Registro Diario de Avances – Sprint 1

**Fecha:01/10/2025**

**Autores: Pool K. Apaza, Erick K. Aguila, Victor E. Condori, Crystian L. García**

**Versión:** 1.0

# 1. Introducción

Este documento detalla el seguimiento diario del Sprint 1 del proyecto Invernadero Automatizado, centrado en el desarrollo de funcionalidades básicas. Se incluyen actividades diarias, tareas completadas, evidencias de código, un burndown chart y un análisis del rendimiento del equipo.

# 2. Lineamientos de Diseño

Desarrollar una primera versión funcional del prototipo de Invernadero Automatizado que va a permitir:

* Registrar y almacenar datos de sensores en tiempo real (temperatura y humedad del suelo).
* Visualizar los datos recolectados mediante una interfaz sencilla.
* Guardar el historial de datos de manera persistente utilizando **la base de datos**.
* Establecer alertas o indicadores cuando los valores de temperatura o humedad salgan de los rangos óptimos.

# 3. Historias de Usuario y Tareas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| HU-1.1 | Visualización en tiempo real | 6 días. | 22/09/2025 | 27/09/2025 |
| HU-1.2 | Registro histórico | 4 días. | 28/10/2025 | 02/10/2025 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Tarea | Responsable | Estimación (Horas) | Estado |
| 1.1.1 | Investigar sensores de humedad compatibles con Arduino | Pool | 4 | Concluido |
| 1.1.2 | Conectar y programar sensor de humedad | Victor | 6 | Concluido |
| 1.1.3 | Diseñar la interfaz de usuario para mostrar la lectura de humedad | Cristian | 5 | Concluido |
| 1.1.4 | Integrar la lectura del sensor a la interfaz | Erick | 6 | Concluido |
| 1.1.5 | Probar la visualización de datos en tiempo real | Victor | 4 | Pendiente |
| 1.1.6 | Implementar alerta en caso de fallo del sensor | Erick | 3 | Pendiente |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Tarea** | **Responsable** | **Estimación (Horas)** | **Estado** |
| **1.2.1** | Configurar base de datos local para almacenar lecturas diarias | Pool | 8 | Pendiente |
| **1.2.2** | Crear la lógica de la base de datos para registrar lecturas | Victor | 8 | Pendiente |
| **1.2.3** | Desarrollar la interfaz para consultar y visualizar el historial | Cristian | 6 | Pendiente |
| **1.2.4** | Implementar la funcionalidad de guardar y mostrar el historial | Erick | 8 | Pendiente |
| **1.2.5** | Realizar pruebas de base de datos y consultas | Victor | 6 | Pendiente |

# 4. Evidencias de Investigación de Sensores

**1.1.1 Investigación: sensores de humedad compatibles con Arduino**

## 1. Objetivo y alcance

Objetivo: proporcionar una guía técnica y práctica (investigación) para elegir, conectar, calibrar y usar sensores de humedad de suelo y humedad ambiental (aire) con placas Arduino (UNO, Nano, Mega, ESP32, etc.).  
 Alcance: revisión de tipos comerciales comunes (resistivos, capacitivos, DHTxx, BME280, SHT31/SHT3x, SI7021), comparativa de especificaciones, instrucciones de conexión, código de ejemplo, métodos de calibración/validación y recomendaciones para proyectos hobby y aplicaciones más exigentes (agricultura, estación meteorológica, IoT).

## 2. Resumen ejecutivo / recomendación rápida

● Para proyectos baratos y temporales (hobby): YL-69 / FC-28 (resistivo) + DHT11 (ambiente).

● Para proyectos permanentes/fiables de riego: sensor capacitivo de humedad de suelo (v1.2 o similar) + DHT22 o BME280.

● Para estaciones meteorológicas o aplicaciones profesionales: BME280 o SHT31 (humedad con ±2–3 %RH) y sensores de suelo capacitivo bien calibrados.  
 (Evidencia: corrosión en sondas resistivas; ventajas de capacitivo; especificaciones DHT/BME).

## 3. Tipos de sensores: principios y características

### 3.1. Sensores de humedad de suelo

#### **A) Resistivos(YL-69 / FC-28)**

● Principio: miden la conductividad eléctrica entre dos electrodos insertados en el suelo (mayor humedad entonces más conductividad).

