmd_next?device={id}** (ESP32): obtiene la siguiente orden pendiente no expirada.
- **POST /cmd_ack**: {cmd_id, status, note} registra ejecución (idempotente).

Seguridad: API key en cabecera `Authorization: Bearer <token>`, verificación de origen (CORS) y *rate limit* por IP.

4.5.5. Actualización “casi en tiempo real”

Se usa **SSE** (Server-Sent Events) o **polling** adaptativo:

$$T_{\text{refresh}} = \max(5 \text{ s}, 0.5 t_{\text{parte}}), \quad (4.47)$$

garantizando baja carga y sincronía con el ciclo del ESP32. Al recibir un lote se realizan promedios y suavizado:

$$\bar{x}_k = \alpha x_k + (1 - \alpha) \bar{x}_{k-1}, \quad \alpha \in [0.2, 0.4]. \quad (4.48)$$

4.5.6. Diseño de UI (vistas)

- **Dashboard**: tarjetas con KPIs, medidores de $V_{\text{bat}}/V_{\text{panel}}$.
- **Histórico**: selector `start/end`, gráfica multi-serie (T, H, P, Lux, V).
- **Comparativa**: Compara el comportamiento de los datos de diferentes sensores.

4.5.7. Accesibilidad y respuesta

Layout responsivo (*mobile-first*), contraste AA y *lazy loading* de series extensas.

4.5.8. Métricas y rendimiento

$$\text{latencia} = t_{\text{render}} - t_{\text{ts_srv}}, \quad \text{tasa_fallos} = \frac{N_{\text{FETCH_FAIL}}}{N_{\text{FETCH}}}. \quad (4.49)$$

Se fijan umbrales de alerta (p. ej., latencia > 5 s sostenida o tasa de fallos > 1 %).

4.5.9. Pruebas

- **Conectividad:** degradar red (modo *offline*) y verificar reconexión automática.
- **Carga:** histórica de 30 muestras \Rightarrow subsampling (`step`) y viewport virtual para mantener FPS.

CAPÍTULO 5

Implementación y Validación

5.0.1. Implementación del hardware (esquemáticos y PCB)

Esta sección documenta la implementación física final con base en la tarjeta mostrada en la Figura 5.1. La placa tiene dimensiones de **100 mm × 100 mm** y alberga el módulo **FireBeetle ESP32** junto con conectores para sensores, RTC, encoder y un sistema de relé.

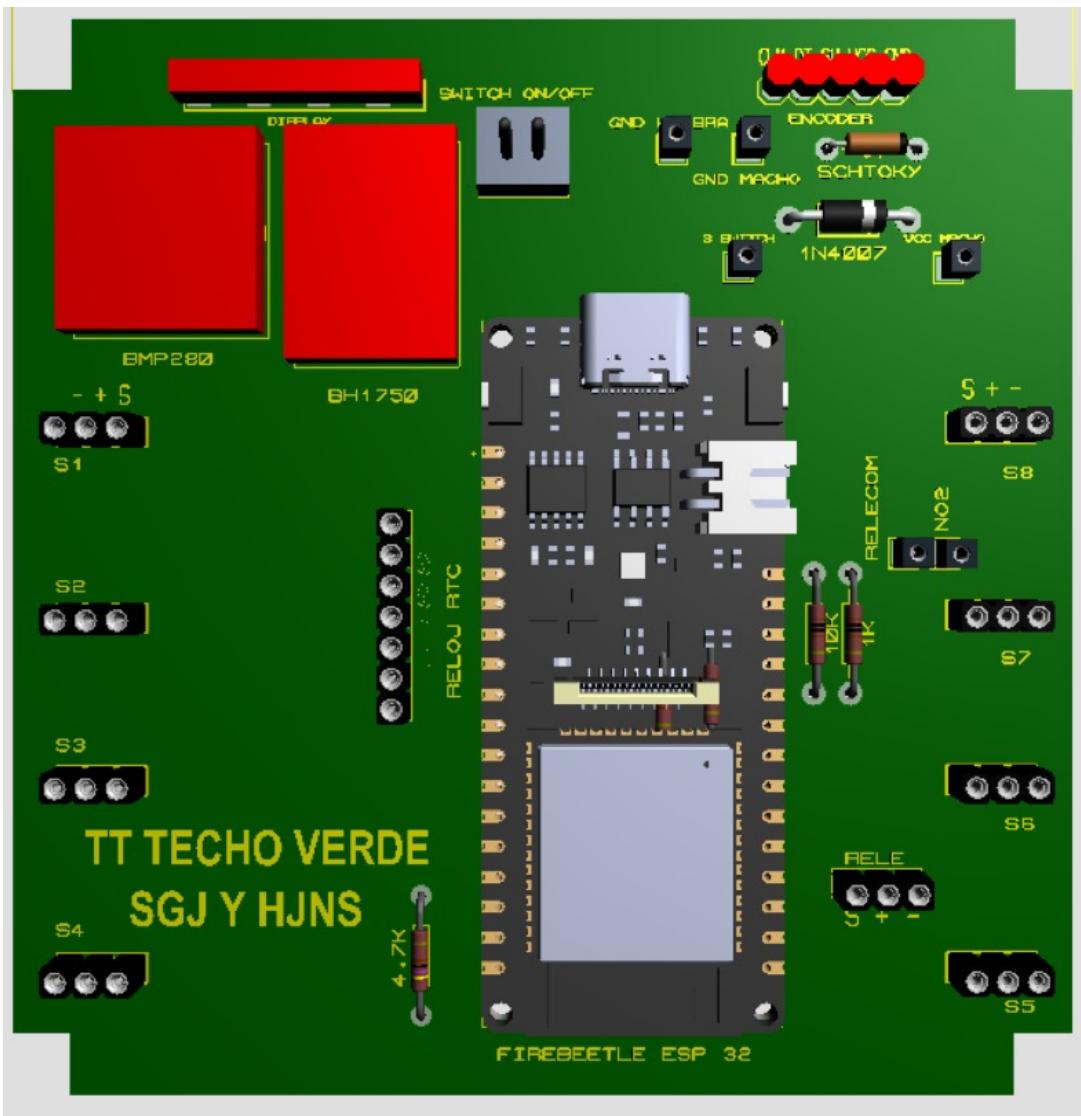


Figura 5.1: Vista superior de la PCB final (100 mm × 100 mm). Se aprecian: FireBeetle ESP32 (centro), cabeceras para BMP280 y BH1750 (superior-izquierda), cabecera *RELOJ RTC* (vertical al centro-izquierda), cabecera del *encoder* (superior-derecha), zona de *relé* y su driver (derecha) y ocho conectores S1–S8 para sondas DS18B20 (lados izquierdo y derecho).

5.0.2. Bloques implementados en la tarjeta

- **MCU:** alojamiento para **FireBeetle ESP32** con cuatro tornillos de fijación del módulo.
- **Sensores I²C:** cabeceras dedicadas para **BMP280** (presión) y **BH1750** (iluminancia) en la zona superior-izquierda.

- **Sondas DS18B20:** S1–S8 (tres pines c/u) distribuidos en ambos laterales. La serigrafía indica el orden de pines - + S (GND, V_{CC}, Señal 1-Wire).
- **RTC:** cabecera RELOJ RTC (vertical en el centro-izquierda) para el módulo basado en DS1307/DS3231 (I²C).
- **Interfaz de usuario:** cabecera del encoder rotativo (CLK, DT, SW, V_{CC}, GND) en la zona superior-derecha y punto para SWITCH ON/OFF.
- **Relé y potencia:** área de *driver* con transistor y diodos (**Schottky/1N4007**) para la bobina; conector RELE S + - para control/alimentación del relé y cabeceras de **contactos** (COM/NO/NC) para la carga DC.
- **Alimentación y protección:** ruta de potencia reforzada hacia el relé; diodos de protección (*flyback* y antipolaridad) próximos a los conectores de potencia.

5.0.3. Esquemáticos por bloques

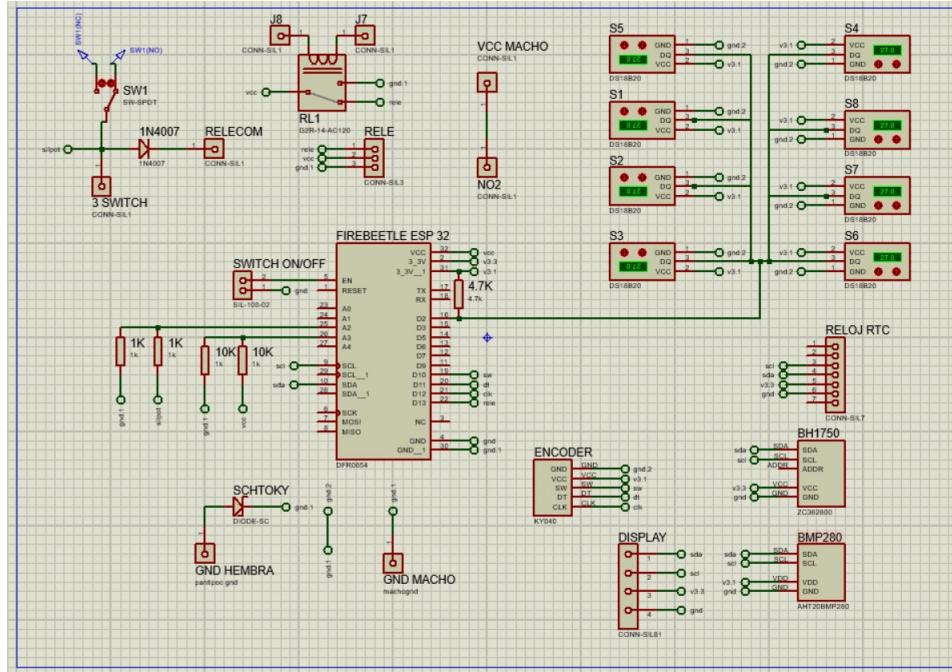


Figura 5.2: Esquemático

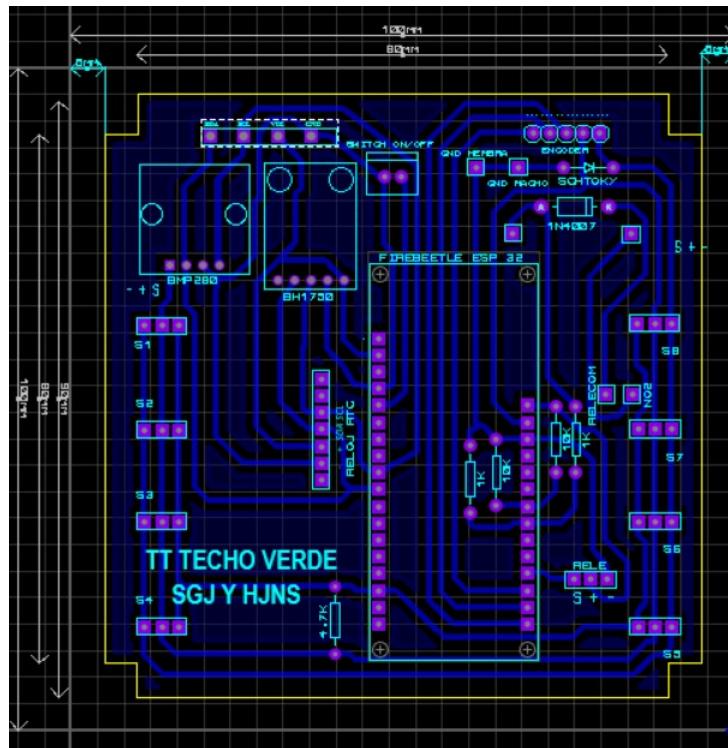


Figura 5.3: PCB

Figura 5.4: Esquemáticos.

5.0.4. Pinout y conectores de la PCB

Los conectores se nombran como en la serigrafía de la Figura 5.1. A continuación se resumen sus señales y uso.

Tabla 5.1: Pinout resumido de conectores principales.

Conector	Pin 1	Pin 2	Pin 3	Descripción / Notas
S1-S8	GND (-)	V _{CC} (+, 3.3 V)	Señal (S)	Entradas para sondas DS18B20 . El bus 1-Wire comparte S por grupo; <i>pull-up</i> típico 4.7 kΩ a 3.3 V cercano al MCU.
BMP280	V _{CC}	GND	SDA/SCL	Cabecera para el módulo BMP280 (I ² C).
BH1750	V _{CC}	GND	SDA/SCL	Cabecera para el módulo BH1750 (I ² C).
RELOJ RTC	V _{CC}	GND	SDA	SCL
Cabecera vertical RELOJ RTC (DS1307/DS3231). Puede incluir líneas auxiliares (SQW/BAT) según módulo.				
ENCODER	CLK	DT	SW	V _{CC} / GND
Cabecera para encoder rotativo con pulsador; <i>debounce</i> por <i>software</i> .				
RELE S + -	Señal (S)	V _{CC} (+)	GND (-)	Conejero de control/alimentación del relé. La bobina se comunica mediante transistor; diodo <i>flyback</i> cercano.
COM/NO/NC	COM	NO	NC	Terminales de potencia del relé para la carga DC (contacto seco).

Reglas de diseño y ruteo aplicado

- **Planos y retorno:** plano de GND continuo bajo FireBeetle y sensores; retorno corto para las mediciones analógicas y el bus 1-Wire.
- **Potencia:** pistas del *driver* de relé y contactos dimensionadas para la corriente prevista; separación aumentada respecto de señales de baja potencia.
- **Protecciones:** diodo Schottky/1N4007 en la ruta de entrada y *flyback* en la bobina, tal como se muestra en la zona del relé.
- **EMC/ruido:** desacoplos de 100 nF por IC y 10–47 μF en el bus de 3.3 V; pistas I²C cortas; *pull-ups* de 4.7–10 kΩ para I²C y 1-Wire.
- **Módulo ESP32:** cuatro tornillos del portamódulo mantienen rigidez mecánica y masa térmica para el conector USB.

5.0.5. Inclusión de Librerías y Credenciales

Este bloque inicial incluye todas las librerías requeridas para el funcionamiento del sistema. El código se muestra a continuación:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_AHTX0.h>
#include <Adafruit_BMP280.h>
#include <BH1750.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <LittleFS.h>
```

```

#include <RTClib.h>
#include <WiFiClientSecure.h>

// Credenciales
const char* ssid = "Totalplay-2.4G-43C0";
const char* password = "4EFtUaebfdE9GtA9";

// URL actual del Google Apps Script
const char* WEB_APP_URL = "https://script.google.com/macros/s/AKfycbxEPunM9L0781G0MRiR8F-oxIgTuJH";

```

Explicación

- **Adafruit_SSD1306** y **Adafruit_GFX**: control de la pantalla OLED.
- **AHT10, BMP280** y **BH1750**: lectura de temperatura, humedad, presión e iluminación.
- **DallasTemperature** y **OneWire**: manejo de los sensores DS18B20.
- **WiFi, HTTPClient** y **WiFiClientSecure**: conexión a internet y envío de datos al servidor.
- **LittleFS**: respaldo en memoria interna del ESP32.
- **RTClib**: control de tiempo mediante el RTC DS1307.

Posteriormente se definen las credenciales WiFi y la URL del script en la nube. Estos parámetros permiten al dispositivo conectarse a la red y enviar la información recolectada hacia Google Sheets.



Figura 5.5: Comunicación ESP32 → WiFi → Google Apps Script → Google Sheets

5.0.6. Estructura de Comunicación y Envío de Datos

En este bloque se definen las funciones encargadas de enviar datos al servidor (Google Apps Script) mediante HTTP POST sobre HTTPS.

Código

```

struct PostResult {
    int http;
    bool ack;
    String body;
};

PostResult postAWebApp(const String& payload) {
    static unsigned long lastPostMs = 0;
    unsigned long nowMs = millis();

```

```

if (nowMs - lastPostMs < 1500) delay(1500 - (nowMs - lastPostMs));
lastPostMs = millis();

WiFiClientSecure client;
client.setInsecure();

HTTPClient http;
http.setTimeout(12000);
http.setFollowRedirects(HTTPC_DISABLE_FOLLOW_REDIRECTS);
http.setReuse(false);

if (!http.begin(client, WEB_APP_URL)) {
    return { -90, false, "" };
}

http.addHeader("Content-Type", "application/x-www-form-urlencoded");

int code = http.POST(payload);
String body = http.getString();
http.end();

bool ok = (code >= 200 && code < 300) || (code >= 300 && code < 400);
if (!ok) {
    String upper = body;
    upper.toUpperCase();
    if (upper.indexOf("OK") >= 0) ok = true;
    if (code == -11) ok = true;
}
return { code, ok, body };
}

```

Explicación

- **Estructura PostResult:** encapsula el resultado del envío (código HTTP, *ack* lógico y cuerpo de respuesta).
- **postAWebApp:** construye una conexión HTTPS al Web App de GAS, envía el *payload* en formato *application/x-www-form-urlencoded* y retorna un *PostResult*.

Características importantes

- **Rate limiting:** se impone una espera mínima de **1500 ms** entre envíos para evitar saturar la red/endpoint.
- **TLS simplificado:** `WiFiClientSecure` en modo *insecure* para facilitar el HTTPS en entorno embebido.
- **Criterio de éxito tolerante:** además de respuestas *2xx/3xx*, se consideran exitosos ciertos casos en los que el cuerpo contiene “OK” o cuando aparece el timeout controlado -11 (caso

típico donde el servidor procesó la petición).

1	Hora de recepción	Timestamp	dia o noche y reintento
2	26/8/2025 19:05:16	2025-08-26 10:05:01	<input checked="" type="checkbox"/> Día
3	28/8/2025 0:06:00	2025-08-28 0:03:22	Noche <input type="checkbox"/> Reintento
4	28/8/2025 0:06:05	2025-08-28 0:05:42	Noche <input type="checkbox"/> Reintento
5	28/8/2025 1:06:08	2025-08-28 0:36:22	Noche <input type="checkbox"/> Reintento
6	28/8/2025 1:06:24	2025-08-28 1:05:54	Noche <input type="checkbox"/> Reintento
7	28/8/2025 1:36:03	2025-08-28 1:35:51	Noche
8	28/8/2025 2:05:53	2025-08-28 2:05:36	Noche

Figura 5.6: Captura de Google Sheets mostrando datos enviados correctamente desde el ESP32.

5.0.7. Definición de Pines, Sensores y Variables Globales

Este bloque concentra todas las constantes de hardware, las instancias de librerías (sensores, pantalla, reloj), y el estado global que comparten las distintas rutinas del programa. Al mantener esta información en un único lugar se facilita el mantenimiento y la trazabilidad del diseño.

5.0.8. Pantalla OLED y dimensiones

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

Descripción: Se declara un panel OLED de 128×64 px basado en SSD1306, conectado por I²C. Esta pantalla se utiliza para mostrar páginas de datos (sensores, voltajes, estado de envío y temporizadores), evitando depender solamente del puerto serial durante operación en campo.

5.0.9. Entradas del encoder y pulsador

```
#define CLK 16
#define DT 17
#define SW 4
```

Descripción: El encoder rotativo permite seleccionar el tiempo de muestreo y navegar entre páginas.

- CLK y DT leen el giro (sentido horario/antihorario).
- SW actúa como botón: “tap” confirma el tiempo seleccionado; “hold” ejecuta acciones especiales (ej. borrar respaldos en LittleFS).

Variables asociadas:

```

int indexSeleccion = 0;           // índice crudo del encoder
const int MAX_INDEX = 105;        // tope del selector
bool modoVisualizacion = false;
unsigned long btnPressedAt = 0;
int paginaActual = 0;           // 0..4: páginas OLED
int lastClkState = HIGH;

```

5.0.10. Puntos de medición analógicos (panel y batería)

```

#define PIN_PANEL 34
#define PIN_BATERIA 35

```

Descripción: Los pines ADC (GPIO34/GPIO35) reciben los voltajes escalados por divisores resistivos para no exceder los 3.3 V del ADC.

Escalado en el código:

```

float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;

```

Interpretación:

- El factor 2 proviene del divisor ($\approx R1 = R2$).3.3 es la referencia de tensión del ADC.
- 4095 es el valor máximo de 12 bits.

Cálculo de porcentajes:

```

float pb = vb / 4.2 * 100.0; // % batería (4.2 V = 100%)
float pp = vp / 5.0 * 100.0; // % panel (5.0 V plena irradiancia)

```

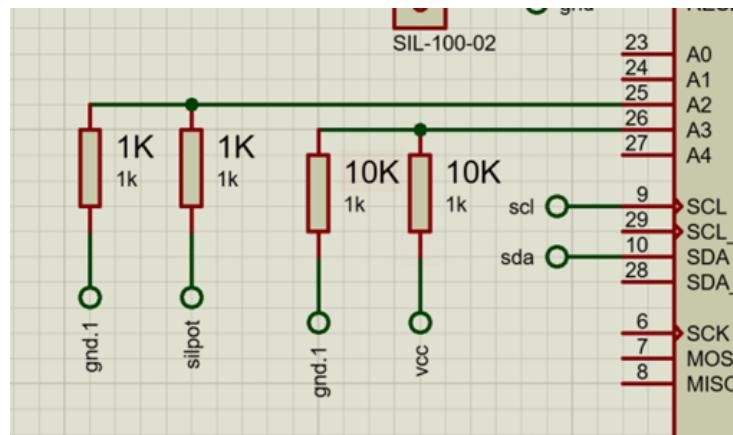


Figura 5.7: Divisor de tensión y entrada ADC.

5.0.11. Bus 1-Wire de sensores DS18B20

```

#define ONE_WIRE_BUS 25
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensoresDS18B20(&oneWire);

```

```

DeviceAddress sensores[8] = {
  { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x66, 0x24, 0xC5 }, // S1
  { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x65, 0x5C, 0xAA }, // S2
  ...
};


```

Descripción: El bus 1-Wire centraliza hasta 8 termómetros DS18B20 sobre un único pin (GPIO25). Se emplean direcciones físicas para identificar cada sonda y nombres asociados para mostrar en OLED.

5.0.12. Sensores ambientales por I²C

```

Adafruit_AHTX0 aht;
Adafruit_BMP280 bmp;
BH1750 lightMeter;


```

Descripción:

- AHT10 → humedad relativa y temperatura del aire.
- BMP280 → presión barométrica y temperatura.
- BH1750 → iluminancia en lux.

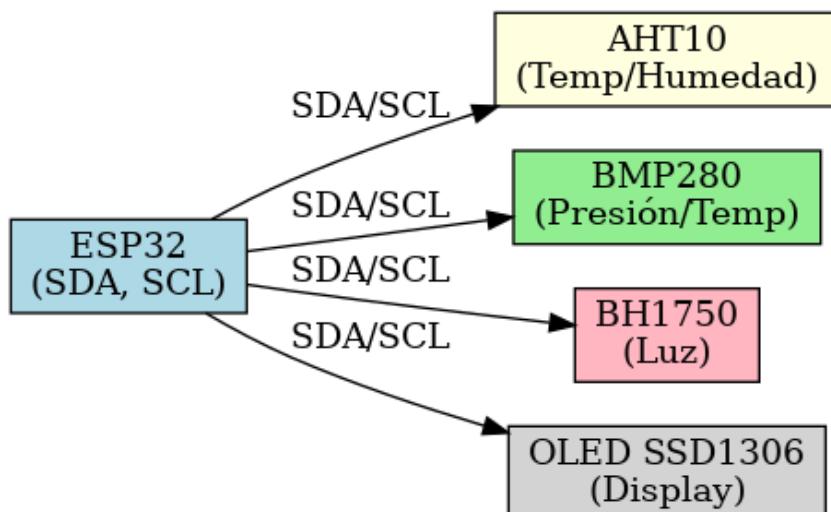


Figura 5.8: Bus I²C compartido para AHT10, BMP280, BH1750 y OLED.

5.0.13. Relevador y umbrales de protección

```

#define RELAY_PIN 12
#define UMBRAL_BATERIA 3.6
#define UMBRAL_PANEL 3.0
bool releActivo = false;


```

Descripción: El relevador gobierna la ruta de carga/uso en función de:

- Horario (día/noche).
- Voltaje batería ≥ 3.6 V (protección sobredescarga).
- Voltaje panel ≥ 3.0 V (potencia suficiente).

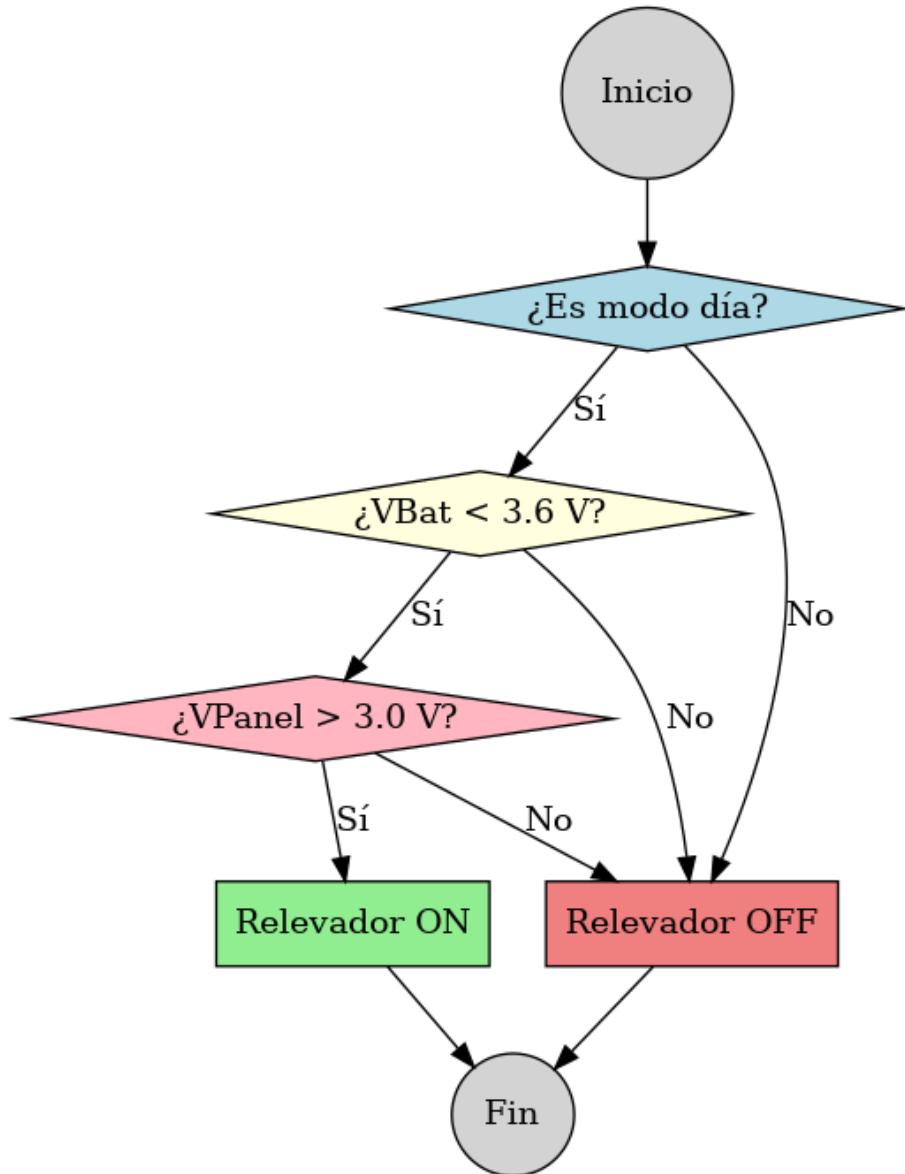


Figura 5.9: Decisión del relevador con umbrales VBAT y VPANEL.

5.0.14. Reloj de tiempo real (RTC) y ventanas día/noche

```

RTC_DS1307 rtc;
const int HORA_INICIO_DIA = 8;
const int MINUTO_INICIO_DIA = 0;
const int HORA_FIN_DIA = 16;
const int MINUTO_FIN_DIA = 0;
  
```

```
bool esModoDia = false;
bool esModoNoche = false;
```

Descripción: Se establece una ventana horaria para gobernar:

- Día → control cíclico, activación de relevador, envío de datos.
- Noche → envío único, deep sleep.

5.0.15. Estado global y persistencia entre deep sleeps

```
RTC_DATA_ATTR bool tiempoConfirmado = false;
RTC_DATA_ATTR int timerActual = 1;
RTC_DATA_ATTR int parteCiclo = 1;
RTC_DATA_ATTR bool envioNocturnoHecho = false;
...
```

Descripción:

- Variables con RTC_DATA_ATTR sobreviven a deep sleep.
- Permiten retomar ciclos sin perder contexto.
- Incluyen banderas de control (día/noche, reintentos, estado de envío, etc.).

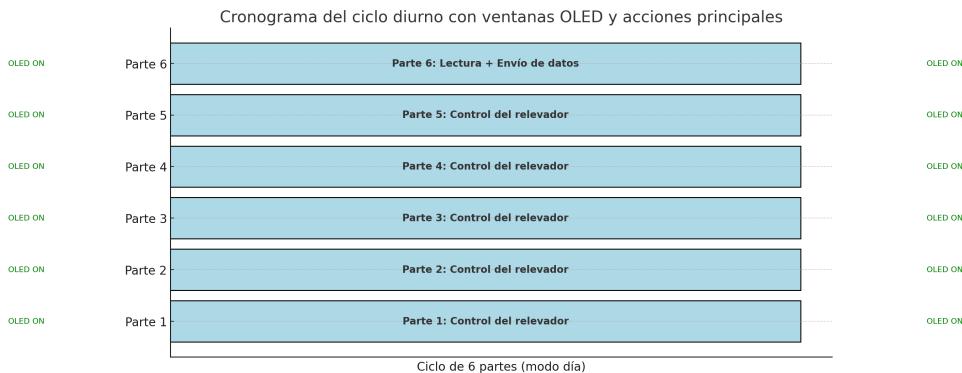


Figura 5.10: División en 6 partes del ciclo diurno y ventanas de visualización OLED.

5.0.16. Resumen visual del bloque

- Entradas/salidas: encoder (CLK/DT/SW), relevador (GPIO12), ADC batería/panel.
- Buses: 1-Wire (DS18B20), I²C (AHT10, BMP280, BH1750, OLED).
- Persistencia: variables críticas en RTC_DATA_ATTR.
- UI: OLED con páginas de diagnóstico.

5.1. Funciones Auxiliares y Utilidades

Este conjunto de funciones provee herramientas de bajo nivel que permiten simplificar la lógica principal del programa. Aunque no ejecutan tareas completas de ciclo o comunicación, resultan esenciales para garantizar un flujo correcto de datos.

5.1.1. Codificación de texto para envío HTTP

```
// -- URL encoder para application/x-www-form-urlencoded --
String urlEncode(const String& s) {
    String out;
    out.reserve(s.length() * 3);
    const char* hex = "0123456789ABCDEF";
    for (size_t i = 0; i < s.length(); ++i) {
        char c = s[i];
        if ((‘a’ <= c && c <= ‘z’) || (‘A’ <= c && c <= ‘Z’) ||
            (‘0’ <= c && c <= ‘9’) || c == ‘-’ || c == ‘_’ ||
            c == ‘.’ || c == ‘~’)
            out += c;
        else if (c == ‘ ’) out += ‘+’;
        else {
            out += ‘%’;
            out += hex[(c >> 4) & 0xF];
            out += hex[c & 0xF];
        }
    }
    return out;
}
```

Explicación: Convierte caracteres especiales en formato `application/x-www-form-urlencoded`, garantizando que los parámetros (sean fechas u otros) no se corrompan en el envío. Ejemplo: espacio → “+” y dos puntos “:” → “%3A”.

5.1.2. Conversión de índice del encoder a minutos/horas

```
int calcularTiempoDesdeIndex(int idx) {
    if (idx < 60) return idx + 1;
    return 90 + (idx - 60) * 30;
}
```

Explicación:

- Índices 0–59 → minutos directos.
- Índices ≥ 60 → saltos de 30 min.

Ejemplo:

- $idx = 0 \rightarrow 1$ min.

- $\text{idx} = 59 \rightarrow 60 \text{ min.}$
- $\text{idx} = 60 \rightarrow 90 \text{ min.}$
- $\text{idx} = 61 \rightarrow 120 \text{ min.}$

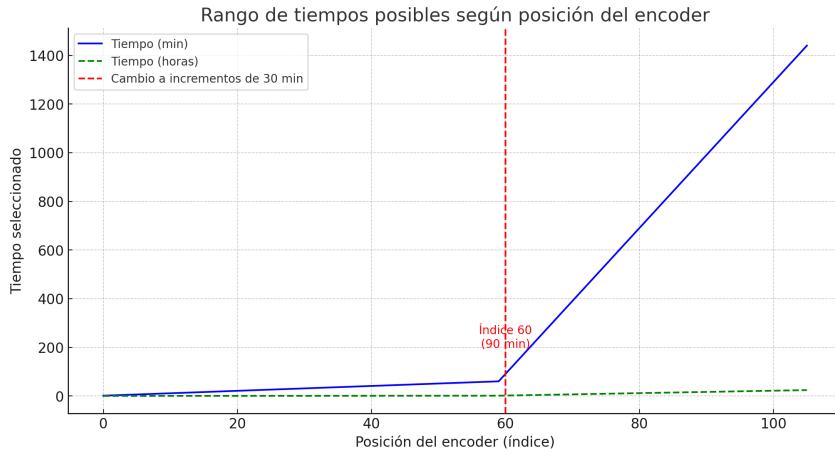


Figura 5.11: Rango de tiempos posibles según posición del encoder.

5.1.3. Lectura de BMP280 (temperatura y presión)

```
void leerBMP(float& bmpTemp, float& presion_hPa) {
    bmp.takeForcedMeasurement(); // dispara 1 medición
    bmpTemp = bmp.readTemperature();
    presion_hPa = bmp.readPressure() / 100.0;
}
```

Explicación: Modo forzado → ahorro energético y lectura simultánea de temperatura y presión.

5.1.4. Lectura de BH1750 (intensidad lumínica)

```
float leerLuz() {
    lightMeter.configure(BH1750::ONE_TIME_HIGH_RES_MODE);
    delay(180); // tiempo de conversión típico
    float lx = lightMeter.readLightLevel();
    return lx < 0 ? 0 : lx; // evita valores negativos
}
```

Explicación: Cada llamada genera una única lectura en alta resolución. El sensor entra en bajo consumo automáticamente tras la medición.

5.1.5. Persistencia de respaldos (LittleFS)

Funciones de soporte:

- `guardarDatosPendientes(payload)` → almacena en `/pendientes.txt`.
- `verContenidoLittleFS()` → imprime en consola el contenido pendiente.

- `contarLineasLittleFS()` → cuenta registros pendientes.
- `hayPendientesLittleFS()` → indica si existen datos sin enviar.



Figura 5.12: Flujo de respaldo y reintento: Normal → Sin WiFi → LittleFS → Reintento → Envío exitoso.

5.2. Manejo de Pantalla OLED

La pantalla OLED es la interfaz visual del sistema. Aunque el envío de datos se realiza en segundo plano, la visualización local permite al usuario conocer en tiempo real el estado de sensores, voltajes y ciclos de operación.

5.2.1. Encendido y apagado del display

```

void encenderOLED() {
    if (!displayInicializado) {
        if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
            Serial.println("OLED no respondio en 0x3C");
            return;
        }
        displayInicializado = true;
        display.clearDisplay();
        display.display();
    } else {
        display.ssdi306_command(SSD1306_DISPLAYON);
    }
    oledOn = true;
}

void apagarOLED() {
    if (!displayInicializado) return;
    display.ssdi306_command(SSD1306_DISPLAYOFF);
    oledOn = false;
}
  
```

Explicación:

- `encenderOLED()` inicializa y activa el display (dirección I²C: 0x3C).
- `apagarOLED()` lo suspende para reducir consumo.
- La bandera `oledOn` indica si la pantalla está encendida.

5.2.2. Visualización del tiempo seleccionado

```
void mostrarTiempo() {
    encenderOLED();
    if (!displayInicializado) return;
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print("Tiempo:");

    display.setCursor(0, 20);
    if (timerPot < 60) {
        display.print(timerPot);
        display.print(" min");
    } else {
        display.print(timerPot / 60.0, 1);
        display.print(" h");
    }

    display.setTextSize(1);
    display.setCursor(0, 50);
    display.print("Tap=OK Hold=Salir");
    display.display();
}
```

Explicación:

- Muestra el tiempo elegido con el encoder.
- Hasta 60 → minutos, a partir de 60 → horas.
- Incluye instrucciones de uso: Tap = confirmar, Hold = salir.



Figura 5.13: OLED mostrando el menú de selección de tiempo.

5.2.3. Páginas de sensores DS18B20

```
void mostrarPaginaSensoresDS(int inicio) {
    encenderOLED();
    if (!displayInicializado) return;

    sensoresDS18B20.requestTemperatures();
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println(inicio == 0 ? "Sensores izquierda" : "Sensores derecha");

    for (int i = inicio; i < inicio + 4; i++) {
        float temp = sensoresDS18B20.getTempC(sensores[i]);
        display.printf("%s: %.1f C\n", nombressensores[i], temp);
    }
    display.display();
}
```

Explicación:

- Muestra 4 sensores por página: 1–4 (izquierda), 5–8 (derecha).
- Incluye nombre y valor en °C.

5.2.4. Páginas de sensores ambientales

```
void mostrarPaginaAmbientales() {
    encenderOLED();
    if (!displayInicializado) return;

    sensors_event_t humidity, temp;
    aht.getEvent(&humidity, &temp);
    float luz = leerLuz();
    float bmpTemp, presion;
    leerBMP(bmpTemp, presion);