● Ventajas: muy baratos, fáciles de leer (analógico), disponibles en módulos con salida A0 y salida digital.

● Desventajas: corrosión de los electrodos cuando están permanentemente alimentados/insertados, vida útil corta; sensibles a la salinidad del suelo y variaciones químicas. Para alargar la vida se sugiere alimentar el sensor solo al tomar lectura (pulsado) o invertir polaridad periódicamente.

#### **B) Capacitivos (Capacitive Soil Moisture Sensor v1.2)**

● Principio: miden la capacitancia entre placas o trazos cuando se alteran por la humedad del sustrato; la presencia de agua cambia la permitividad dieléctrica.

● Ventajas: mayor durabilidad (sin electrodos expuestos), menos corrosión, lecturas más estables y menos afectadas por salinidad; adecuados para instalaciones semi-permanentes. Estudios recientes muestran buena relación costo/precisión frente a equipos TDR profesionales.

● Desventajas: requieren calibración para convertir lectura analógica (V) a contenido volumétrico de agua (VWC). Algunos clones baratos pueden tener problemas de calidad o envejecimiento; comprar de proveedores confiables.

### 3.2. Sensores de humedad ambiental (aire)

#### **A) DHT11 / DHT22 (AM2302)**

● DHT11: rango ~20–80 %RH, precisión ≈ ±5 %RH, económico; muestreo lento.

● DHT22 / AM2302: rango 0–100 %RH (o amplio), mejor precisión ≈ ±2–5 %RH según fuente, mayor resolución; más adecuado que DHT11 si necesitas mejor lectura. Ambos usan protocolo digital de un solo pin y requieren lectura no más de 0.5–1 Hz.

#### **B) Sensores de precisión I²C/SPI (BME280, SHT31/SHT3x, SI7021)**

● BME280 (Bosch): mide humedad, temperatura y presión; precisión humedad ~±3 %RH típ. Soporta I²C y SPI, bajo consumo y buena estabilidad a largo plazo. Ideal si quieres también presión (altitud/meteorología).

● SHT31 / SHT3x (Sensirion): sensores de alta precisión para RH y temperatura, linealidad y estabilidad superiores a DHT; comunicación I²C y mejores prestaciones térmicas. (Usados en aplicaciones profesionales).

## 4. Especificaciones útiles y comparativa resumida

*Nota:* los valores exactos dependen del fabricante / módulo. Aquí se muestran rangos típicos basados en datasheets y guías.

### ● YL-69 / FC-28 (resistivo)

○ Salida: analógica (0–Vcc).

○ Vida útil: baja si se deja alimentado y en contacto permanente (corrosión).

### ● Capacitive Soil Moisture Sensor v1.2

○ Salida: analógica (AOUT) — generalmente 0–3.3/0–5V según módulo.

○ Durabilidad: alta relativa; menos corrosión.

### ● DHT11

○ Humedad: 20–80 %RH, ±5 %RH.

○ Tiempo lectura: ≤1 Hz.

### ● DHT22 / AM2302

○ Humedad: 0–100 %RH, precisión ±2–5 %RH (mejor que DHT11), -40–80°C.

### ● BME280

○ Humedad: precisión típica ±3 %RH; I²C (0x76/0x77) o SPI; bajo consumo; alta estabilidad.

### ● SHT31 / SHT3x / SI7021

○ Humedad: típicamente ±2 %RH (SHT31), buena estabilidad y respuesta.

# 5. Evidencias de Código

## ID 1.1.2 — Conectar y programar sensor de humedad

Archivo: sensor\_humedad/sensor\_humedad.ino

// Conexión y programación de sensor de humedad

int pinHumedad = 34; // Pin analógico en ESP32

int valorHumedad = 0;

void setup() {

Serial.begin(115200);

}

void loop() {

// Leer valor analógico (0 - 4095 en ESP32)

valorHumedad = analogRead(pinHumedad);

// Convertir a porcentaje (aproximado, depende de calibración)

int humedadPorc = map(valorHumedad, 4095, 0, 0, 100);

Serial.print("Valor crudo: ");

Serial.print(valorHumedad);

Serial.print(" | Humedad: ");

Serial.print(humedadPorc);

Serial.println("%");

delay(1000);

}

## ID 1.1.3— Diseñar la interfaz de usuario para mostrar la lectura de humedad

Archivo: components/login.js

import tkinter as tk  **# Importa la librería Tkinter para crear interfaces gráficas**

def interfaz\_humedad():

root = tk.Tk()  **# Crea la ventana principal**

root.title("Lectura de Humedad") **# Define el título de la ventana**

root.geometry("400x200") **# Establece tamaño de la ventana (ancho x alto)**

root.configure(bg="#eaf7ea") **# Cambia el color de fondo a verde claro**

**# Etiqueta del título principal**

titulo = tk.Label(

root,

text="Monitor de Humedad", **# Texto que se muestra**

font=("Arial", 16, "bold"), **# Fuente: Arial, tamaño 16, negrita**

bg="#eaf7ea", **# Fondo igual al de la ventana**

fg="darkblue" **# Color del texto: azul oscuro**

)

titulo.pack(pady=10) **# Coloca la etiqueta con separación vertical de 10 píxeles**

**# Etiqueta que mostrará la lectura de la humedad**

label\_hum = tk.Label(

root,

text="Humedad: -- %", **# Texto inicial**

font=("Arial", 14), **# Fuente Arial, tamaño 14**

bg="#eaf7ea", **# Fondo verde claro**

fg="darkgreen" **# Texto en verde oscuro**

)

label\_hum.pack(pady=20) **# Coloca la etiqueta con separación vertical de 20 píxeles**

**# Botón para actualizar (por ahora no tiene lógica asociada)**

boton = tk.Button(

root,

text="Actualizar", **# Texto del botón**

font=("Arial", 12), **# Fuente Arial, tamaño 12**

bg="#cce5ff",  **# Fondo celeste**

relief="raised" **# Estilo de botón elevado**

)

boton.pack(pady=10) **# Coloca el botón con separación de 10 píxeles**

root.mainloop() **# Mantiene la ventana abierta esperando interacción del usuario**

**# Ejecuta la función solo si se corre este archivo directamente**

if \_\_name\_\_ == "\_\_main\_\_":

interfaz\_humedad()

Descripción del código: El presente fragmento de código en Python utiliza la librería Tkinter para diseñar una interfaz gráfica que muestre la lectura de humedad.

Primero se importa tkinter con el alias tk. Luego, dentro de la función interfaz\_humedad(), se crea la ventana principal con tk.Tk(), a la que se le asigna un título, dimensiones y un color de fondo.

Se añade un título principal mediante un Label con el texto Monitor de Humedad, configurado con fuente Arial en negrita y color azul oscuro, que se coloca con un margen superior para darle espacio visual. Después, se crea otro Label que muestra la humedad inicial con el texto Humedad: -- %, usando un tamaño de fuente más grande y color verde oscuro, pensado para representar la lectura del sensor de humedad.

A continuación, se incorpora un botón con la etiqueta Actualizar, diseñado con fuente mediana, fondo azul claro y un relieve para dar apariencia de botón real, aunque en esta versión aún no ejecuta ninguna acción, simplemente aparece en la ventana.

Todos los elementos (Label y Button) se organizan en la interfaz usando el gestor de geometría .pack(), que permite posicionarlos uno debajo del otro con márgenes verticales (pady). Finalmente, la instrucción root.mainloop() mantiene la ventana activa esperando interacciones del usuario, evitando que se cierre automáticamente.

Para ejecutar la aplicación, se utiliza la estructura if \_\_name\_\_ == "\_\_main\_\_": interfaz\_humedad(), lo que asegura que la interfaz se muestre solo si el archivo se corre directamente y no cuando se importe como módulo en otro programa.

En resumen, este código construye una ventana simple con un título, un indicador de humedad y un botón de actualización, sirviendo como base para simular o mostrar valores de humedad en un sistema más completo.