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.printf("AHT: %.1f C\n", temp.temperature);
    display.printf("Hum: %.1f %%\n", humidity.relative_humidity);
    display.printf("BMP: %.1f C\n", bmpTemp);
    display.printf("Pres: %.1f hPa\n", presion);
    display.printf("Luz: %.1f lx\n", luz);
    display.display();
}
```

Explicación: Muestra variables de AHT10 (T/HR), BMP280 (T/P) y BH1750 (lux).

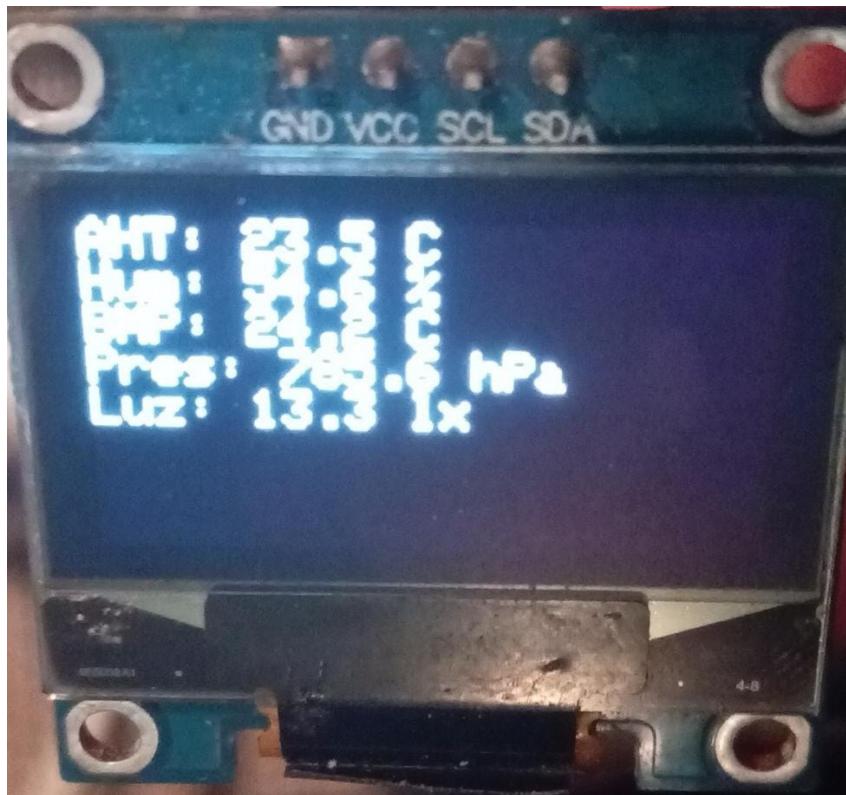


Figura 5.14: Pantalla OLED mostrando lecturas ambientales.

5.2.5. Página de voltajes y estado del relevador

```
void mostrarPaginaVoltajes() {
    encenderOLED();
    if (!displayInicializado) return;

    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    bool bateriaBaja = vb < UMBRAL_BATERIA;
    bool panelSuficiente = vp > UMBRAL_PANEL;

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.printf("Bat: %.2f V (%.0f%%)\n", vb, pb);
    display.printf("Panel: %.2f V (%.0f%%)\n", vp, pp);
    display.printf("Rele: %s\n", releActivo ? "ON" : "OFF");

    display.setCursor(0, 33);
    display.printf("VBaja : %s\n", bateriaBaja ? "SI" : "NO");
    display.printf("VPnl : %s\n", panelSuficiente ? "SI" : "NO");
```

```
display.display();
}
```

Explicación: Muestra estado de batería, panel y relevador, con condiciones de activación.

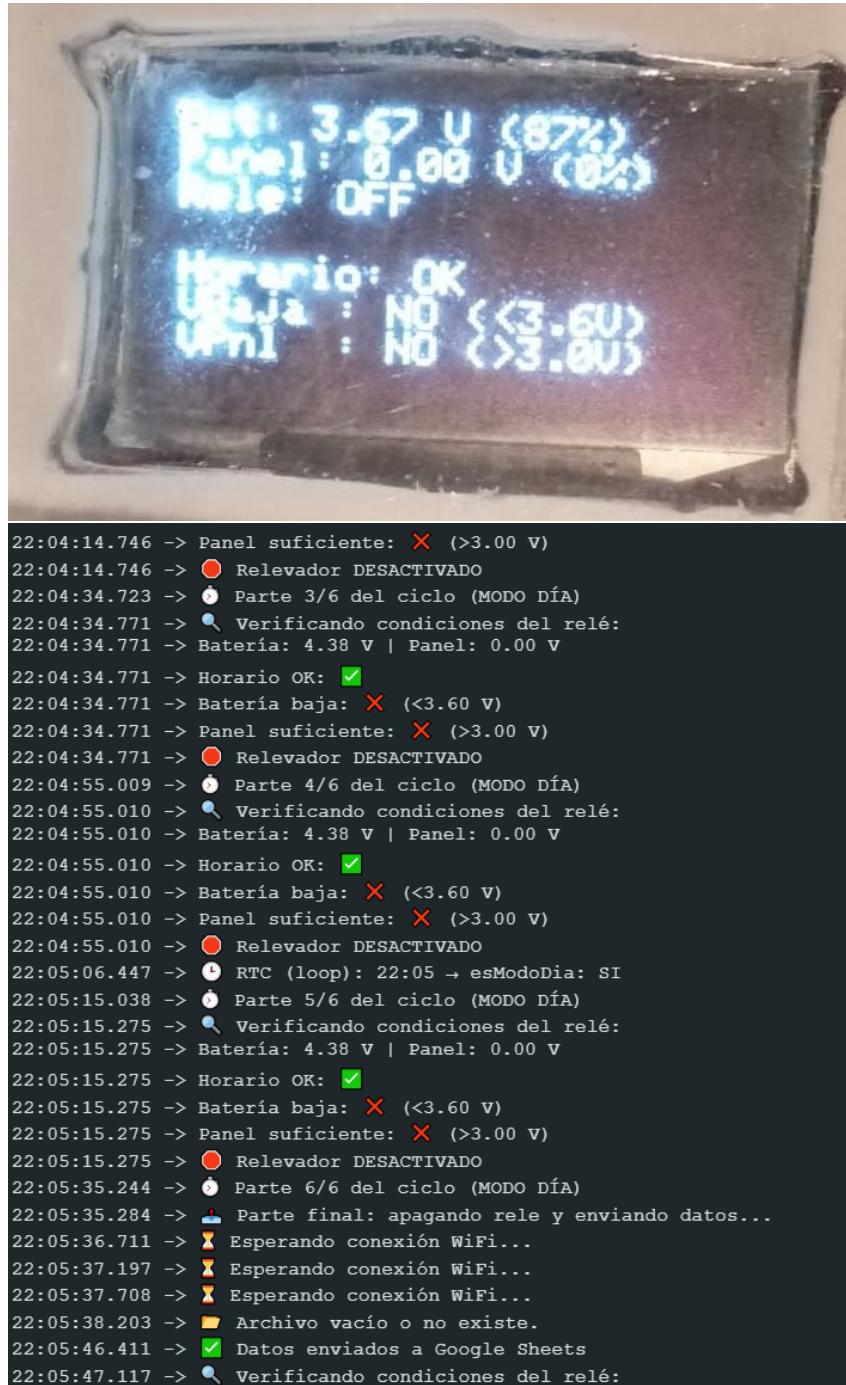


Figura 5.15: Voltajes del prototipo en funcionamiento.

5.2.6. Página de envío de datos

```
void mostrarPaginaEnvio() {
```

```
encenderOLED();
if (!displayInicializado) return;

DateTime now = rtc.now() + TimeSpan(0, 0, 3, 0);
display.clearDisplay();
display.setTextSize(1);

display.setCursor(0, 0);
display.printf("%02d:%02d:%02d %02d/%02d/%04d\n",
              now.hour(), now.minute(), now.second(),
              now.day(), now.month(), now.year());

display.setCursor(0, 15);
if (timerActual < 60) {
    display.printf("Tiempo sel.: %d min\n", timerActual);
} else {
    display.printf("Tiempo sel.: %.1f h\n", timerActual / 60.0);
}

display.setCursor(0, 30);
display.printf("Ult. envío: %lds\n", (millis() - tEnvio) / 1000);

display.setCursor(0, 40);
display.printf("Parte %d/6\n", parteCiclo);

display.setCursor(0, 55);
display.print(estadoUltimoEnvio);

display.display();
}
```

Explicación: Incluye:

- Fecha/hora RTC.
- Tiempo de muestreo confirmado.
- Tiempo desde el último envío.
- Parte actual del ciclo.
- Estado de transmisión (OK, fallo, sin WiFi).

5.3. Control del Relevador y Gestión Energética

El relevador funciona como un interruptor controlado por software, encargado de habilitar o deshabilitar la alimentación a ciertos subsistemas en función del estado energético y del horario de operación (día o noche).

5.3.1. Función principal de control

```

void controlarRelé(float vb, float vp) {
    bool dentroHorario = esModoDia;           // solo se permite control en día
    bool bateriaBaja = vb < UMBRAL_BATERIA; // protección a la batería
    bool panelSuficiente = vp > UMBRAL_PANEL; // asegurar aporte del panel

    Serial.println("Verificando condiciones del relé:");
    Serial.printf("Batería: %.2f V | Panel: %.2f V\n", vb, vp);

    Serial.printf("Horario OK: %s\n", dentroHorario ? "SI" : "NO");
    Serial.printf("Batería baja: %s (<%.2f V)\n", bateriaBaja ? "SI" : "NO", UMBRAL_BATERIA);
    Serial.printf("Panel suficiente: %s (>%.2f V)\n", panelSuficiente ? "SI" : "NO", UMBRAL_PA

    if (dentroHorario && bateriaBaja && panelSuficiente) {
        digitalWrite(RELAY_PIN, HIGH);
        releActivo = true;
        Serial.println("Relevador ACTIVADO");
    } else {
        digitalWrite(RELAY_PIN, LOW);
        releActivo = false;
        Serial.println("Relevador DESACTIVADO");
    }
}

```

Explicación:

- **dentroHorario**: el control sólo se habilita durante el modo día.
- **bateriaBaja**: se activa si $V_{bat} < 3,6$ V.
- **panelSuficiente**: se activa si $V_{panel} > 3,0$ V.

Condición de activación del relevador:

$$Activar\ Relé \iff esModoDia \wedge (V_{bat} < 3,6V) \wedge (V_{panel} > 3,0V)$$

Esto asegura que sólo se cargue la batería cuando realmente es necesario y cuando el panel lo permite, evitando ciclos de carga innecesarios.

5.3.2. Variables de soporte

```

#define RELAY_PIN 12
#define UMBRAL_BATERIA 3.6
#define UMBRAL_PANEL 3.0
bool releActivo = false;

```

Descripción:

- **RELAY_PIN** → GPIO12 conectado al módulo de relevador.
- **UMBRAL_BATERIA** → voltaje mínimo aceptable antes de recarga.

- `UMBRAL_PANEL` → voltaje mínimo del panel para carga útil.
- `releActivo` → bandera de estado actual del relevador.

5.3.3. Interacción con el ciclo de trabajo

- Durante las partes 1 a 5 del ciclo diurno, la lógica del relevador es la acción principal: medir voltajes y decidir activación.
- En la parte 6, el relevador se desactiva para realizar lectura completa de sensores y envío de datos a la nube, asegurando estabilidad en las mediciones.

5.3.4. Visualización en OLED

La página de voltajes (sección anterior) incluye también el estado del relevador, permitiendo confirmar en campo si el sistema está cargando la batería o permanece en reposo.

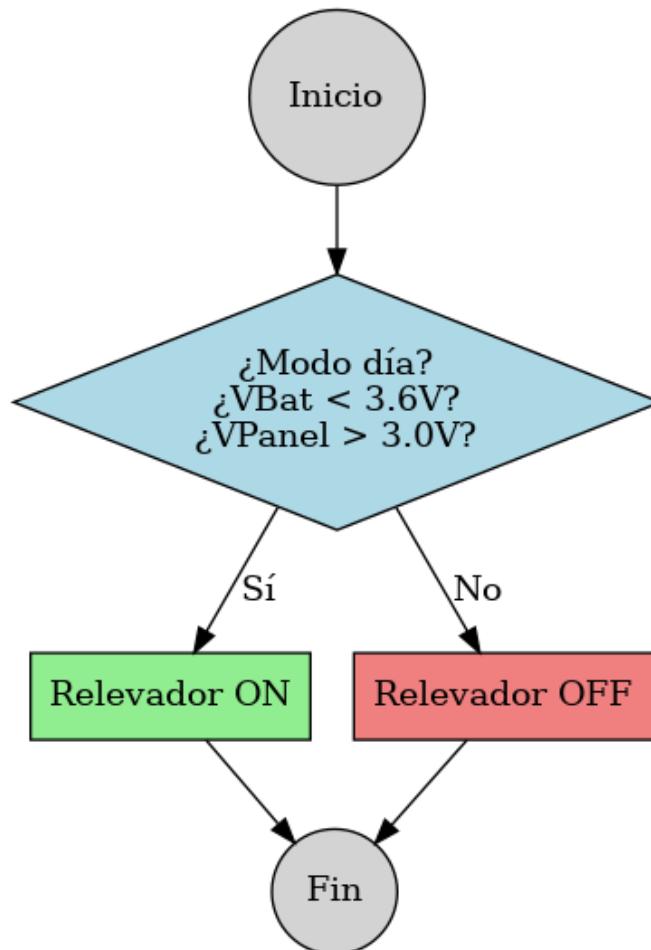


Figura 5.16: Decisión del relevador: si es modo día, $V_{bat} < 3,6$ V y $V_{panel} > 3,0$ V → Rele ON; en otro caso → Rele OFF.

5.4. Modo Noche

El modo noche está diseñado para reducir al mínimo el consumo energético. A diferencia del día, en el que se realizan múltiples lecturas parciales y control del relevador, en la noche se hace un único envío de datos y después el sistema entra en *deep sleep* hasta el siguiente evento programado.

5.4.1. Detección del modo noche

```
DateTime inicioDia(now.year(), now.month(), now.day(),
                   HORA_INICIO_DIA, MINUTO_INICIO_DIA, 0);
DateTime finDia(now.year(), now.month(), now.day(),
                HORA_FIN_DIA, MINUTO_FIN_DIA, 0);

esModoDia    = (ahoraUnix >= inicioDia.unixtime()) &&
                (ahoraUnix < finDia.unixtime());
esModoNoche = !esModoDia;
```

Cuando `esModoNoche = true`, el sistema se prepara para ejecutar la rutina especial.

5.4.2. Ejecución de la rutina nocturna

```
void ejecutarModoNoche() {
    DateTime now = rtc.now();

    // Calcular el próximo inicio de modo día
    DateTime inicioDiaHoy(now.year(), now.month(), now.day(),
                           HORA_INICIO_DIA, MINUTO_INICIO_DIA, 0);
    DateTime inicioDiaManana = inicioDiaHoy + TimeSpan(1, 0, 0, 0);
    DateTime proximoInicioDia =
        (now.unixtime() < inicioDiaHoy.unixtime()) ?
        inicioDiaHoy : inicioDiaManana;

    if (!tiempoConfirmado || envioNocturnoHecho) return;

    // 1. Leer sensores
    sensoresDS18B20.requestTemperatures();
    float tempDS[8];
    for (int i = 0; i < 8; i++) tempDS[i] = sensoresDS18B20.getTempC(sensores[i]);

    sensors_event_t humidity, temp;
    aht.getEvent(&humidity, &temp);
    float bmpTemp, presion;
    leerBMP(bmpTemp, presion);
    float luz = leerLuz();
    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
```

```

// 2. Construir payload
String payload = "s1=" + String(tempDS[0], 2);
for (int i = 1; i < 8; i++) {
    payload += "&s" + String(i+1) + "=" + String(tempDS[i], 2);
payload += "&aht=" + String(temp.temperature, 2);
payload += "&hum=" + String(humidity.relative_humidity, 2);
payload += "&bmp=" + String(bmpTemp, 2);
payload += "&presion=" + String(presion, 2);
payload += "&luz=" + String(luz, 2);
payload += "&vb=" + String(vb, 2);
payload += "&vp=" + String(vp, 2);
payload += "&intervalo=" + String(timerActual);
payload += "&modo_dia=0&modo_noche=1";

// 3. Enviar o respaldar
if (WiFi.status() == WL_CONNECTED) {
    enviarDatosPendientes();
    auto r = postAWebApp(payload);
    if (r.ack) borrarPendientesLittleFS();
    else guardarDatosPendientes(payload);
} else {
    guardarDatosPendientes(payload);
}

// 4. Marcar bandera y entrar en deep sleep
envioNocturnoHecho = true;
unsigned long segundosSleep = timerActual * 60;
unsigned long segundosHastaDia =
    proximoInicioDia.unixtime() - now.unixtime();
if (segundosHastaDia < segundosSleep) segundosSleep = segundosHastaDia;

esp_sleep_enable_timer_wakeup(segundosSleep * 1000000ULL);
configurarDespertarPorBoton();
esp_deep_sleep_start();
}

```

5.4.3. Explicación de la lógica

1. **Lectura de sensores:** se recogen todas las variables (DS18B20, AHT10, BMP280, BH1750, batería y panel).
2. **Construcción del payload:** se arma la cadena de datos, marcando `modo_noche=1`.
3. **Comunicación:**
 - Si hay WiFi → se envían primero respaldos y luego la lectura actual.
 - Si falla → se guarda el paquete en LittleFS.
4. **Deep sleep:** el ESP32 duerme por el intervalo configurado o hasta el inicio del día (lo que ocurra primero).

5. Bandera envioNocturnoHecho: evita duplicar envíos dentro de la misma noche.

5.4.4. Visualización en OLED

Durante la rutina nocturna, el sistema enciende brevemente la pantalla para mostrar:

- Hora de ejecución.
- Estado del envío (OK, fallo, sin WiFi).
- Mensaje: “Modo noche activo”.

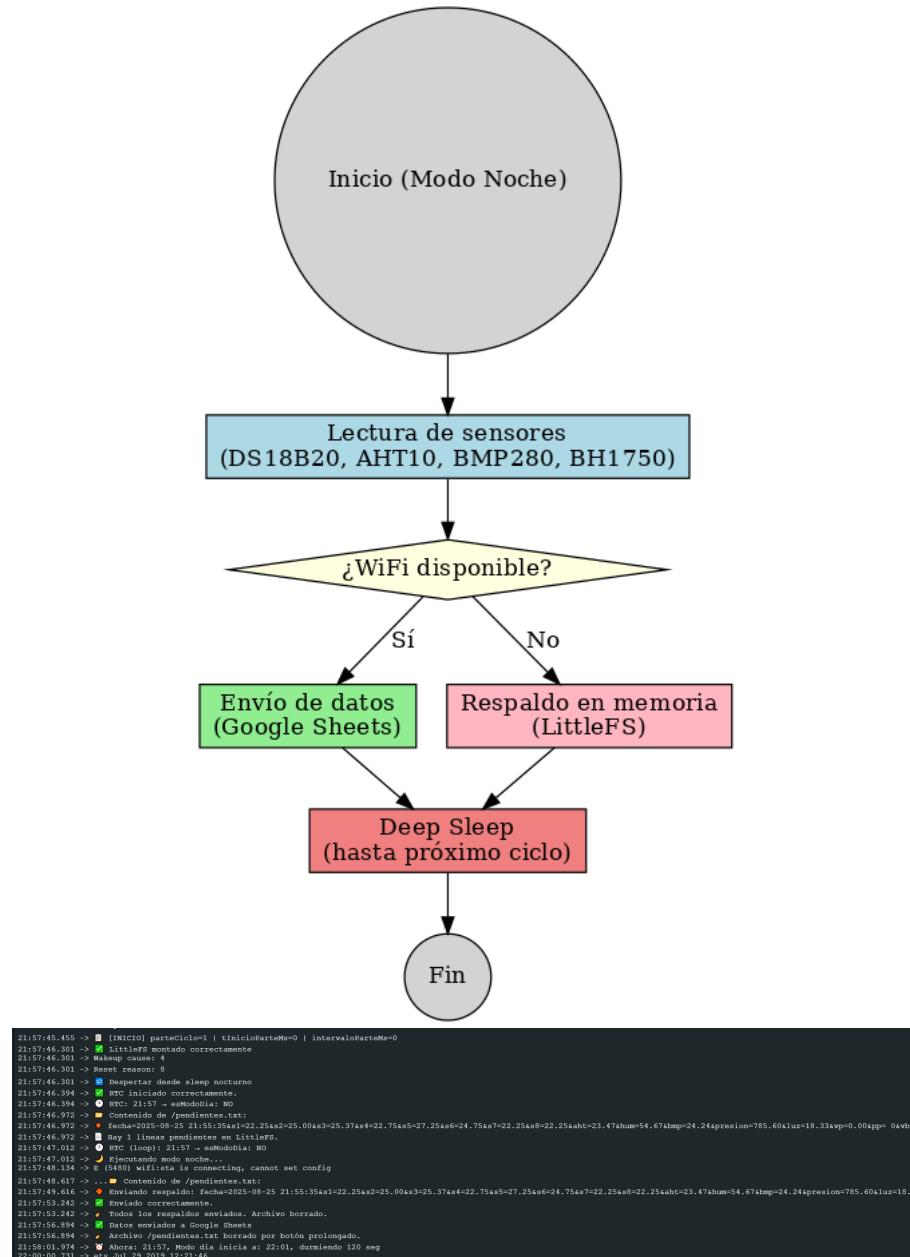


Figura 5.17: Diagrama de flujo del modo noche: Lectura → Envío/Respaldo → Deep Sleep.

5.5. Modo Día y Ciclo de 6 Partes

El modo día está diseñado para operar en ciclos periódicos divididos en seis partes. Esta división permite equilibrar el control energético (relevador y panel) con la adquisición y el envío de datos a la nube.

5.5.1. División en partes

Cada intervalo de muestreo definido por el usuario (`timerActual`, en minutos) se divide en 6 fracciones iguales:

$$\text{intervaloParteMs} = \frac{\text{timerActual} \cdot 60000}{6}$$

Ejemplo: si el usuario selecciona 30 minutos, cada parte dura 5 minutos.

- **Partes 1 a 5:** sólo controlan el relevador (decisión de carga/descarga).
- **Parte 6:** se desactiva el relevador, se leen todos los sensores y se envían los datos al servidor.

5.5.2. Ejecución del ciclo en el `loop()`

```
if (millis() - tInicioParteMs >= intervaloParteMs) {
    Serial.printf(" Parte %d/6 del ciclo (MODO DÍA)\n", parteCiclo);

    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    DateTime nowRTC = rtc.now();
    char fechaHoraRTC[20];
    sprintf(fechaHoraRTC, "%04d-%02d-%02d %02d:%02d:%02d",
            nowRTC.year(), nowRTC.month(), nowRTC.day(),
            nowRTC.hour(), nowRTC.minute(), nowRTC.second());

    if (parteCiclo <= 5) {
        // Partes 1..5: control del relé
        controlarRele(vb, vp);

    } else {
        // Parte 6: apagar relé y enviar datos
        digitalWrite(RELAY_PIN, LOW);
        releActivo = false;
        delay(300);
```

```
sensoresDS18B20.requestTemperatures();
float tempDS[8];
for (int i = 0; i < 8; i++) tempDS[i] = sensoresDS18B20.getTempC(sensores[i]);

sensors_event_t humidity, temp;
aht.getEvent(&humidity, &temp);
float bmpTemp, presion;
leerBMP(bmpTemp, presion);
float luz = leerLuz();

esReintento = hayPendientesLittleFS();

String payload = "s1=" + String(tempDS[0], 2);
for (int i = 1; i < 8; i++)
    payload += "&s" + String(i + 1) + "=" + String(tempDS[i], 2);
payload += "&aht=" + String(temp.temperature, 2);
payload += "&hum=" + String(humidity.relative_humidity, 2);
payload += "&bmp=" + String(bmpTemp, 2);
payload += "&presion=" + String(presion, 2);
payload += "&luz=" + String(luz, 2);
payload += "&vp=" + String(vp, 2);
payload += "&pp=" + String(pp, 0);
payload += "&vb=" + String(vb, 2);
payload += "&pb=" + String(pb, 0);
payload += "&intervalo=" + String(timerActual);
payload += "&modo_dia=1&modo_noche=0";
payload += "&rtc=" + urlEncode(String(fechaHoraRTC));
if (esReintento) payload += "&reintento=1";

if (WiFi.status() == WL_CONNECTED) {
    enviarDatosPendientes();
    auto r = postAWebApp(payload);
    if (r.ack) estadoUltimoEnvio = "Enviado correctamente";
    else { estadoUltimoEnvio = "Fallo en envio";
        guardarDatosPendientes(payload); }
} else {
    estadoUltimoEnvio = "Sin WiFi";
    guardarDatosPendientes(payload);
}

WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
delay(50);
tEnvio = millis();
controlarRele(vb, vp);
}

// Avanzar de parte
```

```

parteCiclo++;
if (parteCiclo > 6) parteCiclo = 1;
tInicioParteMs = millis();
}

```

5.5.3. Explicación de la lógica

1. **Temporización:** cuando `millis() - tInicioParteMs` supera `intervaloParteMs`, finaliza la parte actual y comienza la siguiente.
2. **Partes 1–5:** ejecutan `controlarRele(vb, vp)` para decidir si cargar o no.
3. **Parte 6:**
 - Se apaga el relevador para no interferir con mediciones.
 - Se leen DS18B20, AHT10, BMP280, BH1750, batería y panel.
 - Se arma el `payload` y se envía mediante `postAWebApp()`.
 - Si no hay WiFi, los datos se guardan en LittleFS para reintento posterior.
 - Se actualiza `estadoUltimoEnvio` para la pantalla OLED.
4. **Avance de parte:** `parteCiclo` se incrementa; al superar 6, vuelve a 1 y se reinicia el contador.

5.5.4. Ventanas de visualización OLED

Para ahorrar energía, el OLED no permanece encendido todo el tiempo:

- Se define una ventana del **20 %** al inicio y **20 %** al final de cada parte donde la pantalla se enciende y refresca.
- El resto del tiempo permanece apagada.

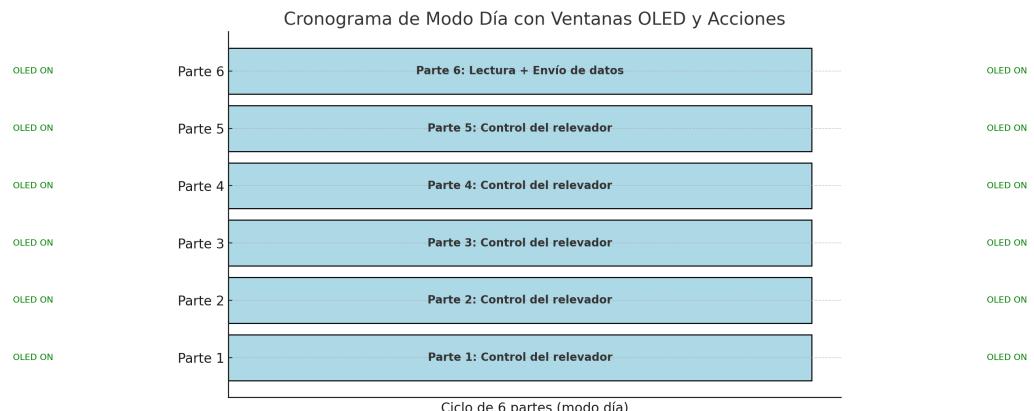


Figura 5.18: Las 6 partes del ciclo diurno, ventanas OLED al inicio y al final, y acción principal de cada parte.

5.6. Manejo del Botón y Encoder

El encoder rotativo con pulsador integrado constituye la interfaz de control local del sistema. Permite:

- Seleccionar el tiempo de muestreo (timerPot).
- Confirmar la selección (timerActual).
- Navegar entre páginas de la OLED.
- Ejecutar acciones especiales (borrar respaldos en memoria).

5.6.1. Lectura del giro del encoder

```
int clkState = digitalRead(CLK);
if (clkState != lastClkState && clkState == LOW) {
    int dtState = digitalRead(DT);
    int direction = (dtState != clkState) ? 1 : -1;

    if (!modoVisualizacion) {
        indexSeleccion += direction;
        indexSeleccion = constrain(indexSeleccion, 0, MAX_INDEX);
        timerPot = calcularTiempoDesdeIndex(indexSeleccion);
        mostrarTiempo();
    } else {
        paginaActual = (paginaActual + direction + 5) % 5;
        mostrarPaginaActual();
    }
    delay(2);
}
lastClkState = clkState;
```

Explicación: Cada flanco de CLK es un paso; DT define dirección. Si no está confirmado el tiempo → ajusta timerPot. Si ya está en modo visualización → cambia página OLED (0–4).

5.6.2. Lectura del botón integrado

```
bool btn = (digitalRead(SW) == LOW);
if (btn && btnPressedAt == 0) btnPressedAt = millis();
if (!btn && btnPressedAt > 0) {
    unsigned long pressTime = millis() - btnPressedAt;

    if (pressTime > 5000) {
        borrarPendientesSolicitado = false;
        borrarPendientesLittleFS();
        mostrarBorradoRespaldo();
        delay(1500);
```

```

} else if (pressTime > 2000) {
    modoVisualizacion = false;
    tiempoConfirmado = false;
    mostrarTiempo();
    estadoUltimoEnvio = "Sin enviar";
} else if (!modoVisualizacion) {
    timerActual = timerPot;
    delay(200);
    lastClkState = digitalRead(CLK);
    tiempoConfirmado = true;
    Serial.println("Boton presionado: tiempo confirmado");

    if (esModoNoche) {
        envioNocturnoHecho = false;
        ejecutarModoNoche();
        return;
    }
    if (esModoDia) {
        intervaloParteMs = (timerActual * 60000UL) / 6;
        tInicioParteMs = millis();
        parteCiclo = 1;
    }
    modoVisualizacion = true;
    paginaActual = 0;
    tEnvio = millis();
    estadoUltimoEnvio = "Sin enviar";
    encenderOLED();
    mostrarPaginaActual();
}
btnPressedAt = 0;
}

```

Explicación según tiempo de pulsación:

- **Tap corto (¡2 s):** confirma el tiempo seleccionado.
 - Copia `timerPot` en `timerActual`.
 - Si es de día → inicia el ciclo dividido en 6 partes.
 - Si es de noche → ejecuta la rutina nocturna y duerme el sistema.
- **Hold medio (2–5 s):** cancela la selección, vuelve a pantalla de tiempo (útil para reconfigurar).
- **Hold largo (¡5 s):** borra los respaldos almacenados en LittleFS (llama `borrarPendientesLittleFS()` y muestra aviso en OLED).

5.6.3. Confirmación del tiempo seleccionado

Una vez confirmado, el sistema distingue entre modo día y modo noche:

- **Día:** arranca `parteCiclo=1`, define `intervaloParteMs`, activa visualización proporcional y control de relevador.
- **Noche:** se prepara para un único envío con `ejecutarModoNoche()` y luego entra en *deep sleep*.

5.6.4. Visualización en OLED

Durante la interacción con el botón/encoder, la OLED muestra:

- Menú de tiempo: minutos u horas seleccionados.
- Instrucciones: “Tap = OK, Hold = Salir”.
- Mensajes de confirmación o borrado: cuando se guardan cambios o se eliminan respaldos.



Figura 5.19: Mensaje de “Respaldo borrado” mostrado en la pantalla OLED tras un hold largo.

5.7. Integración con Google Apps Script y Google Sheets

Este bloque documenta el servicio web en Google Apps Script (GAS) que recibe lecturas del ESP32 por HTTP POST y las persiste en Google Sheets.

5.7.1. Flujo de recepción

1. El ESP32 construye un payload `application/x-www-form-urlencoded`.

2. Envía un POST a la URL del Web App GAS.
3. El servicio `doPost(e)`:
 - Valida parámetros.
 - Construye un arreglo con 22 campos en orden fijo.
 - Inserta una fila en la hoja “4.0”.
 - Responde “OK”.
4. Si no hay red → se guarda en LittleFS y se reintenta después.

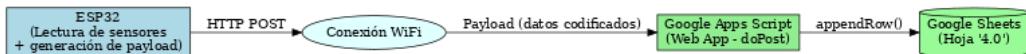


Figura 5.20: Diagrama de flujo ESP32 -> WEB APP (GAS).

5.7.2. 4.11.2 Estructura del `doPost(e)`

1. Selección de hoja y obtención segura de parámetros

Se obtiene la hoja con `getSheetByName("4.0")`.

Se extraen parámetros con manejo defensivo: `p = (e && e.parameter) ? e.parameter : {}` evita fallos cuando no hay cuerpo.

`horaRecepcion = new Date()` registra la hora del servidor (carátula de recepción).

2. Manejo de tiempos

`horaRTC`: marca temporal enviada por el ESP32 (RTC propio).

`fechaTexto`: si el paquete proviene de respaldo local (LittleFS) puede venir como `fecha=`; si no, se usa `rtc`.

Esto permite distinguir hora de recepción (columna A) de hora/fecha de medición (columna B).

3. Banderas y estado del envío

`modo_dia` y `modo_noche` llegan como "1/"0z se traducen a una etiqueta de estado: “Día” o “Noche”.

Si llega `reintento=1` o se detecta `origen=backup/fecha=`, se agrega “Reintento”.

El resultado se almacena en la columna C como `estadoEnvio`.

4. Construcción de la fila (orden fijo)

Se conservó el mismo orden que usa el firmware para que el *parsing* sea consistente:

Col	Campo	Descripción
A	horaRecepcion	Hora del servidor (GAS) al recibir el POST.
B	fechaTexto	Fecha/hora de medición (RTC ESP32 o fecha= de respaldo).
C	estadoEnvio	Etiquetas: “ Día” / “ Noche” y “ Reintento” si aplica.
D	s1	DS18B20 – S1.
E	s2	DS18B20 – S2.
F	s3	DS18B20 – S3.
G	s4	DS18B20 – S4.
H	s5	DS18B20 – S5.
I	s6	DS18B20 – S6.
J	s7	DS18B20 – S7.
K	s8	DS18B20 – S8.
L	aht	Temperatura AHTX0 (°C).
M	hum	Humedad relativa AHTX0 (%).
N	bmp	Temperatura BMP280 (°C).
O	presion	Presión BMP280 (hPa).
P	luz	Iluminancia BH1750 (lx).
Q	vp	Voltaje panel (V).
R	pp	% relativo panel (estimado por firmware).
S	vb	Voltaje batería (V).
T	pb	% batería (estimado por firmware).
U	intervalo	timerActual (min).
V	timer_pot	timerPot (seleccionado en el encoder).

Tabla 5.2

Nota: Si también se envía `evento=inicio_dia` en lecturas instantáneas, puede agregarse una columna adicional a futuro para registrar eventos especiales. Por ahora se conserva el formato A-V

Figura 5.21: Hoja "4.0" mostrando encabezados A-V

5.7.3. 4.11.3 Respuesta del servicio

Retorna texto plano ".K", con `ContentService.MimeType.TEXT`.

El firmware trata como correcto cualquier 2xx/3xx o la presencia de ".K." en cuerpo, y en ciertos timeouts controlados.

5.7.4. 4.11.4 Despliegue del Web App

Pasos de despliegue recomendados:

1. Abrir el editor de Apps Script (vinculado a la hoja).
2. Crear el archivo con la función `doPost(e)`.
3. Proyecto → Implementar → Nueva implementación → Tipo: Aplicación web.
4. Establecer:
 - “Ejecutar como”: Tú (propietario).
 - “Quién tiene acceso”: Cualquiera con el enlace (requerido para recibir desde ESP32).
5. Copiar la URL de la implementación y actualizarla en el firmware (`WEB_APP_URL`).



Figura 5.22: Implementación del Web App.

Observación: si se vuelve a implementar una versión nueva, la URL puede cambiar. Verificar y actualizar en el firmware.

5.7.5. 4.11.5 Seguridad y buenas prácticas

- Ámbito mínimo: exponer el Web App solo con POST, y procesar únicamente los parámetros esperados.
- Rate limiting en firmware: el código ya espacia envíos (helper `postAWebApp` con espera mínima entre POST) para no saturar el endpoint.
- Validaciones defensivas: el GAS evita fallos si faltan parámetros, y conserva coherencia de columnas.

- Control de acceso básico: aunque el Web App esté abierto, la estructura y nombres de parámetros ofician de “contrato” mínimo. Si se requiere, pueden añadirse:
 - Token simple (p. ej., `key=...`) que se valida en `doPost`.
 - Filtro por IP (no nativo en GAS; se podría inferir desde cabeceras si aplica).
- Cuotas GAS/Sheets: el diseño inserta una fila por POST. Los tiempos de envío desde firmware están distribuidos para no acercarse a límites.

5.7.6. 4.11.6 Pruebas de integración

Para validar el flujo sin el ESP32 se puede usar un cURL de ejemplo (solo ilustrativo; sustituir URL y valores):

```
curl -X POST "https://script.google.com/macros/s/WEB_APP_ID/exec" \
-H "Content-Type: application/x-www-form-urlencoded" \
--data "rtc=2025-08-20+12:34:56&modo_dia=1&modo_noche=0&s1=21.5&s2=21.7&s3=21.6&s4=21.9&s5=21.8"
```

Resultados esperados:

- Respuesta OK.
- Nueva fila al final de “4.0” con columnas A–V completas.

5.7.7. 4.11.7 Manejo de registros provenientes de respaldo (LittleFS)

- Si el firmware detecta envío diferido, adjunta `origen=backup` y/o `fecha=` con el sello de tiempo del RTC al momento del respaldo.
- El GAS interpreta esos campos y marca el estado en C con “Reintento”.
- El orden de columnas no cambia, garantizando trazabilidad entre lecturas en tiempo real y reenvíos.

5.7.8. 4.11.8 Buenas prácticas de hoja de cálculo

- Encabezados fijos (A–V).
- Formato de fecha/hora en A y B:
 - A: Fecha y hora (hora de recepción).
 - B: Fecha y hora (hora RTC o respaldo).
- Protección opcional de encabezados/hoja para evitar movimientos accidentales.
- Filtros y vistas para análisis (por modo, por reintento, por rango de fechas).
- Gráficos vinculados (luz vs. hora, voltajes vs. hora, temperatura por sensor, etc.).

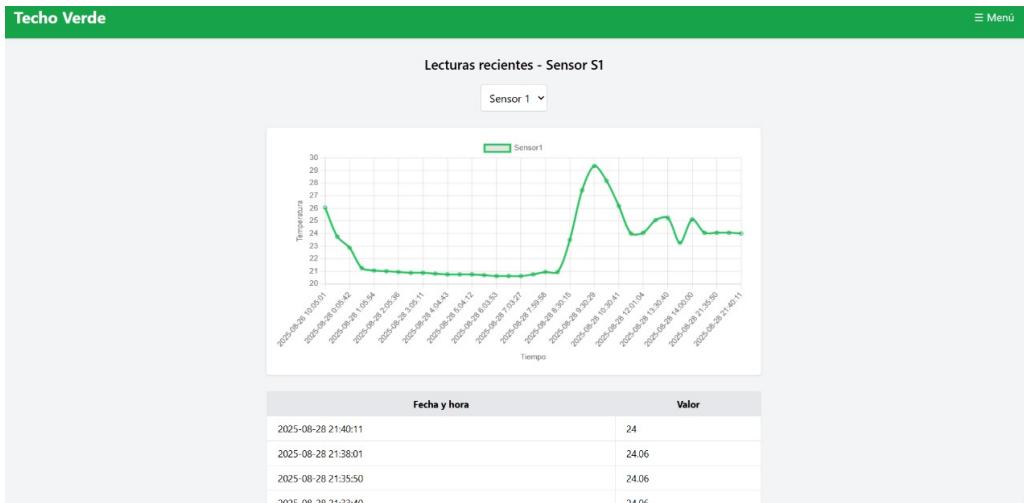


Figura 5.23: Captura de filtros y gráficos propuestos.

5.7.9. 4.11.9 Diagnóstico de fallos comunes

Síntoma	Possible causa	Acción sugerida
OK no aparece y no hay fila nueva	URL incorrecta / versión no implementada	Verificar Implementación y WEB_APP_URL.
Columnas corridas o vacías	Parámetros faltantes o con nombre errado	Revisar payload en consola del ESP32.
Muchos Reintento consecutivos	Fallas de WiFi o latencia alta	Revisar señal, AP, y tiempos de POST.
Códigos -11 u otros timeouts en firmware	Respuesta tardía de GAS	El helper ya trata -11 como “ACK probable”.
Doble inserción de una lectura	Reintento tardío + POST original sí procesado	Mantener política actual (tolerar duplicados) o agregar token idempotente si se requiere.

5.7.10. 4.11.10 Posibles mejoras futuras

- Token HMAC simple para firmar payloads (validación en GAS).
- Columna “evento” (p. ej., inicio_dia, inspeccion, etc.).
- Hoja de “logs” paralela para registrar códigos HTTP y métricas de latencia.
- Cuotas y alertas: script que avise si se detectan >N reintentos por día.

5.8. 4.12 Interfaz OLED y Navegación por Páginas

El sistema cuenta con una pantalla OLED de 128×64 px (controlador SSD1306) que constituye la interfaz visual principal. Su uso se optimiza para reducir el consumo de energía, encendiéndose solo

en momentos clave (ventanas de inicio/fin de parte, confirmación de tiempo, mensajes breves).

5.8.1. 4.12.1 Inicialización y control de energía