## ID-1.1.4— Integrar la lectura del sensor a la interfaz

|  |
| --- |
| #include <DHT.h>  #include <LiquidCrystal.h>  // Configuración del sensor DHT  #define DHT\_PIN 2  #define DHT\_TYPE DHT22 // Cambia a DHT11 si usas ese modelo  // Configuración de la pantalla LCD (RS, E, D4, D5, D6, D7)  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Crear objeto DHT  DHT dht(DHT\_PIN, DHT\_TYPE);  // Variables para almacenar lecturas  float temperatura;  float humedad;  float indiceCalor;  // Variables para control de actualización  unsigned long ultimaLectura = 0;  const unsigned long intervaloLectura = 2000; // Leer cada 2 segundos  // Variables para animación  int frame = 0;  unsigned long ultimoFrame = 0;  void setup() {  // Inicializar comunicación serial  Serial.begin(9600);    // Inicializar LCD (16 columnas, 2 filas)  lcd.begin(16, 2);    // Inicializar sensor DHT  dht.begin();    // Mostrar pantalla de inicio  mostrarPantallaInicio();    Serial.println("=== SISTEMA DHT22 + LCD ===");  Serial.println("Esperando lecturas...");  }  void loop() {  unsigned long tiempoActual = millis();    // Leer sensor cada intervalo definido  if (tiempoActual - ultimaLectura >= intervaloLectura) {  leerSensor();  ultimaLectura = tiempoActual;  }    // Actualizar animación cada 500ms  if (tiempoActual - ultimoFrame >= 500) {  actualizarAnimacion();  ultimoFrame = tiempoActual;  }    // Mostrar datos en LCD  mostrarDatosLCD();  }  void leerSensor() {  // Leer humedad y temperatura  humedad = dht.readHumidity();  temperatura = dht.readTemperature(); // Leer en Celsius    // Verificar si la lectura fue exitosa  if (isnan(humedad) || isnan(temperatura)) {  Serial.println("Error: No se pudo leer el sensor DHT");    // Mostrar error en LCD  lcd.clear();  lcd.setCursor(0, 0);  lcd.print("Error sensor DHT");  lcd.setCursor(0, 1);  lcd.print("Verificar conexion");  return;  }    // Calcular índice de calor  indiceCalor = dht.computeHeatIndex(temperatura, humedad, false);    // Mostrar datos por serial  mostrarDatosSerial();  }  void mostrarDatosLCD() {  lcd.clear();    // Línea 1: Temperatura  lcd.setCursor(0, 0);  lcd.print("Temp: ");  lcd.print(temperatura, 1);  lcd.print((char)223); // Símbolo de grados  lcd.print("C");    // Icono de temperatura según valor  lcd.setCursor(14, 0);  if (temperatura < 15) {  lcd.print("❄"); // Frío  } else if (temperatura >= 15 && temperatura <= 28) {  lcd.print("☀"); // Normal  } else {  lcd.print("🔥"); // Caliente  }    // Línea 2: Humedad  lcd.setCursor(0, 1);  lcd.print("Humedad: ");  lcd.print(humedad, 0);  lcd.print("%");    // Icono de humedad según valor  lcd.setCursor(14, 1);  if (humedad < 30) {  lcd.print("☁"); // Seco  } else if (humedad >= 30 && humedad <= 70) {  lcd.print("⛅"); // Normal  } else {  lcd.print("💧"); // Húmedo  }  }  void mostrarDatosSerial() {  Serial.print("T: ");  Serial.print(temperatura, 1);  Serial.print("°C | H: ");  Serial.print(humedad, 0);  Serial.print("% | IC: ");  Serial.print(indiceCalor, 1);  Serial.print("°C | Estado: ");    if (humedad < 30) {  Serial.println("Muy seco");  } else if (humedad >= 30 && humedad <= 60) {  Serial.println("Confortable");  } else {  Serial.println("Húmedo");  }  }  void mostrarPantallaInicio() {  lcd.setCursor(0, 0);  lcd.print(" SISTEMA DHT22 ");  lcd.setCursor(0, 1);  lcd.print(" INICIANDO... ");  delay(3000);  lcd.clear();  }  void actualizarAnimacion() {  // Esta función puede usarse para animaciones simples  frame = (frame + 1) % 4;  } |

El código comienza incluyendo las librerías necesarias para el sensor DHT22 y la pantalla LCD, definiendo los pines de conexión y creando los objetos correspondientes. En la función setup(), se inicializa la comunicación serial a 9600 baudios, se configura la pantalla LCD como display de 16x2 caracteres y se inicia el sensor DHT22. Posteriormente, se muestra una pantalla de bienvenida en el LCD con el mensaje "SISTEMA DHT22" y "INICIANDO..." durante 3 segundos, mientras que por el monitor serial se imprime un mensaje indicando que el sistema está listo y esperando lecturas. Todo este proceso de inicialización asegura que todos los componentes hardware estén correctamente configurados antes de comenzar el ciclo principal de medición.

El núcleo del programa se ejecuta en la función loop(), que utiliza temporizadores no bloqueantes para gestionar las diferentes tareas. Cada 2 segundos (definido por intervaloLectura), el sistema realiza una lectura del sensor DHT22, obteniendo los valores de temperatura en grados Celsius y humedad relativa en porcentaje. Si la lectura es exitosa, calcula automáticamente el índice de calor (sensación térmica) utilizando la función computeHeatIndex(). En caso de error en la lectura, el sistema detecta valores NaN (Not a Number) y muestra mensajes de error tanto en el LCD como en el monitor serial, indicando que se debe verificar la conexión del sensor, lo que garantiza robustez ante fallos hardware.

La presentación de datos se realiza de forma simultánea en la pantalla LCD y el monitor serial. En el LCD, la primera línea muestra la temperatura con un decimal seguida del símbolo de grados Celsius, mientras que la segunda línea presenta la humedad como valor porcentual entero. Adicionalmente, se incluyen iconos visuales que cambian dinámicamente según los valores medidos: para temperatura usa (frío), (normal) o (caliente), y para humedad utiliza (seco), (normal) o (húmedo). Paralelamente, el monitor serial recibe toda la información en formato texto, incluyendo temperatura, humedad, índice de calor y un descriptor del estado ambiental, facilitando el registro y análisis histórico de los datos mediante herramientas externas.

## ID-1.1.5— Probar la visualización de datos en tiempo real

## ID-1.1.6— Implementar alerta en caso de fallo del sensor

|  |
| --- |
| #include <DHT.h>  #include <LiquidCrystal.h>  // Configuración del sensor DHT  #define DHT\_PIN 2  #define DHT\_TYPE DHT22  // Configuración LCD  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Pin para buzzer de alerta  #define BUZZER\_PIN 8  DHT dht(DHT\_PIN, DHT\_TYPE);  float temperatura, humedad;  int fallosConsecutivos = 0;  bool alertaActiva = false;  void setup() {  Serial.begin(9600);  lcd.begin(16, 2);  pinMode(BUZZER\_PIN, OUTPUT);  dht.begin();    lcd.print("Iniciando...");  delay(2000);  }  void loop() {  // Leer sensor cada 2 segundos  if (leerSensor()) {  // Lectura exitosa  mostrarDatosNormal();  alertaActiva = false;  noTone(BUZZER\_PIN); // Apagar alerta  } else {  // Error en lectura  fallosConsecutivos++;  mostrarAlertaError();  activarAlertaSonora();  alertaActiva = true;  }    delay(2000);  }  bool leerSensor() {  humedad = dht.readHumidity();  temperatura = dht.readTemperature();    // Si la lectura es exitosa, resetear contador  if (!isnan(humedad) && !isnan(temperatura)) {  fallosConsecutivos = 0;  return true;  }  return false;  }  void mostrarDatosNormal() {  lcd.clear();  lcd.setCursor(0, 0);  lcd.print("T:");  lcd.print(temperatura, 1);  lcd.print("C H:");  lcd.print(humedad, 0);  lcd.print("%");    lcd.setCursor(0, 1);  lcd.print("Sensor: OK");  }  void mostrarAlertaError() {  lcd.clear();  lcd.setCursor(0, 0);  lcd.print("ALERTA!");  lcd.setCursor(0, 1);  lcd.print("Error sensor DHT");    Serial.print("Alerta! Fallo #");  Serial.println(fallosConsecutivos);  }  void activarAlertaSonora() {  // Tono de alerta intermitente  tone(BUZZER\_PIN, 1000, 500);  } |