```
void encenderOLED() {
    if (!displayInicializado) {
        if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
            Serial.println(" OLED no respondio en 0x3C");
            return;
        }
        displayInicializado = true;
        display.clearDisplay();
        display.display();
    } else {
        display.ssdi306_command(SSD1306_DISPLAYON);
    }
    oledOn = true;
}

void apagarOLED() {
    if (!displayInicializado) return;
    display.ssdi306_command(SSD1306_DISPLAYOFF);
    oledOn = false;
}
```

Puntos clave:

- Se inicializa solo una vez y se reutiliza en ciclos posteriores.
- La función `apagarOLED()` se llama al salir de ventanas de visualización o al entrar en modo noche.
- Esto evita consumo innecesario de la pantalla (20 mA).

5.8.2. 4.12.2 Navegación por páginas

El encoder permite moverse entre 5 páginas diferentes en modo visualización:

```
void mostrarPaginaActual() {
    switch (paginaActual) {
        case 0: mostrarPaginaSensoresDS(0); break;
        case 1: mostrarPaginaSensoresDS(4); break;
        case 2: mostrarPaginaAmbientales(); break;
        case 3: mostrarPaginaVoltajes(); break;
        case 4: mostrarPaginaEnvio(); break;
    }
}
```

Resumen de páginas:

- Página 0: Sensores DS18B20 (1-4).
- Página 1: Sensores DS18B20 (5-8).
- Página 2: Sensores ambientales (AHT10, BMP280, BH1750).
- Página 3: Voltajes (batería, panel, estado del relevador).
- Página 4: Estado de envío y parte actual del ciclo.

5.8.3. 4.12.3 Página de selección de tiempo

Cuando aún no se confirma el intervalo, la pantalla muestra el menú de selección:

```
void mostrarTiempo() {
    encenderOLED();
    display.clearDisplay();
    display.setTextSize(2);
    display.setCursor(0, 0);
    display.print("Tiempo:");
    display.setCursor(0, 20);
    if (timerPot < 60) {
        display.print(timerPot);
        display.print(" min");
    } else {
        display.print(timerPot / 60.0, 1);
        display.print(" h");
    }
    display.setTextSize(1);
    display.setCursor(0, 50);
    display.print("Tap=OK Hold=Salir");
    display.display();
}
```

Uso:

- Encoder → ajusta minutos/horas.
- Tap corto → confirma y arranca ciclo.
- Hold medio → cancela.

5.8.4. 4.12.4 Página de sensores DS18B20

```
void mostrarPaginaSensoresDS(int inicio) {
    encenderOLED();
    sensoresDS18B20.requestTemperatures();
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println(inicio == 0 ? "Sensores izquierda" : "Sensores derecha");
    for (int i = inicio; i < inicio + 4; i++) {
```

```

        float temp = sensoresDS18B20.getTempC(sensores[i]);
        display.printf("%s: %.1f C\n", nombresSensores[i], temp);
    }
    display.display();
}

```

Muestra: 4 sensores por página, con nombre y temperatura (°C).

- Página 0 → Sensores 1–4.
- Página 1 → Sensores 5–8.

5.8.5. 4.12.5 Página de sensores ambientales

```

void mostrarPaginaAmbientales() {
    encenderOLED();
    sensors_event_t humidity, temp;
    aht.getEvent(&humidity, &temp);
    float luz = leerLuz();
    float bmpTemp, presion;
    leerBMP(bmpTemp, presion);

    display.clearDisplay();
    display.setTextSize(1);
    display.printf("AHT: %.1f C\n", temp.temperature);
    display.printf("Hum: %.1f %%\n", humidity.relative_humidity);
    display.printf("BMP: %.1f C\n", bmpTemp);
    display.printf("Pres: %.1f hPa\n", presion);
    display.printf("Luz: %.1f lx\n", luz);
    display.display();
}

```

Muestra: Temperatura/humedad (AHT10), temperatura/presión (BMP280) e iluminancia (BH1750).

5.8.6. 4.12.6 Página de voltajes y relevador

```

void mostrarPaginaVoltajes() {
    encenderOLED();
    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    display.clearDisplay();
    display.setTextSize(1);
    display.printf("Bat: %.2f V (%.0f%%)\n", vb, pb);
    display.printf("Panel: %.2f V (%.0f%%)\n", vp, pp);
    display.printf("Rele: %s\n", releActivo ? "ON" : "OFF");
}

```

```
    display.display();
}
```

Muestra:

- Voltaje de batería y % estimado.
- Voltaje de panel y % estimado.
- Estado del relevador (ON/OFF).

5.8.7. 4.12.7 Página de envío

```
void mostrarPaginaEnvio() {
    encenderOLED();
    DateTime now = rtc.now() + TimeSpan(0, 0, 3, 0);
    display.clearDisplay();
    display.setTextSize(1);
    display.printf("%02d:%02d:%02d %02d/%02d/%04d\n",
                  now.hour(), now.minute(), now.second(),
                  now.day(), now.month(), now.year());
    display.printf("Tiempo sel.: %d min\n", timerActual);
    display.printf("Ult. envío: %lds\n", (millis() - tEnvio) / 1000);
    display.printf("Parte %d/6\n", parteCiclo);
    display.println(estadoUltimoEnvio);
    display.display();
}
```

Muestra:

- Fecha y hora del RTC.
- Intervalo confirmado.
- Tiempo desde el último envío.
- Parte actual del ciclo.
- Estado del último envío (correcto, fallo, sin WiFi).

5.8.8. 4.12.8 Mensajes especiales

Además de las páginas, la OLED se usa para:

- Mostrar borrado de respaldos.
- Resumen final antes de entrar en modo noche.
- Mensajes de arranque (WiFi, inicialización).

5.9. 4.13 Manejo del RTC y Control Horario

El sistema emplea un RTC DS1307 como referencia de tiempo confiable, independiente de reinicios o ciclos de deep sleep. Este componente asegura que las mediciones y transiciones entre modos (día/noche) se realicen de manera precisa y consistente.

5.9.1. 4.13.1 Inicialización del RTC

```

if (!rtc.begin()) {
    Serial.println(" No se encontró el RTC");
} else if (!rtc.isrunning()) {
    Serial.println(" RTC no está corriendo, se ajustará a hora de compilación.");
    rtc.adjust(DateTime(2025, 7, 24, 9, 59, 30));
} else {
    Serial.println(" RTC iniciado correctamente.");
}

```

Explicación:

- Si el RTC no responde → se reporta error.
- Si está detenido → se ajusta a la hora de compilación como valor de referencia.
- Si funciona → se continúa con la hora real en curso.

5.9.2. 4.13.2 Definición de intervalos de día y noche

```

const int HORA_INICIO_DIA = 8;
const int MINUTO_INICIO_DIA = 0;

const int HORA_FIN_DIA = 16;
const int MINUTO_FIN_DIA = 0;

DateTime now = rtc.now();
unsigned long ahoraUnix = now.unixtime();

DateTime inicioDia(now.year(), now.month(), now.day(),
                   HORA_INICIO_DIA, MINUTO_INICIO_DIA, 0);
DateTime finDia(now.year(), now.month(), now.day(),
                HORA_FIN_DIA, MINUTO_FIN_DIA, 0);

esModoDia = (ahoraUnix >= inicioDia.unixtime() && (ahoraUnix < finDia.unixtime()));
esModoNoche = !esModoDia;

```

Explicación:

- Se calculan los límites en formato Unix timestamp.
- Si la hora actual está dentro de la ventana → esModoDia = true.
- En caso contrario → esModoNoche = true.

5.9.3. 4.13.3 Transiciones entre modos

```

// Día → Noche
if (!estabaEnModoNoche && esModoNoche) {
    Serial.println(" Transición Día→Noche");
}

```

```

envioNocturnoHecho = false;
modoVisualizacion = false;
paginaActual = 0;
apagarOLED();
}

// Noche → Día
if (estabaEnModoNoche && esModoDia) {
    Serial.println(" Cambio de Modo Noche a Día");
    if (tiempoConfirmado) {
        enviarInstantaneo("inicio_dia");
    }
    parteCiclo = 1;
    intervaloParteMs = (timerActual * 60000UL) / 6;
    tInicioParteMs = millis();
    modoVisualizacion = true;
    paginaActual = 0;
    timerPot = timerActual;
    encenderOLED();
    mostrarPaginaActual();
}

```

Explicación:

- Día → Noche: se apaga la OLED, se resetea estado de envío nocturno, y se prepara para ejecutarModoNoche().
- Noche → Día: se realiza un envío instantáneo como marcador, se reinicia el ciclo de 6 partes y se reactiva la OLED.

5.9.4. 4.13.4 Marca temporal en los registros

```

DateTime nowRTC = rtc.now();
char fechaHoraRTC[20];
sprintf(fechaHoraRTC, "%04d-%02d-%02d %02d:%02d:%02d",
        nowRTC.year(), nowRTC.month(), nowRTC.day(),
        nowRTC.hour(), nowRTC.minute(), nowRTC.second());

payload += "&rtc=" + urlEncode(String(fechaHoraRTC));

```

Beneficio:

- La columna A en Google Sheets muestra la hora de recepción en el servidor.
- La columna B muestra la hora RTC del dispositivo.
- Esto permite identificar diferencias entre mediciones en tiempo real y reenvíos diferidos desde respaldo.

5.9.5. 4.13.5 Ejemplo de cronograma horario

Hora	Estado del sistema	Acción principal
07:59	Noche	Deep sleep, sin actividad.
08:00	Cambio a Día	Inicio de ciclo 1/6.
12:00	Día	Ciclo en curso, parte 3/6.
15:55	Día	Último ciclo antes de cerrar.
16:00	Cambio a Noche	Lectura única, deep sleep.

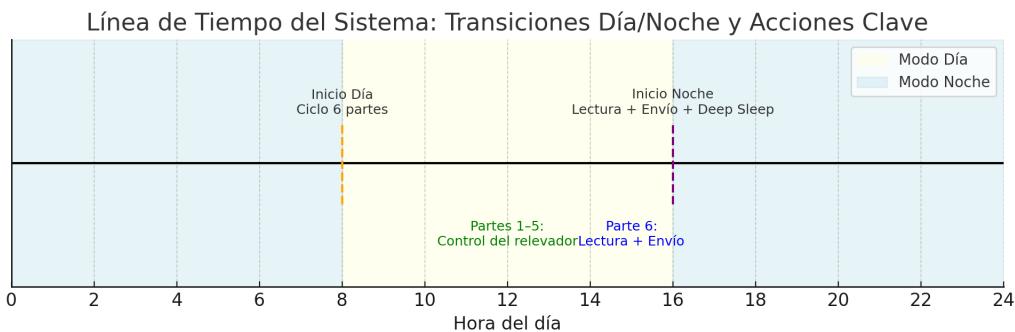


Figura 5.24: Línea de tiempo del sistema. Transiciones día-noche y acciones clave.

5.10. 4.14 Control del Relevador

El relevador implementado en el sistema actúa como un interruptor controlado electrónicamente. Su función principal es regular el uso de energía dependiendo del nivel de batería, la disponibilidad de energía solar y el horario de operación (día o noche).

5.10.1. 4.14.1 Configuración básica

```
#define RELAY_PIN 12
#define UMBRAL_BATERIA 3.6 // Voltaje mínimo deseado (V)
#define UMBRAL_PANEL 3.0 // Voltaje mínimo del panel para carga (V)
bool releActivo = false;
```

Explicación:

- `RELAY_PIN` → GPIO12 conectado al módulo de relevador.
- `UMBRAL_BATERIA` → si la batería cae por debajo de 3.6 V, se considera “baja”.
- `UMBRAL_PANEL` → el panel debe entregar al menos 3.0 V para considerarse “apto para carga”.
- `releActivo` → bandera lógica que refleja si el relevador está activado (ON) o no (OFF).

5.10.2. 4.14.2 Lógica de decisión

La activación del relevador depende de tres condiciones:

1. Que el sistema se encuentre en modo día.
2. Que el voltaje de batería sea menor al umbral configurado.
3. Que el voltaje del panel sea mayor al umbral de carga.

```
void controlarRele(float vb, float vp) {
    bool dentroHorario = esModoDia;
    bool bateriaBaja = vb < UMBRAL_BATERIA;
    bool panelSuficiente = vp > UMBRAL_PANEL;

    Serial.println(" Verificando condiciones del relé:");
    Serial.printf("Batería: %.2f V | Panel: %.2f V\n", vb, vp);
    Serial.printf("Horario OK: %s\n", dentroHorario ? "" : "");
    Serial.printf("Batería baja: %s (<%.2f V)\n", bateriaBaja ? "" : "", UMBRAL_BATERIA);
    Serial.printf("Panel suficiente: %s (>%.2f V)\n", panelSuficiente ? "" : "", UMBRAL_PANEL);

    if (dentroHorario && bateriaBaja && panelSuficiente) {
        digitalWrite(RELAY_PIN, HIGH);
        releActivo = true;
        Serial.println(" Relevador ACTIVADO");
    } else {
        digitalWrite(RELAY_PIN, LOW);
        releActivo = false;
        Serial.println(" Relevador DESACTIVADO");
    }
}
```

Interpretación:

- **Activación (ON):** $esModoDia \wedge (V_b < 3,6V) \wedge (V_p > 3,0V)$.
- **Desactivación (OFF):** si alguna de las condiciones no se cumple, el relevador se apaga.

5.10.3. 4.14.3 Interacción con el ciclo de 6 partes

- Partes 1–5 (día): la función principal del ciclo es decidir el estado del relevador.
- Parte 6: el relevador se apaga para realizar las lecturas de sensores y transmitir datos de manera estable (sin ruido eléctrico asociado al switching).

Esto asegura que las mediciones no se vean afectadas y que la energía se use solo cuando realmente es necesario.

5.10.4. 4.14.4 Visualización en la OLED

En la página de voltajes se incluye el estado del relevador:

Bat: 3.92 V (85%)
 Panel: 4.21 V (95%)
 Rele: ON

De esta forma, el operador puede verificar directamente si el sistema está cargando la batería o si permanece en reposo.

5.10.5. 4.14.5 Ventajas del diseño

- Protección de la batería: evita descargas profundas al cortar consumo cuando el nivel es bajo.
- Optimización de la carga: se activa solo cuando el panel puede aportar energía suficiente.
- Ahorro energético: en modo noche, el relevador permanece siempre apagado.

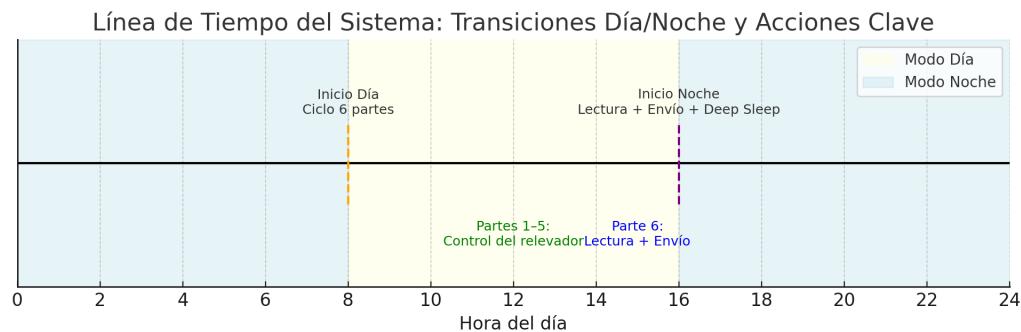


Figura 5.25: Diagrama de decisión. Tipo Árbol

5.11. 4.15 Conclusiones del Capítulo de Programación

La integración del firmware desarrollado para el ESP32 junto con el script en Google Apps Script ha permitido implementar un sistema de monitoreo y control ambiental robusto, confiable y energéticamente eficiente.

Conclusiones del Capítulo

1. Modularidad y estructura

El código fue organizado en bloques funcionales claramente diferenciados:

- Lectura de sensores de suelo y ambientales.
- Control de visualización mediante OLED.
- Manejo de horarios y transición Día/Noche con RTC.
- Control de relevador bajo criterios de voltaje.

- Comunicación y respaldo de datos con Google Sheets.

Esta división modular facilita la depuración y la escalabilidad del sistema.

2. Manejo eficiente de la energía

- Modo Día: división en 6 partes que equilibran monitoreo constante con ahorro energético.
- Modo Noche: un único envío y posterior deep sleep, reduciendo consumo al mínimo.
- OLED y WiFi: encendidos solo en ventanas específicas o cuando se requiere.

Esto asegura autonomía prolongada en aplicaciones remotas.

3. Respaldo y confiabilidad en la comunicación

El uso de LittleFS como almacenamiento de respaldo garantiza que ningún dato se pierda.

- Si hay WiFi → envío directo a Google Sheets.
- Si falla → el payload se conserva y se reenvía automáticamente en la siguiente conexión.

Se logra así un sistema tolerante a fallos de red.

4. Interfaz de usuario intuitiva

El encoder con pulsador y la OLED permiten al usuario:

- Seleccionar y confirmar intervalos de muestreo.
- Consultar temperaturas, humedad, presión, lux y voltajes en tiempo real.
- Verificar estado del relevador y de los envíos.
- Eliminar respaldos manualmente en caso necesario.

La interfaz es simple pero funcional, eliminando la necesidad de periféricos externos.

5. Integración con la nube

El Google Apps Script recibe los datos y los organiza en Google Sheets, con columnas dedicadas a:

- Hora de recepción (servidor).
- Hora de medición (RTC).
- Estado de envío (día, noche, reintento).
- Lecturas de sensores y variables eléctricas.

Esto proporciona un histórico centralizado y accesible para análisis, gráficas y reportes.

6. Logro global del sistema

En conjunto, el capítulo demuestra que el sistema:

- Opera de manera autónoma sin intervención constante.
- Garantiza confiabilidad en la adquisición de datos.
- Optimiza el uso de energía solar y de batería.
- Facilita la integración con plataformas de análisis mediante Google Sheets.

Conclusión final del capítulo

El desarrollo del firmware y su integración con el servicio web constituyen el núcleo funcional del proyecto, uniendo la capa física (sensores, relevador, batería) con la capa digital (procesamiento, respaldo y transmisión).

Con ello, se establece una base sólida para el análisis matemático, la implementación de estrategias de control más avanzadas y la futura integración con interfaces gráficas o aplicaciones móviles.

Cronograma seguido para TTII

En la tabla 5.3 se presenta el cronograma de actividades seguido para desarrollar adecuadamente los objetivos específicos para Trabajo Terminal 2 se muestra a continuación.

Tabla 5.3: Cronograma de actividades para TTII

2025/2 (Semanas)								
10	11	12	13	14	15	16	17	18
Implementación del sistema								
Montaje electrónico en protoboard								
Programación: máquina de estados								
Implementación de doble alimentación								
Codificación de almacenamiento local								
Despliegue de interfaz web								
Pruebas de campo (3 días)								
Evaluación de resultados								
Análisis de datos y gráficos								
Redacción de resultados y discusión								
Conclusiones y entrega final TT2								

CAPÍTULO 6

Conclusiones Finales

El sistema desarrollado demostró que un microcontrolador ESP32 puede integrar de manera efectiva la adquisición de datos ambientales, la gestión energética y la transmisión hacia la nube en un mismo dispositivo, garantizando confiabilidad y eficiencia. El diseño modular del código, estructurado en librerías, funciones auxiliares, controladores y modos de operación, facilitó la depuración, el mantenimiento y la escalabilidad futura del sistema. La implementación de un mecanismo de respaldo en memoria interna mediante LittleFS, junto con el reenvío automático de datos pendientes, aseguró la continuidad de la información aun en condiciones de conectividad inestable.

En términos de consumo energético, la diferenciación entre modo día y modo noche representó una estrategia eficaz para prolongar la autonomía del sistema sin sacrificar la calidad de los registros. Durante el modo día, el ciclo dividido en seis partes permitió alternar entre el control del relevador y el envío de datos, mientras que en el modo noche se ejecutó un único envío seguido de un periodo de deep sleep, optimizando la operación en horarios sin irradiancia solar. Además, la interfaz de usuario basada en un encoder rotativo y una pantalla OLED proporcionó una solución práctica y accesible para la configuración de intervalos y la supervisión de estados sin necesidad de modificar el firmware.

Más allá de los logros técnicos alcanzados, el trabajo abre la puerta a futuras ampliaciones y aplicaciones. La lógica implementada puede adaptarse para incluir nuevos sensores ambientales, como medición de calidad del aire, niveles de CO o humedad del suelo. La información recolectada puede servir como base para análisis predictivos, integración de algoritmos de inteligencia artificial y sistemas de gestión ambiental en tiempo real. Asimismo, el sistema es escalable hacia arquitecturas IoT más complejas mediante protocolos como MQTT o plataformas en la nube con dashboards interactivos. Finalmente, la gestión energética aplicada establece un camino para optimizar aún más el uso de fuentes renovables, posibilitando la implementación del sistema en escenarios remotos con altos requerimientos de autonomía.

En conclusión, este proyecto constituye no solo una solución funcional de monitoreo ambiental, sino también un prototipo flexible y adaptable que puede evolucionar hacia aplicaciones de mayor impacto, contribuyendo al desarrollo de herramientas tecnológicas orientadas a la sustentabilidad y la gestión eficiente de recursos.

6.1. Trabajo Futuro

El sistema desarrollado constituye una base sólida para el monitoreo ambiental autónomo, pero existen diversas áreas de mejora y expansión que podrían abordarse en etapas posteriores:

1. Ampliación de variables medidas

Integrar nuevos sensores para registrar parámetros adicionales como concentración de CO, material particulado, humedad de suelo o radiación UV, ampliando el alcance de las aplicaciones en agricultura, calidad del aire y control ambiental urbano.

2. Optimización energética avanzada

Incorporar algoritmos dinámicos de gestión de energía que ajusten automáticamente la frecuencia de muestreo y transmisión en función del nivel de batería y la disponibilidad solar, maximizando la autonomía en entornos críticos.

3. Mejoras en la comunicación

Explorar protocolos alternativos como MQTT o LoRaWAN para aplicaciones de largo alcance o entornos con conectividad WiFi limitada, garantizando mayor versatilidad en el despliegue del sistema.

4. Seguridad en la transmisión de datos

Implementar autenticación y cifrado de payloads mediante llaves HMAC o certificados, reforzando la integridad y privacidad de la información enviada a la nube.

5. Interfaz de usuario ampliada

Desarrollar una aplicación móvil o dashboard web que permita la consulta de datos en tiempo real, la configuración remota de parámetros y la visualización de gráficas históricas, facilitando el uso del sistema a usuarios no técnicos.

6. Análisis predictivo y control inteligente

Aplicar técnicas de inteligencia artificial y modelos matemáticos sobre los datos recolectados para anticipar condiciones críticas, optimizar la activación del relevador y proponer estrategias de gestión ambiental basadas en predicciones.

7. Mejoras en la estructura física y electrónica

Diseñar una versión optimizada de la tarjeta lógica con mejoras electrónicas en estabilidad, protección contra sobrecargas y capacidad de expansión, que incremente la robustez del sistema en condiciones de campo.

8. Integración de un sistema de energía solar dedicado

Incorporar un módulo fotovoltaico independiente con regulador y almacenamiento dedicado, garantizando una operación completamente autónoma y sostenible en ubicaciones sin acceso a la red eléctrica.

Apéndices

Apéndice Código de la lógica ESP32 FireBeetle

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_AHTX0.h>
#include <Adafruit_BMP280.h>
#include <BH1750.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <LittleFS.h>
#include <RTClib.h>
#include <WiFiClientSecure.h>

// Credenciales
const char* ssid = "cheki";
const char* password = "123456789";

//URL actual del Google Apps Script
const char* WEB_APP_URL = "https://script.google.com/macros/s/AKfycbwtisoZ792VxR0pPR4702bbWtP9KmF"

struct PostResult {
    int http;
    bool ack;
    String body;
};

PostResult postAWebApp(const String& payload) {
    static unsigned long lastPostMs = 0;
    unsigned long nowMs = millis();
    if (nowMs - lastPostMs < 1500) delay(1500 - (nowMs - lastPostMs));
    lastPostMs = millis();

    WiFiClientSecure client;
    client.setInsecure();

    HTTPClient http;
    http.setTimeout(12000);
    http.setFollowRedirects(HTTPC_DISABLE_FOLLOW_REDIRECTS); // clave
    http.setReuse(false);
    // (opcional) http.setUserAgent("ESP32");

    if (!http.begin(client, WEB_APP_URL)) {
        return { -90, false, "" };
    }
}
```

```

http.addHeader("Content-Type", "application/x-www-form-urlencoded");

int code = http.POST(payload);
String body = http.getString();
http.end();

// Reglas de ACK:
// - 2xx → OK (respuesta \normal")
// - 3xx → OK (Apps Script suele redirigir tras procesar)
// - -11 → OK (timeout de lectura, el server casi siempre ya procesó)
// - "OK" → OK (por si tu doPost devuelve \OK" en texto)
bool ok = (code >= 200 && code < 300) || (code >= 300 && code < 400);
if (!ok) {
    String upper = body;
    upper.toUpperCase();
    if (upper.indexOf("OK") >= 0) ok = true;
    if (code == -11) ok = true;
}

return { code, ok, body };
}

// -- URL encoder para application/x-www-form-urlencoded --
String urlEncode(const String& s) {
    String out;
    out.reserve(s.length() * 3);
    const char* hex = "0123456789ABCDEF";
    for (size_t i = 0; i < s.length(); ++i) {
        char c = s[i];
        if ((‘a’ <= c && c <= ‘z’) || (‘A’ <= c && c <= ‘Z’) || (‘0’ <= c && c <= ‘9’) || c == ‘ ‘)
            out += c;
        else if (c == ‘+’) out += ‘+’;
        else {
            out += ‘%’;
            out += hex[(c >> 4) & 0xF];
            out += hex[c & 0xF];
        }
    }
    return out;
}

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

```

```
// Pines del encoder
#define CLK 16
#define DT 17
#define SW 4

// Pines de voltaje
#define PIN_PANEL 34
#define PIN_BATERIA 35

// DS18B20
#define ONE_WIRE_BUS 25
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensoresDS18B20(&oneWire);

#define RELAY_PIN 12
#define UMBRAL_BATERIA 3.6 // Voltaje mínimo deseado
#define UMBRAL_PANEL 3.0 // Voltaje mínimo del panel para cargar
bool releActivo = false;

// Sensores ambientales
Adafruit_AHTX0 aht;
Adafruit_BMP280 bmp;
BH1750 lightMeter;

int indexSeleccion = 0;
const int MAX_INDEX = 105;
bool modoVisualizacion = false;
unsigned long btnPressedAt = 0;
int paginaActual = 0;
String estadoUltimoEnvio = "Sin enviar";
bool esModoNoche = false;
bool esModoDia = false;
int lastClkState = HIGH;
unsigned long tEnvio = 0;
RTC_DS1307 rtc;
RTC_DATA_ATTR bool tiempoConfirmado = false;
RTC_DATA_ATTR int timerActual = 1;
RTC_DATA_ATTR int timerPot = 1;
RTC_DATA_ATTR bool estabaEnModoNoche = false;
RTC_DATA_ATTR bool envioNocturnoHecho = false;
bool displayInicializado = false;
unsigned long tInicioParteMs = 0;
unsigned long intervaloParteMs = 0;
bool esReintento = false;
// === Definiciones reales de las globales (persisten entre deep-sleeps) ===
RTC_DATA_ATTR int parteCiclo = 1; // contador 1..6
bool borrarPendientesSolicitado = false; // flag para borrar respaldos
bool oledOn = false;
```

```

const int HORA_INICIO_DIA = 20;
const int MINUTO_INICIO_DIA = 45;

const int HORA_FIN_DIA = 20;
const int MINUTO_FIN_DIA = 55;

DeviceAddress sensores[8] = {
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x66, 0x24, 0xC5 }, // S1 - Lagrima de María
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x65, 0x5C, 0xAA }, // S2 - Dedo de niño
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x6F, 0xAF, 0xDB }, // S3 - Madre perla
    { 0x28, 0xFF, 0x64, 0x1E, 0xC3, 0x93, 0x89, 0x5A }, // S4 - Sedo dorado
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x7F, 0x5E, 0x1D }, // S5 - Echeveria obscura
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x6F, 0xA7, 0x19 }, // S6 - Dedos de Dios
    { 0x28, 0xFF, 0x64, 0x1E, 0xC3, 0x90, 0xE2, 0x4A }, // S7 - Rosa de alabastro
    { 0x28, 0xFF, 0x64, 0x1E, 0xC2, 0x65, 0x65, 0x88 } // S8 - Sin vegetación
};

const char* nombresSensores[8] = {
    "Lagrima Maria",           // 1
    "Dedo de nino",            // 2
    "Madre perla",             // 3
    "Sedo dorado",              // 4
    "Echeveria obs",            // 5
    "Dedos de Dios",             // 6
    "Rosa de \nalabastro",      // 7
    "Vacio"                    // 8
};

// --- forward-decls de globales usadas antes de definirse ---
extern int parteCiclo;
extern bool borrarPendientesSolicitado;

// ===== Prototipos =====
void encenderOLED();
void apagarOLED();
float leerLuz();
void leerBMP(float& bmpTemp, float& presion_hPa);
void mostrarPaginaActual();
void mostrarPaginaEnvio();
void borrarPendientesLittleFS();
void mostrarResumenFinal();
void mostrarBorradoRespaldo();
int calcularTiempoDesdeIndex(int idx);
void mostrarTiempo();
void mostrarPaginaSensoresDS(int inicio);
void mostrarPaginaAmbientales();
void mostrarPaginaVoltajes();
void guardarDatosPendientes(const String& payload);

```

```
void verContenidoLittleFS();
void enviarDatosPendientes();
void contarLineasLittleFS();
void controlarRele(float vb, float vp);
void configurarDespertarPorBoton();
void ejecutarModoNoche();
bool hayPendientesLittleFS();

void enviarInstantaneo(const char* evento) {
    // Opcional: asegura el rele en OFF para una lectura estable
    digitalWrite(RELAY_PIN, LOW);
    releActivo = false;

    // Lecturas
    sensoresDS18B20.requestTemperatures();
    float tempDS[8];
    for (int i = 0; i < 8; i++) tempDS[i] = sensoresDS18B20.getTempC(sensores[i]);

    sensors_event_t humidity, temp;
    aht.getEvent(&humidity, &temp);
    float bmpTemp, presion;
    leerBMP(bmpTemp, presion);
    float luz = leerLuz();
    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    DateTime nowRTC = rtc.now();
    char fechaHoraRTC[20];
    sprintf(fechaHoraRTC, "%04d-%02d-%02d %02d:%02d:%02d",
            nowRTC.year(), nowRTC.month(), nowRTC.day(),
            nowRTC.hour(), nowRTC.minute(), nowRTC.second());

    // Payload (igual al de parte 6/6 + etiqueta de evento)
    String payload = "s1=" + String(tempDS[0], 2);
    for (int i = 1; i < 8; i++) payload += "&s" + String(i + 1) + "=" + String(tempDS[i], 2);
    payload += "&aht=" + String(temp.temperature, 2);
    payload += "&hum=" + String(humidity.relative_humidity, 2);
    payload += "&bmp=" + String(bmpTemp, 2);
    payload += "&presion=" + String(presion, 2);
    payload += "&luz=" + String(luz, 2);
    payload += "&vp=" + String(vp, 2);
    payload += "&pp=" + String(pp, 0);
    payload += "&vb=" + String(vb, 2);
    payload += "&pb=" + String(pb, 0);
    payload += "&intervalo=" + String(timerActual);
    payload += "&timer_pot=" + String(timerPot);
```

```

payload += "&modo_dia=" + String(esModoDia ? "1" : "0");
payload += "&modo_noche=" + String(esModoNoche ? "1" : "0");
payload += "&rtc=" + urlEncode(String(fechaHoraRTC));
if (evento) payload += "&evento=" + String(evento); // ← marcador para tu Sheet
esReintento = hayPendientesLittleFS();
if (esReintento) payload += "&reintento=1";

// Asegura WiFi breve
if (WiFi.status() != WL_CONNECTED) {
    WiFi.mode(WIFI_STA);
    WiFi.persistent(false);
    WiFi.setSleep(true);

    WiFi.begin(ssid, password);
    unsigned long t0 = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - t0 < 5000) delay(500);
}

if (WiFi.status() == WL_CONNECTED) {
    // ...
    enviarDatosPendientes(); // primero respaldos
    auto r = postAWebApp(payload);

    if (r.ack) {
        Serial.printf(" Envío instantáneo OK (code=%d, body=%s)\n", r.http, r.body.c_str());
        estadoUltimoEnvio = "Enviado correctamente";
    } else {
        Serial.printf(" Error POST instantáneo: %d body=%s\n", r.http, r.body.c_str());
        estadoUltimoEnvio = "Fallo en envio";
        guardarDatosPendientes(payload);
    }
} else {
    Serial.println(" Sin WiFi en envío instantáneo. Respaldado.");
    estadoUltimoEnvio = "Sin WiFi";
    guardarDatosPendientes(payload);
}

// SIEMPRE apagar radio y marcar tiempo, conecte o no
WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
delay(50);
tEnvio = millis();
}

void borrarPendientesLittleFS() {
    LittleFS.remove("/pendientes.txt");
}

```

```
    Serial.println(" Archivo /pendientes.txt borrado por botón prolongado.");
}

void mostrarResumenFinal() {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }

    DateTime now = rtc.now() + TimeSpan(0, 0, 3, 0);

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.printf("%02d:%02d:%02d %02d/%02d/%04d\n",
                  now.hour(), now.minute(), now.second(),
                  now.day(), now.month(), now.year());

    display.setCursor(0, 15);
    display.println("Ciclo completado");
    display.println("Datos enviados");

    display.setCursor(0, 45);
    display.println("Modo noche activo");

    display.display();
}

void mostrarBorradoRespaldo() {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println(" Respaldo borrado");
    display.println("LittleFS limpio");
    display.display();
}

int calcularTiempoDesdeIndex(int idx) {
    if (idx < 60) return idx + 1;
    return 90 + (idx - 60) * 30;
```

```
}

void mostrarTiempo() {
    encenderOLED(); // <- PRIMERO inicializa/enciende
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print("Tiempo:");

    display.setCursor(0, 20);
    if (timerPot < 60) {
        display.print(timerPot);
        display.print(" min");
    } else {
        display.print(timerPot / 60.0, 1);
        display.print(" h");
    }

    display.setTextSize(1);
    display.setCursor(0, 50);
    display.print("Tap=OK Hold=Salir");
    display.display();
}

void mostrarPaginaSensoresDS(int inicio) {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }

    sensoresDS18B20.requestTemperatures();

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println(inicio == 0 ? "Sensores izquierda" : "Sensores derecha");

    for (int i = inicio; i < inicio + 4; i++) {
        float temp = sensoresDS18B20.getTempC(sensores[i]);
        display.printf("%s: %.1f C\n", nombresSensores[i], temp);
    }
}
```

```
    display.display();
}

void mostrarPaginaAmbientales() {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }

    sensors_event_t humidity, temp;
    aht.getEvent(&humidity, &temp);
    float luz = leerLuz();
    float bmpTemp, presion;
    leerBMP(bmpTemp, presion);

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.printf("AHT: %.1f C\n", temp.temperature);
    display.printf("Hum: %.1f %%\n", humidity.relative_humidity);
    display.printf("BMP: %.1f C\n", bmpTemp);
    display.printf("Pres: %.1f hPa\n", presion);
    display.printf("Luz: %.1f lx\n", luz);
    display.display();
}

void mostrarPaginaVoltajes() {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }

    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    bool bateriaBaja = vb < UMBRAL_BATERIA;
    bool panelSuficiente = vp > UMBRAL_PANEL;
    bool dentroHorario = esModoDia;

    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.printf("Bat: %.2f V (%.0f%%)\n", vb, pb);
    display.printf("Panel: %.2f V (%.0f%%)\n", vp, pp);
```

```

display.printf("Rele: %s\n", releActivo ? "ON" : "OFF");

display.setCursor(0, 33);
display.printf("Horario: %s\n", dentroHorario ? "OK" : "NO");
display.printf("VBaja : %s (<%.1fV)\n", bateriaBaja ? "SI" : "NO", UMBRAL_BATERIA);
display.printf("VPnl : %s (>%.1fV)\n", panelSuficiente ? "SI" : "NO", UMBRAL_PANEL);
display.display();
}

void guardarDatosPendientes(const String& payload) {
    String data = payload;
    data.trim(); // Elimina espacios en blanco o saltos de línea

    // Verifica que el payload tenga contenido válido
    if (data.length() < 10 || !data.startsWith("s1=")) {
        Serial.println(" Payload inválido. No se guarda respaldo.");
        return;
    }

    File archivo = LittleFS.open("/pendientes.txt", FILE_APPEND);
    if (archivo) {
        DateTime now = rtc.now();
        String fechaTexto = String(now.year()) + "-" + (now.month() < 10 ? "0" : "") + String(now.month());
        archivo.println("origen=backup&fecha=" + urlEncode(fechaTexto) + "&" + data);

        archivo.close();
        Serial.println(" Datos guardados en respaldo local con timestamp.");
    } else {
        Serial.println(" No se pudo guardar en LittleFS.");
    }
}

void verContenidoLittleFS() {
    File archivo = LittleFS.open("/pendientes.txt", FILE_READ);
    if (!archivo || archivo.size() == 0) {
        Serial.println(" Archivo vacío o no existe.");
        archivo.close();
        return;
    }
    Serial.println(" Contenido de /pendientes.txt:");
    while (archivo.available()) {
        String linea = archivo.readStringUntil('\n');
        linea.trim();
        Serial.println(" " + linea);
    }
    archivo.close();
}