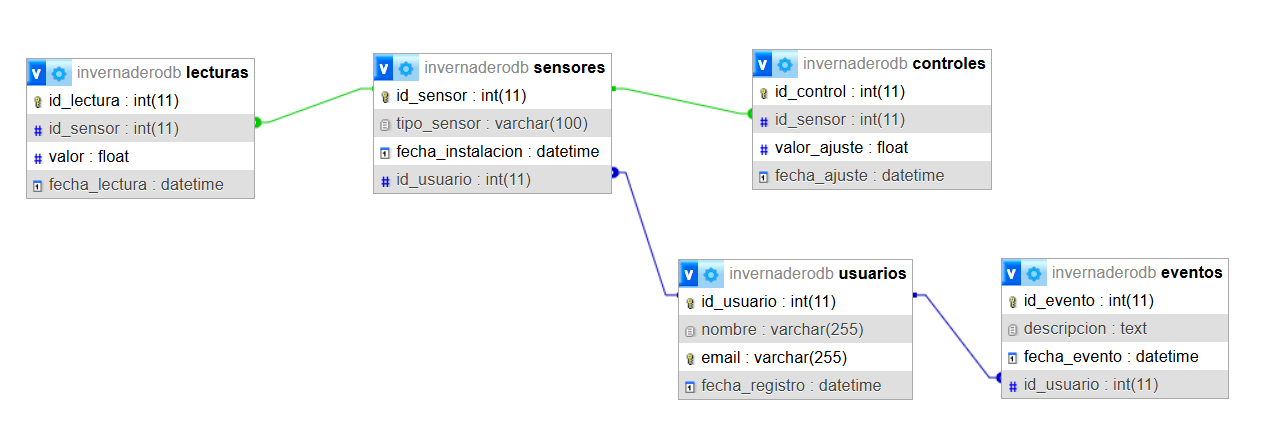
El código comienza incluyendo las librerías necesarias para el sensor DHT22 y la pantalla LCD, definiendo los pines de conexión específicos para cada componente. En la función setup(), se inicializa la comunicación serial a 9600 baudios para el monitoreo, se configura la pantalla LCD con 16 columnas y 2 filas, se establece el pin del buzzer como salida y se inicia el sensor DHT22. El sistema muestra un mensaje de "Iniciando..." en la pantalla durante 2 segundos, indicando que todos los componentes se están preparando para comenzar las lecturas. Esta fase asegura que tanto el hardware como el software estén correctamente configurados antes de iniciar el ciclo principal de medición.

En el bucle principal loop(), el sistema realiza lecturas del sensor cada 2 segundos. La función leerSensor() obtiene los valores de humedad y temperatura, verificando si son números válidos mediante isnan(). Si ambas lecturas son correctas, el contador de fallos consecutivos se reinicia a cero y el sistema continúa en modo normal. Sin embargo, si alguna lectura falla (devuelve NaN), el contador de fallos se incrementa y se activa inmediatamente el protocolo de alerta. Esta detección automática permite identificar problemas de conexión, fallos del sensor o condiciones ambientales que impidan lecturas precisas.

Cuando se detecta un fallo, el sistema activa múltiples formas de alerta simultáneamente: ¡en la pantalla LCD muestra “ALERTA!" y "Error sensor DHT", por el monitor serial envía un mensaje indicando el número de fallo consecutivo, y activa el buzzer con un tono de 1000 Hz durante 500 milisegundos. En contraste, cuando el sensor funciona correctamente, la pantalla muestra temperatura y humedad en formato claro ("T: XX.XC H: XX%") con el mensaje "Sensor: OK", y se silencia el buzzer. Este enfoque dual proporciona una respuesta inmediata ante fallos mientras mantiene una interfaz limpia durante el funcionamiento normal.

## Historia de Usuario 1.2: Registro histórico

**1.2.1 Configurar base de datos local para almacenar lecturas diarias 4**

**Base de Datos:**

**Revisión de tu modelo**

1. **Usuarios**
   * Correcto: identificación única, email único y fecha de registro.
   * Puedes agregar campo rol o estado (activo/inactivo) si quieres diferenciar administradores de usuarios comunes.
2. **Sensores**
   * Bien vinculado a un usuario (propietario/instalador).
   * Podrías agregar campos como:
   * ubicacion VARCHAR(255),
   * unidad\_medida VARCHAR(50)