```

```

}

void enviarDatosPendientes() {
    File file = LittleFS.open("/pendientes.txt", "r");
    if (!file || file.size() == 0) {
        Serial.println(" Archivo vacío o no existe.");
        if (file) file.close();
        return;
    }

    Serial.println(" Contenido de /pendientes.txt:");
    String nuevosPendientes = ""; // los que fallen se vuelven a guardar

    auto fixFecha = [](String& p) {
        int pos = p.indexOf("fecha=");
        if (pos >= 0) {
            int start = pos + 6;
            int amp = p.indexOf('&', start);
            if (amp < 0) amp = p.length();
            String v = p.substring(start, amp);
            v.replace(" ", "+"); // limpia espacios antiguos
            p = p.substring(0, start) + v + p.substring(amp);
        }
    };

    while (file.available()) {
        String payload = file.readStringUntil('\n');
        payload.trim();
        fixFecha(payload); // por si quedaron espacios en backups viejos
        if (payload.length() == 0) continue;

        auto r = postAWebApp(payload); // envía SOLO una vez
        if (r.ack) {
            Serial.printf(" Respaldo enviado (code=%d)\n", r.http);
        } else {
            Serial.printf(" Error al enviar respaldo: %d body=%s\n",
                         r.http, r.body.c_str());
            nuevosPendientes += payload + "\n";
        }
    }

    delay(350); // pequeño respiro (puedes quitarlo si quieres; ya tienes rate-limit de 1500 ms)
    file.close();
}

// sobrescribe el archivo con los que fallaron (o lo vacía si todo ok)
File outFile = LittleFS.open("/pendientes.txt", "w");
outFile.print(nuevosPendientes);
outFile.close();

```

```

if (nuevosPendientes.length() == 0) {
    Serial.println(" Todos los respaldos enviados. Archivo limpio.");
} else {
    Serial.println(" Algunos respaldos no se enviaron. Guardados nuevamente.");
}
}

void contarLineasLittleFS() {
    File archivo = LittleFS.open("/pendientes.txt", FILE_READ);
    int lineas = 0;
    if (archivo) {
        while (archivo.available()) {
            String linea = archivo.readStringUntil('\n');
            if (linea.length() > 0) lineas++;
        }
        archivo.close();
        Serial.printf(" Hay %d líneas pendientes en LittleFS.\n", lineas);
    } else {
        Serial.println("No se pudo abrir pendientes.txt para contar.");
    }
}

void controlarRele(float vb, float vp) {
    bool dentroHorario = esModoDia; // Ya se calcula en setup() y loop()
    bool bateriaBaja = vb < UMBRAL_BATERIA;
    bool panelSuficiente = vp > UMBRAL_PANEL;

    Serial.println(" Verificando condiciones del relé:");
    Serial.printf("Batería: %.2f V | Panel: %.2f V\n", vb, vp);
    Serial.printf("Horario OK: %s\n", dentroHorario ? "" : "");
    Serial.printf("Batería baja: %s (<%.2f V)\n", bateriaBaja ? "" : "", UMBRAL_BATERIA);
    Serial.printf("Panel suficiente: %s (>%.2f V)\n", panelSuficiente ? "" : "", UMBRAL_PANEL);

    if (dentroHorario && bateriaBaja && panelSuficiente) {
        digitalWrite(RELAY_PIN, HIGH);
        releActivo = true;
        Serial.println(" Relevador ACTIVADO");
    } else {
        digitalWrite(RELAY_PIN, LOW);
        releActivo = false;
        Serial.println(" Relevador DESACTIVADO");
    }
}

void mostrarPaginaActual() {
    switch (paginaActual) {
        case 0: mostrarPaginaSensoresDS(0); break;

```

```
case 1: mostrarPaginaSensoresDS(4); break;
case 2: mostrarPaginaAmbientales(); break;
case 3: mostrarPaginaVoltajes(); break;
case 4: mostrarPaginaEnvio(); break;
}
}

void configurarDespertarPorBoton() {
    pinMode(SW, INPUT_PULLUP);
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_4, 0); // solo wake-up por botón
}

void encenderOLED() {
    if (!displayInicializado) {
        if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
            Serial.println(" OLED no respondio en 0x3C");
            return;
        }
        displayInicializado = true;
        display.clearDisplay();
        display.display();
    } else {
        display.ssdi306_command(SSD1306_DISPLAYON);
    }
    oledOn = true;
}

void apagarOLED() {
    if (!displayInicializado) return;
    display.ssdi306_command(SSD1306_DISPLAYOFF);
    oledOn = false;
}

void mostrarPaginaEnvio() {
    encenderOLED();
    if (!displayInicializado) {
        Serial.println(" OLED no inicializado");
        return;
    }
    DateTime now = rtc.now() + TimeSpan(0, 0, 3, 0);

    display.clearDisplay();
    display.setTextSize(1);

    // 1) Fecha y hora
    display.setCursor(0, 0);
    display.printf("%02d:%02d:%02d ", now.hour(), now.minute(), now.second());
}
```

```

display.printf("%02d/%02d/%04d", now.day(), now.month(), now.year());

// 2) Tiempo seleccionado
display.setCursor(0, 15);
display.print("Tiempo sel.: ");
if (timerActual < 60) {
    display.print(timerActual);
    display.println(" min");
} else {
    display.print(timerActual / 60.0, 1);
    display.println(" h");
}

// 3) Último envío
display.setCursor(0, 30);
display.print("Ult. envio: ");
display.print((millis() - tEnvio) / 1000);
display.println(" s");

// 4) Parte actual
display.setCursor(0, 40);
display.printf("Parte %d/6", parteCiclo);

// 5) Tiempo restante (fórmula robusta con delta)
long restanteMs = (long)intervaloParteMs - (long)(millis() - tInicioParteMs);
if (restanteMs < 0) restanteMs = 0;
long tiempoRestanteS = restanteMs / 1000;

display.setCursor(0, 47);
display.printf("restan %lds", tiempoRestanteS);

// 6) Estado del envío
display.setCursor(0, 55);
display.print(estadoUltimoEnvio);

display.display();
}

void ejecutarModoNoche() {
    DateTime now = rtc.now();

    // Calcular hora del próximo inicio del modo día
    DateTime inicioDiaHoy(now.year(), now.month(), now.day(), HORA_INICIO_DIA, MINUTO_INICIO_DIA);
    DateTime inicioDiaManana = inicioDiaHoy + TimeSpan(1, 0, 0, 0);
    DateTime proximoInicioDia = (now.unixtime() < inicioDiaHoy.unixtime()) ? inicioDiaHoy : inicioDiaManana;

    // Verificación crítica
    if (now.unixtime() >= proximoInicioDia.unixtime()) {

```

```
Serial.println(" Ya estamos en horario de modo día. Cancelando modo noche.");
  return;
}

char buffer[25];
sprintf(buffer, "%04d-%02d-%02d %02d:%02d:%02d", now.year(), now.month(), now.day(), now.hour());
String fechaHoraRTC = String(buffer);

Serial.println(" Ejecutando modo noche...");

if (!tiempoConfirmado || envioNocturnoHecho) return;

// Leer sensores
sensoresDS18B20.requestTemperatures();
float tempDS[8];
for (int i = 0; i < 8; i++) tempDS[i] = sensoresDS18B20.getTempC(sensores[i]);

sensors_event_t humidity, temp;
aht.getEvent(&humidity, &temp);
float bmpTemp, presion;
leerBMP(bmpTemp, presion);
float luz = leerLuz();
float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
float pb = vb / 4.2 * 100.0;
float pp = vp / 5.0 * 100.0;

// Construir payload
String payload = "s1=" + String(tempDS[0], 2);
for (int i = 1; i < 8; i++) payload += "&s" + String(i + 1) + "=" + String(tempDS[i], 2);
payload += "&aht=" + String(temp.temperature, 2);
payload += "&hum=" + String(humidity.relative_humidity, 2);
payload += "&bmp=" + String(bmpTemp, 2);
payload += "&presion=" + String(presion, 2);
payload += "&luz=" + String(luz, 2);
payload += "&vp=" + String(vp, 2);
payload += "&pp=" + String(pp, 0);
payload += "&vb=" + String(vb, 2);
payload += "&pb=" + String(pb, 0);
payload += "&intervalo=" + String(timerActual);
payload += "&timer_pot=" + String(timerPot);

payload += "&rtc=" + urlEncode(fechaHoraRTC);
payload += "&modo_dia=" + String(esModoDia ? "1" : "0");
payload += "&modo_noche=" + String(esModoNoche ? "1" : "0");
esReintento = hayPendientesLittleFS(); // Puedes cambiarlo por true si quieres forzar pruebas
if (esReintento) payload += "&reintento=1";
```

```
WiFi.mode(WIFI_STA);
WiFi.persistent(false);
WiFi.setSleep(true);

// Conexión WiFi
WiFi.begin(ssid, password);
int intentos = 0;
while (WiFi.status() != WL_CONNECTED && intentos < 20) {
    delay(500);
    Serial.print(".");
    intentos++;
}

// Espera hasta 5 segundos a que se conecte el WiFi
unsigned long tiempoInicio = millis();
while (WiFi.status() != WL_CONNECTED && millis() - tiempoInicio < 5000) {
    Serial.println(" Esperando conexión WiFi...");
    delay(500);
}

if (WiFi.status() == WL_CONNECTED) {
    // ...
    enviarDatosPendientes();
    auto r = postAWebApp(payload);

    if (r.ack) {
        Serial.printf(" Datos enviados a Google Sheets (code=%d)\n", r.http);
        estadoUltimoEnvio = "Enviado correctamente";
        borrarPendientesLittleFS();
    } else {
        Serial.printf(" Error POST nocturno: %d body=%s\n", r.http, r.body.c_str());
        estadoUltimoEnvio = "Fallo en envio";
        guardarDatosPendientes(payload);
    }
} else {
    Serial.println(" Sin conexión WiFi. Guardando respaldo...");
    estadoUltimoEnvio = "Sin WiFi";
    guardarDatosPendientes(payload);
}

// Radio OFF siempre (tengas o no WiFi)
WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
delay(50);

// Mensaje rápido en OLED (opcional)
if (displayInicializado) {
    encenderOLED();
```

```

display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 0);
DateTime nowOLED = rtc.now() + TimeSpan(0, 0, 3, 0);
display.printf("%02d:%02d %02d/%02d\n", nowOLED.hour(), nowOLED.minute(), nowOLED.day(), nowOLED.year());
display.println(estadoUltimoEnvio);
display.println("Modo noche activo");
display.display();
delay(5000);
apagarOLED();
}

envioNocturnoHecho = true;

// Calcular y dormir
unsigned long segundosSleep = timerActual * 60;
unsigned long segundosHastaModoDia = proximoInicioDia.unixtime() - now.unixtime();
if (segundosHastaModoDia < segundosSleep) segundosSleep = segundosHastaModoDia;

if (segundosSleep == 0) {
    Serial.println(" Ya es momento de iniciar el modo día. No se duerme.");
    return;
}

Serial.printf(" Ahora: %02d:%02d, Modo día a: %02d:%02d, durmiendo %lu seg\n",
             now.hour(), now.minute(), proximoInicioDia.hour(), proximoInicioDia.minute(), segundosSleep);

esp_sleep_enable_timer_wakeup(segundosSleep * 1000000ULL);
configurarDespertarPorBoton();
esp_deep_sleep_start();
} // ← CIERRE de ejecutarModoNoche()

bool hayPendientesLittleFS() {
    File archivo = LittleFS.open("/pendientes.txt", "r");
    bool hayDatos = archivo && archivo.size() > 0;
    archivo.close();
    return hayDatos;
}

void leerBMP(float& bmpTemp, float& presion_hPa) {
    bmp.takeForcedMeasurement(); // dispara 1 medición
    bmpTemp = bmp.readTemperature();
    presion_hPa = bmp.readPressure() / 100.0;
}

// Lectura one-shot del BH1750 para ahorrar energía
float leerLuz() {

```

```
// medición one-shot: el sensor se apaga al terminar
lightMeter.configure(BH1750::ONE_TIME_HIGH_RES_MODE);
delay(180); // ~120{180 ms para conversión
float lx = lightMeter.readLightLevel();
return lx < 0 ? 0 : lx; // evita valores negativos por error
}

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

    // Mostrar valores persistentes al inicio
    Serial.printf(" [INICIO] parteCiclo=%d | tInicioParteMs=%lu | intervaloParteMs=%lu\n", par
    WiFi.mode(WIFI_STA);
    WiFi.persistent(false);
    WiFi.setSleep(true);
    Wire.begin();
    Wire.setClock(100000); // 100 kHz
    Wire.setTimeOut(50); // 50 ms

    // Inicializar LittleFS
    if (!LittleFS.begin()) {
        Serial.println(" Error montando LittleFS");
    } else {
        Serial.println(" LittleFS montado correctamente");
    }

    // Si alguien solicitó borrar respaldos (por hold largo), hazlo ahora
    if (borrarPendientesSolicitado) {
        borrarPendientesLittleFS();
        borrarPendientesSolicitado = false;
        Serial.println(" Respaldo borrado por hold largo.");
    }

    esp_sleep_wakeup_cause_t wakeupReason = esp_sleep_get_wakeup_cause();
    Serial.print("Wakeup cause: ");
    Serial.println((int)wakeupReason);
    Serial.print("Reset reason: ");
    Serial.println((int)esp_reset_reason());

    // Reinicio completo (no viene de deep sleep)
    if (esp_reset_reason() != ESP_RST_DEEPSLEEP) {
        Serial.println(" Reinicio externo o botón RESET: limpiando estado visual");
        // en vez de poner displayInicializado=false, simplemente apaga
        apagarOLED();
        modoVisualizacion = false;
        paginaActual = 0;
    }
}
```

```
// Despertar por botón
if (wakeupReason == ESP_SLEEP_WAKEUP_EXT0) {
    Serial.println(" Despertar por botón (modo nocturno)");

    encenderOLED(); // inicializa OLED aquí
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println("Boton mantenido?");
    display.println("Hold 10s = borrar");
    display.println("Tap = seleccionar tiempo");
    display.display();

    pinMode(SW, INPUT_PULLUP);
    unsigned long tiempoPresionado = 0;
    while (digitalRead(SW) == LOW) {
        tiempoPresionado++;
        delay(100);
        if (tiempoPresionado > 100) break;
    }

    if (tiempoPresionado > 100) {
        borrarPendientesLittleFS();
        mostrarBorradoRespaldo();
        delay(2000);
    }

    modoVisualizacion = false;
    tiempoConfirmado = false;
    envioNocturnoHecho = false;
    estadoUltimoEnvio = "Sin enviar";
    // NO hagas return; deja que el setup siga y configure pines/sensores/WiFi
}

// Despertar desde sleep programado (modo noche)
if (wakeupReason == ESP_SLEEP_WAKEUP_TIMER) {
    Serial.println(" Despertar desde sleep nocturno");
    tiempoConfirmado = true;
    modoVisualizacion = false;
    envioNocturnoHecho = false;
} else {
    Serial.println(" Inicio limpio, selección de tiempo");
    modoVisualizacion = false;
}

delay(100);
```

```

// RTC
if (!rtc.begin()) {
    Serial.println(" No se encontró el RTC");
} else if (!rtc.isrunning()) {
    Serial.println(" RTC no está corriendo, se ajustará a hora de compilación.");
    rtc.adjust(DateTime(2025, 7, 24, 9, 59, 30));
} else {
    Serial.println(" RTC iniciado correctamente.");
}

// Validar parteCiclo tras RTC
if (parteCiclo < 1 || parteCiclo > 6) {
    parteCiclo = 1;
    Serial.println(" parteCiclo fuera de rango, reiniciado a 1");
}

// Determinar si es modo día o noche
DateTime now = rtc.now();
unsigned long ahoraUnix = now.unixtime();

DateTime inicioDia(now.year(), now.month(), now.day(), HORA_INICIO_DIA, MINUTO_INICIO_DIA,
DateTime finDia(now.year(), now.month(), now.day(), HORA_FIN_DIA, MINUTO_FIN_DIA, 0);

esModoDia = (ahoraUnix >= inicioDia.unixtime()) && (ahoraUnix < finDia.unixtime());
esModoNoche = !esModoDia;

Serial.printf(" RTC: %02d:%02d → esModoDia: %s\n", now.hour(), now.minute(), esModoDia ? "Día" : "Noche");

if (esModoDia && intervaloParteMs == 0) {
    intervaloParteMs = (timerActual * 60000UL) / 6;
    tInicioParteMs = millis();
    Serial.println(" Restaurando intervaloParteMs y tInicioParteMs después de deep sleep");
}

// Pines
pinMode(CLK, INPUT_PULLUP);
pinMode(DT, INPUT_PULLUP);
pinMode(SW, INPUT_PULLUP);
pinMode(RELAY_PIN, OUTPUT);
digitalWrite(RELAY_PIN, LOW);

// Smoke test del OLED (mover aquí)
encenderOLED();
if (displayInicializado) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println("OLED OK");
}