**para saber dónde está instalado el sensor y qué mide (°C, %, ppm, etc.).**

1. **Lecturas**
   * Correcto para lecturas diarias.
   * Si planeas guardar lecturas muy frecuentes (cada segundo o minuto), conviene usar BIGINT para id\_lectura y considerar particionar la tabla por fecha.
2. **Controles**
   * Está bien, sirve para registrar ajustes (ej: apertura de válvulas, ventiladores, etc.).
   * Tal vez convenga relacionar también con Eventos si quieres trazar qué ajuste fue producto de un evento (alarma).
3. **Eventos**
   * Correcto, ya ligado a usuario.
   * Puedes agregar severidad o tipo de evento:
   * tipo\_evento ENUM('Alerta','Error','Mantenimiento','Info') NOT NULL

**Ejemplo de consultas útiles**

1. **Obtener lecturas de un sensor en la última semana:**

SELECT fecha\_lectura, valor

FROM Lecturas

WHERE id\_sensor = 1

AND fecha\_lectura >= NOW() - INTERVAL 7 DAY

ORDER BY fecha\_lectura;

1. **Promedio diario de temperatura por sensor:**

SELECT DATE(fecha\_lectura) AS dia, AVG(valor) AS promedio

FROM Lecturas

WHERE id\_sensor = 1

GROUP BY DATE(fecha\_lectura)

ORDER BY dia;

1. **Últimos eventos de un usuario:**

SELECT descripcion, fecha\_evento

FROM Eventos

WHERE id\_usuario = 2

ORDER BY fecha\_evento DESC

LIMIT 10;

## Historia de Usuario 2.1: Riego automático por humedad

**ID 2.1.1 Investigar y adquirir la bomba de riego**

1. Tipos de bombas recomendadas

* Mini bomba sumergible DC (3V–12V)
  + Ideal para prototipos o pequeños invernaderos.
  + Bajo consumo de energía.
  + Caudal aproximado: 80–120 L/h.
  + Precio referencial: S/ 25 – 40.
* Bomba peristáltica DC (5V–12V)
  + Diseñada para riego por goteo o dosificación de nutrientes.
  + Permite un control preciso del flujo de agua.
  + Precio referencial: S/ 45 – 70.
* Bomba sumergible DC de mayor capacidad (12V–24V)
  + Adecuada para invernaderos medianos o grandes.
  + Caudal aproximado: 240–800 L/h.
  + Requiere fuente de alimentación independiente y módulo relé.
  + Precio referencial: S/ 60 – 120.

2. Componentes complementarios necesarios

* Módulo relé 5V o MOSFET: para controlar la bomba desde el Arduino.
  + Precio: S/ 10 – 20.
* Fuente de alimentación externa (según el voltaje de la bomba).
  + Precio: S/ 20 – 40.
* Sensor de humedad de suelo: para medir el nivel de humedad y automatizar el riego.
  + Precio: S/ 10 – 25.
* Mangueras y válvulas de riego: costo variable (S/ 15 – 30).

3. Ejemplo de funcionamiento del sistema

1. El sensor de humedad detecta sequedad en el suelo.
2. Arduino procesa la señal recibida.
3. Se activa el módulo relé conectado al Arduino.
4. El relé enciende la bomba de agua.
5. El riego se detiene cuando la humedad del suelo alcanza el nivel deseado.

4. Opciones de adquisición en Perú

* Mercado Libre Perú
  + “Mini bomba sumergible DC 5V–12V para Arduino”: S/ 25 – 40.
  + “Bomba de agua peristáltica DC 12V”: S/ 45 – 70.
  + “Kit de riego automático (bomba + sensor + relé)”: S/ 60 – 90.
* Tiendas de electrónica (Promelsa, Infotronica, Steren, distribuidores locales).
  + Bombas DC sumergibles y accesorios.
  + Precios similares a Mercado Libre.
* AliExpress (importación)
  + Bombas de 5V, 6V y 12V desde S/ 12 – 25.
  + Kits completos desde S/ 40 – 70 (con tiempos de entrega más largos).

**Recomendación para el proyecto**

* Prototipo o pruebas en laboratorio: Mini bomba sumergible DC 12V + módulo relé + sensor de humedad (costo total aproximado S/ 60 – 80).
* Invernadero en operación real: Bomba DC 12V de mayor caudal + sistema de riego por goteo + relé + fuente de 2A (costo total aproximado S/ 100 – 160).