```

```
display.display();
delay(800);
} else {
  Serial.println(" OLED no respondio en 0x3C");
}

// Inicializar sensores siempre (independiente del display)
aht.begin();
bmp.begin(0x77);
bmp.setSampling(Adafruit_BMP280::MODE_FORCED,
                Adafruit_BMP280::SAMPLING_X1, // temp
                Adafruit_BMP280::SAMPLING_X1, // presión
                Adafruit_BMP280::FILTER_OFF,
                Adafruit_BMP280::STANDBY_MS_1);
lightMeter.begin(BH1750::ONE_TIME_HIGH_RES_MODE);

sensoresDS18B20.begin();

// Solo si AÚN no han confirmado el tiempo
if (!tiempoConfirmado) {
  modoVisualizacion = false;
  timerPot = max(1, timerPot);
  mostrarTiempo();
  if (!esModoDia) apagarOLED();
}

// Estado inicial de visualización (evitar encender OLED en noche)
if (esModoDia) {
  if (tiempoConfirmado) {
    modoVisualizacion = true;
    paginaActual = 0;
    encenderOLED();
    mostrarPaginaActual();
    if (intervaloParteMs == 0) intervaloParteMs = (timerActual * 60000UL) / 6;
    if (tInicioParteMs == 0) tInicioParteMs = millis();
    if (parteCiclo < 1 || parteCiclo > 6) parteCiclo = 1;
  } else {
    // Ya mostramos "Tiempo" ANTES en setup. No toques el OLED aquí.
    // modoVisualizacion = false; // (ya lo pusimos antes del mostrarTiempo)
  }
} else {
  modoVisualizacion = false;
  apagarOLED(); // apaga en noche
}

lastClkState = digitalRead(CLK);

// Conexión WiFi (independiente del OLED)
```

```
if (displayInicializado && (esModoDia || modoVisualizacion)) {  
    display.clearDisplay();  
    display.setTextSize(1);  
    display.setCursor(0, 0);  
    display.println("Conectando WiFi...");  
    display.display();  
}  
  
if (esModoDia || modoVisualizacion) {  
    WiFi.mode(WIFI_STA);  
    WiFi.persistent(false);  
    WiFi.setSleep(true);  
    WiFi.begin(ssid, password);  
  
    int intentos = 0;  
    while (WiFi.status() != WL_CONNECTED && intentos < 20) {  
        delay(500);  
        if (displayInicializado) {  
            display.print(".");  
            display.display();  
        }  
        intentos++;  
    }  
  
    if (displayInicializado) {  
        display.clearDisplay();  
        display.setCursor(0, 0);  
        if (WiFi.status() == WL_CONNECTED) {  
            display.println("WiFi conectado!");  
            display.println(WiFi.localIP());  
        } else {  
            display.println("Error WiFi");  
        }  
        display.display();  
        delay(2000);  
    }  
  
    WiFi.disconnect(true);  
    WiFi.mode(WIFI_OFF);  
    delay(50);  
  
} else {  
    WiFi.disconnect(true);  
    WiFi.mode(WIFI_OFF);  
    delay(50);  
}  
  
// Mostrar contenido LittleFS
```

```

verContenidoLittleFS();
contarLineasLittleFS();

// Inicializar ciclo si es modo día
if (esModoDia && !tiempoConfirmado) {
    parteCiclo = 1;
    tInicioParteMs = millis();
    intervaloParteMs = (timerActual * 60000UL) / 6;
    Serial.printf(" Inicializando ciclo día desde setup → tInicioParteMs=%lu ms\n", tInicioParteMs);
}
}

void loop() {
    DateTime now = rtc.now();
    unsigned long ahoraUnix = now.unixtime();

    static int ultimoMinutoMostrado = -1;
    if (now.minute() != ultimoMinutoMostrado) {
        ultimoMinutoMostrado = now.minute();
        Serial.printf(" RTC (loop): %02d:%02d → esModoDia: %s\n",
                      now.hour(), now.minute(), esModoDia ? "SI" : "NO");
    }

    // Recalcular horarios
    DateTime inicioDia(now.year(), now.month(), now.day(), HORA_INICIO_DIA, MINUTO_INICIO_DIA, 0);
    DateTime finDia(now.year(), now.month(), now.day(), HORA_FIN_DIA, MINUTO_FIN_DIA, 0);
    esModoDia = (ahoraUnix >= inicioDia.unixtime()) && (ahoraUnix < finDia.unixtime());
    esModoNoche = !esModoDia;

    // Transición Día→Noche
    if (!estabaEnModoNoche && esModoNoche) {
        Serial.println(" Transición Día→Noche");
        envioNocturnoHecho = false;
        modoVisualizacion = false;
        paginaActual = 0;
        apagarOLED();
    }

    // Transición Noche→Día
    if (estabaEnModoNoche && esModoDia) {
        Serial.println(" Cambio de Modo Noche a Día");
        if (tiempoConfirmado) {
            enviarInstantaneo("inicio_dia"); // marcador en Sheet
        }
        parteCiclo = 1;
        intervaloParteMs = (timerActual * 60000UL) / 6;
        tInicioParteMs = millis();
        modoVisualizacion = true;
    }
}

```

```

paginaActual = 0;
// mantener consistencia con el selector
timerPot = timerActual;
encenderOLED();
mostrarPaginaActual();
}

// Ejecutar modo noche cuando toque
if (esModoNoche && tiempoConfirmado && !envioNocturnoHecho) {
    ejecutarModoNoche();
}

// -- manejo de encoder --
int clkState = digitalRead(CLK);
if (clkState != lastClkState && clkState == LOW) {
    int dtState = digitalRead(DT);
    int direction = (dtState != clkState) ? 1 : -1;
    if (!modoVisualizacion) {
        indexSeleccion += direction;
        indexSeleccion = constrain(indexSeleccion, 0, MAX_INDEX);
        timerPot = calcularTiempoDesdeIndex(indexSeleccion);
        mostrarTiempo();
    } else {
        paginaActual = (paginaActual + direction + 5) % 5;
        mostrarPaginaActual();
    }
    delay(2);
}
lastClkState = clkState;

// -- manejo de botón --
bool btn = (digitalRead(SW) == LOW);
if (btn && btnPressedAt == 0) btnPressedAt = millis();
if (!btn && btnPressedAt > 0) {
    unsigned long pressTime = millis() - btnPressedAt;
    if (pressTime > 5000) {
        borrarPendientesSolicitado = false; // no diferir
        borrarPendientesLittleFS(); // borra ya
        mostrarBorradoRespaldo(); // feedback en OLED
        delay(1500);
    } else if (pressTime > 2000) {
        modoVisualizacion = false;
        tiempoConfirmado = false;
        mostrarTiempo();
        estadoUltimoEnvio = "Sin enviar";
    } else if (!modoVisualizacion) {
        // TAP CORTO (<2s): confirmar tiempo
        timerActual = timerPot;
    }
}

```

```

delay(200);
lastClkState = digitalRead(CLK); // re-sincroniza encoder
tiempoConfirmado = true;
Serial.println(" Botón presionado: tiempo confirmado");
if (esModoNoche) {
    envioNocturnoHecho = false; // permitir un envío nocturno nuevo
    ejecutarModoNoche(); // hace lectura, envío y duerme con timerActual
    return; // evita seguir ejecutando este ciclo
}
if (esModoDia) {
    intervaloParteMs = (timerActual * 60000UL) / 6;
    tInicioParteMs = millis();
    parteCiclo = 1;
}
modoVisualizacion = true;
paginaActual = 0;
tEnvio = millis();
estadoUltimoEnvio = "Sin enviar";
encenderOLED();
mostrarPaginaActual();
}
btnPressedAt = 0;
}

// Si cambió a modo noche y ya se había confirmado el tiempo, apaga UI
if (esModoNoche && tiempoConfirmado && modoVisualizacion) {
    Serial.println(" Transición automática a modo nocturno");
    modoVisualizacion = false;
    paginaActual = 0;
    apagarOLED();
}

// Salida temprana si aún no confirman tiempo
if (!tiempoConfirmado) {
    estabaEnModoNoche = esModoNoche;
    return;
}

// === MODO DÍA ===
if (esModoDia) {
    // Asegurar intervalos y arranque de parte
    if (intervaloParteMs == 0) {
        intervaloParteMs = (timerActual * 60000UL) / 6;
    }
    if (tInicioParteMs == 0) {
        tInicioParteMs = millis();
    }
}

```

```

// Si falta menos que una parte para que empiece la noche, apaga UI y espera
unsigned long msHastaModoNoche = (finDia.unixtime() - ahoraUnix) * 1000UL;
if (msHastaModoNoche < intervaloParteMs) {
    modoVisualizacion = false;
    paginaActual = 0;
    if (oledOn) apagarOLED();
    estabaEnModoNoche = esModoNoche;
    return;
}

// ----- Ventana 20% (inicio/fin de cada parte) con flancos + 1 Hz -----
if (esModoDia && modoVisualizacion && tiempoConfirmado) {
    unsigned long tiempoParte = millis() - tInicioParteMs;
    unsigned long tVisual = (intervaloParteMs * 20UL) / 100UL; // 20%
    bool visOn = (tiempoParte <= tVisual) || (tiempoParte >= (intervaloParteMs - tVisual))

    static int partePrev = -1;
    static bool visOnPrev = false;
    static unsigned long lastUiRefreshMs = 0;

    // Si cambió la parte, fuerza flanco de subida y pinta al instante
    if (parteCiclo != partePrev) {
        visOnPrev = false; // asegura detección de flanco
        lastUiRefreshMs = 0;
        encenderOLED();
        mostrarPaginaActual(); // repintado inmediato al entrar a nueva parte
    }

    // Flancos ON/OFF + refresco a 1 Hz mientras está ON
    if (visOn && !visOnPrev) {
        encenderOLED();
        mostrarPaginaActual();
        lastUiRefreshMs = millis();
    } else if (!visOn && visOnPrev) {
        if (oledOn) apagarOLED();
    } else if (visOn && (millis() - lastUiRefreshMs >= 1000)) {
        mostrarPaginaActual();
        lastUiRefreshMs = millis();
    }

    visOnPrev = visOn;
    partePrev = parteCiclo;
} else {
    if (oledOn) apagarOLED();
}

```

```
// Partes del ciclo
if (millis() - tInicioParteMs >= intervaloParteMs) {
    Serial.printf(" Parte %d/6 del ciclo (MODO DÍA)\n", parteCiclo);

    float vb = analogRead(PIN_BATERIA) * 2 * 3.3 / 4095.0;
    float vp = analogRead(PIN_PANEL) * 2 * 3.3 / 4095.0;
    float pb = vb / 4.2 * 100.0;
    float pp = vp / 5.0 * 100.0;

    DateTime nowRTC = rtc.now();
    char fechaHoraRTC[20];
    sprintf(fechaHoraRTC, "%04d-%02d-%02d %02d:%02d:%02d",
            nowRTC.year(), nowRTC.month(), nowRTC.day(),
            nowRTC.hour(), nowRTC.minute(), nowRTC.second());

    if (parteCiclo <= 5) {
        // Partes 1..5: solo control del relé
        controlarRele(vb, vp);

    } else {
        // Parte 6: lecturas + envío
        Serial.println(" Parte final: apagando rele y enviando datos...");
        digitalWrite(RELAY_PIN, LOW);
        releActivo = false;
        delay(300);

        sensoresDS18B20.requestTemperatures();
        float tempDS[8];
        for (int i = 0; i < 8; i++) tempDS[i] = sensoresDS18B20.getTempC(sensores[i]);

        sensors_event_t humidity, temp;
        aht.getEvent(&humidity, &temp);
        float bmpTemp, presion;
        leerBMP(bmpTemp, presion);
        float luz = leerLuz();

        esReintento = hayPendientesLittleFS();

        String payload = "s1=" + String(tempDS[0], 2);
        for (int i = 1; i < 8; i++) payload += "&s" + String(i + 1) + "=" + String(tempDS[i], 2);
        payload += "&aht=" + String(temp.temperature, 2);
        payload += "&hum=" + String(humidity.relative_humidity, 2);
        payload += "&bmp=" + String(bmpTemp, 2);
        payload += "&presion=" + String(presion, 2);
        payload += "&luz=" + String(luz, 2);
        payload += "&vp=" + String(vp, 2);
        payload += "&pp=" + String(pp, 0);
        payload += "&vb=" + String(vb, 2);
```

```

payload += "&pb=" + String(pb, 0);
payload += "&intervalo=" + String(timerActual);
payload += "&timer_pot=" + String(timerPot);
payload += "&modo_dia=" + String(esModoDia ? "1" : "0");
payload += "&modo_noche=" + String(esModoNoche ? "1" : "0");
payload += "&rtc=" + urlEncode(String(fechaHoraRTC));
if (esReintento) payload += "&reintento=1";

if (WiFi.status() != WL_CONNECTED) {
    WiFi.mode(WIFI_STA);
    WiFi.persistent(false);
    WiFi.setSleep(true);
    WiFi.begin(ssid, password);
}

unsigned long t0 = millis();
while (WiFi.status() != WL_CONNECTED && millis() - t0 < 5000) {
    Serial.println(" Esperando conexión WiFi...");
    delay(500);
}

if (WiFi.status() == WL_CONNECTED) {
    // ...
    enviarDatosPendientes();
    auto r = postAWebApp(payload);

    if (r.ack) {
        Serial.printf(" Envío diurno OK (code=%d)\n", r.http);
        estadoUltimoEnvio = "Enviado correctamente";
    } else {
        Serial.printf(" Error POST diurno: %d body=%s\n", r.http, r.body.c_str());
        estadoUltimoEnvio = "Fallo en envío";
        guardarDatosPendientes(payload);
    }
} else {
    Serial.println(" Sin WiFi en envío instantáneo. Respaldado.");
    estadoUltimoEnvio = "Sin WiFi";
    guardarDatosPendientes(payload);
}

// Apaga la radio UNA sola vez (solo si hubo intento de envío)
WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
delay(50);

tEnvio = millis();
controlarRele(vb, vp); // asegura estado tras envío
}

```

```

    // Avanza SIEMPRE de parte (1..5 o 6)
    parteCiclo++;
    if (parteCiclo > 6) parteCiclo = 1;
    tInicioParteMs = millis(); // Reiniciar contador para siguiente parte
}
}

// Tracker de estado SIEMPRE al final del loop
estabaEnModoNoche = esModoNoche;
}

```

Apéndice D. Repositorios de GitHub del sistema Web IoT

En este apéndice se documentan los repositorios públicos que contienen el código fuente de la interfaz Web y de la API utilizada para la comunicación con la nube.

Repositorio (Frontend Web): https://github.com/hugonogueron2607/Techo_Verde

Navegación sugerida:

- **Raíz del proyecto:**
 - `index.html`: documento principal de la interfaz Web.
 - `js/main.js`: lógica en JavaScript para consumo de la API y actualización del DOM.
 - (Opcional) `css/` estilos, `assets/` imágenes/iconos.
- **Ejecución local:** abrir `index.html` en el navegador o servir el contenido estático con, p. ej., `python -m http.server 8000` desde la carpeta raíz.

Repositorio (Backend/API con FastAPI): https://github.com/hugonogueron2607/techo_verde_api

Navegación sugerida:

- **Archivos de aplicación:**
 - `app/` o `src/`: paquete con `main.py` (*FastAPI*), `routers/` (endpoints), `schemas/` y `services/`.
 - `requirements.txt`: dependencias de Python.
 - `README.md`: instrucciones de despliegue y variables de entorno.
- **Despliegue en Render:** incluye los archivos necesarios para lanzar el servidor (p. ej., `render.yaml/Procfile` o comando de inicio).
 - **Comando típico de ejecución:** `uvicorn main:app --host 0.0.0.0 --port 10000`
- **Ejecución local:**
 1. Crear entorno: `python -m venv .venv && source .venv/bin/activate`
 2. Instalar deps: `pip install -r requirements.txt`
 3. Ejecutar: `uvicorn main:app --reload`

Bibliografía

- [1] M. Santamouris, “On the energy impact of urban heat island and global warming on buildings,” *Energy and Buildings*, vol. 82, págs. 100-113, 2014.
- [2] D. Li y E. Bou-Zeid, “Synergistic Interactions between Urban Heat Islands and Heat Waves: The Impact in Cities Is Larger than the Sum of Its Parts,” *Journal of Applied Meteorology and Climatology*, vol. 52, págs. 2051-2064, 2013.
- [3] E. Cuce, “Thermal and energy performance assessments of green roofs under diverse climatic conditions,” *Energy and Buildings*, 2025, in press.
- [4] I. e. a. Asadi, “Evaluating the thermal performance of green roofs: Experimental and numerical approach,” *Building and Environment*, vol. 169, págs. 106 544, 2020.
- [5] R. e. a. Arabi, “Environmental benefits of green roofs on urban areas,” *Procedia Engineering*, vol. 118, págs. 1010-1017, 2015.
- [6] G. e. a. Mihalakakou, “Experimental evaluation of a green roof system’s performance in urban heat island mitigation,” *Sustainable Cities and Society*, vol. 90, págs. 104 285, 2023.
- [7] C. e. a. Baicu, “IoT-based smart environmental monitoring with microcontrollers,” *Sensors and Actuators A: Physical*, vol. 351, págs. 113 054, 2024.
- [8] T. e. a. Padma, “Low-cost thermal monitoring for agriculture using DS18B20 sensors,” *Computers and Electronics in Agriculture*, vol. 201, págs. 107 378, 2023.
- [9] A. e. a. Sharma, “Green roofs as a solution to the urban heat island effect: A review,” *Renewable and Sustainable Energy Reviews*, vol. 60, págs. 1395-1406, 2016.

- [10] P. e. a. Bevilacqua, "Performance analysis of green roofs in a Mediterranean climate," *Renewable Energy*, vol. 146, págs. 1755-1767, 2020.
- [11] S. Cascone, "A comprehensive review of cool roof and green roof mitigation strategies for urban heat island effect," *Renewable and Sustainable Energy Reviews*, vol. 82, págs. 416-428, 2018.
- [12] M. e. a. Lambarki, "Remote sensing and in-situ measurements of green roof thermal behavior," *Remote Sensing Applications: Society and Environment*, vol. 25, pág. 100681, 2024.
- [13] D. Serpanos y M. Wolf, *Internet-of-Things (IoT) Systems Architectures, Algorithms, Methodologies*. Springer, 2018.
- [14] P. D. Singh y M. Angurala, *Integration of Cloud Computing and IoT: Trends, Case Studies and Applications*. CRC press, 2025.
- [15] ResearchGate, "Wireless communication technologies for the Internet of Things," *ResearchGate*, 2021. [En línea]. Disponible en: https://www.researchgate.net/publication/355163956_Wireless_communication_technologies_for_the_Internet_of_Things.
- [16] T. IoT, *IoT connectivity overview & definition*, 2021. [En línea]. Disponible en: <https://iot.telenor.com/technologies/connectivity/>.
- [17] *Microcontrollers for the Internet of Things*, <https://www.edgeimpulse.com/blog/microcontrollers-for-the-internet-of-things>, Accessed: 2025-04-14.
- [18] *ESP32 Series Datasheet*, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, Accessed: 2025-04-14.
- [19] *Raspberry Pi in IoT Applications*, <https://www.raspberrypi.com/news/raspberry-pi-iot-applications/>, Accessed: 2025-04-14.
- [20] *STM32 Microcontroller Application Notes*, <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>, Accessed: 2025-04-14.
- [21] M. Banzi y M. Shiloh, *Getting Started with Arduino*. Make: Books, 2014, ISBN: 978-1-4493-6334-5.
- [22] ResearchGate, "Microcontrollers and their applications in IoT systems," *ResearchGate*, 2021. [En línea]. Disponible en: https://www.researchgate.net/publication/355163957_Microcontrollers_and_their_applications_in_IoT_systems.
- [23] L. G. C. R. J. M. C. G. S. A. Jiménez, *Sensores y actuadores. Aplicaciones con Arduino*, 1st. Larousse - Grupo Editorial Patria, 2014.

- [24] T. Domínguez-Bolaño, O. Campos, V. Barral, C. J. Escudero y J. A. García-Naya, “An overview of IoT architectures, technologies, and existing open-source projects,” *arXiv preprint arXiv:2401.15441*, 2024.
- [25] M. Polo-Labarrios y A. Martínez, “Experimental analysis of green roof performance in warm climates,” *Sustainable Cities and Society*, vol. 52, pág. 101 866, 2020.
- [26] Y. e. a. Tang, “Monitoring and simulation of green roof cooling performance under hot weather conditions,” *Applied Energy*, vol. 239, págs. 123-138, 2019.
- [27] A. e. a. Masali, “Design of a solar-powered wireless sensor node for environmental monitoring,” *Renewable Energy*, vol. 217, págs. 321-332, 2024.
- [28] S. e. a. Junus, “Stability and precision of DS18B20 sensors in aquatic monitoring,” *Environmental Monitoring and Assessment*, vol. 196, pág. 450, 2024.
- [29] J. e. a. Wang, “AI-based prediction and optimization of green roof cooling performance,” *Energy Reports*, vol. 8, págs. 1862-1875, 2022.