## Historia de Usuario 2.3: Opción manual de emergencia

**2.3.1 Seleccionar y adquirir el botón físico para la activación manual**

Se adquirió tres unidades del botón físico (push button) para el proyecto con Arduino. Este componente permitirá interactuar manualmente con el sistema, habilitando funciones como encendido/apagado, reinicio o control de eventos según la programación.



## Historia de Usuario 3.1: Lectura en tiempo real(Temperatura)

**3.1.1 Investigar y adquirir el sensor de temperatura**

Un sensor es un dispositivo que detecta cambios o variaciones físicas del entorno como temperatura, luz, humedad o movimiento y las convierte en señales eléctricas que pueden ser interpretadas por un microcontrolador, como el Arduino o el ESP32.  
En esencia, los sensores permiten que un sistema electrónico “perciba” su entorno y reaccione en consecuencia, convirtiéndose en una parte fundamental para el desarrollo de proyectos automatizados, sistemas de monitoreo e Internet de las Cosas (IoT).

Estos sensores son los que se averiguaron, ideales o con referencia hacia nuestro proyecto:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sensor | Tipo de salida | Ventajas | Desventajas / consideraciones | Precio estimado\* |
| DHT22 (AM2302) | Digital (bus de un solo cable) | Buena precisión (~±0,5 °C), rango amplio, lectura de humedad también | No muy alta frecuencia (solo ~1 medición cada 2 s) | S/ 25 a S/ 35 aprox. Ej: Electromanía lo ofrece por S/ 25. [Electromania](https://www.electromania.pe/producto/sensor-de-temperatura-y-humedad-dht22/?utm_source=chatgpt.com); Tesla Electronic lo tiene a S/ 28 [Tesla Electronic EIRL](https://www.teslaelectronic.com.pe/producto/sensor-dht22-temperatura-y-humedad/?srsltid=AfmBOopdSE3GJwLLtIJwtRU921mlIiKChY3Naz7mrqVAFWybNWVGjNf8&utm_source=chatgpt.com) |
| LM35 | Analógica (voltaje proporcional a temperatura) | Simplicidad, fácil de leer con analogRead | Menos preciso en rangos extremos, requiere calibración y filtrado de ruido | En tiendas de electrónica, podría encontrarse desde unos S/ 10–15 (según tienda local) |
| DS18B20 | Digital (1-Wire) | Alta resolución, bastante usado, tolera cables largos | Necesita librería 1-Wire, a veces requiere pull-up | En algunas tiendas se ve a precios bajos (menos de S/ 20) |

## Historia de Usuario 4.1: Ventilacion automatica

**4.1.1 Investigar y adquirir componentes de control de movimiento de puerta (Motor DC, Driver L298N).**

**1. Introducción**

En un sistema de riego automatizado, el control del flujo de agua es un factor fundamental para garantizar la eficiencia en la distribución del recurso hídrico. Una de las soluciones empleadas para lograr este control es la automatización del movimiento de puertas o compuertas, que permiten abrir o cerrar el paso del agua de manera controlada y precisa.

Para este propósito, se requiere de un conjunto de componentes que transformen la energía eléctrica en movimiento mecánico, permitiendo el accionamiento automático de la compuerta. Entre los elementos más utilizados para esta función destacan el motor de corriente continua (DC) y el módulo controlador de motor (driver L298N), que en conjunto posibilitan el control del sentido y la velocidad del movimiento.

**2. Motor DC: Principio de funcionamiento y selección**

El motor de corriente continua (DC) es un dispositivo electromecánico que convierte energía eléctrica en movimiento rotacional. Su funcionamiento se basa en la interacción de un campo magnético y una corriente eléctrica, generando un torque proporcional a la intensidad de la corriente.

En el caso del sistema de riego automatizado, el motor DC es responsable de accionar la compuerta que regula el caudal de agua en los canales o reservorios. Para lograr un movimiento controlado, se utiliza un motor DC con reductora (geared DC motor), que reduce la velocidad de rotación y aumenta el torque disponible, permitiendo mover compuertas de manera lenta y estable.

**Criterios de selección:**

* **Tensión de operación:** 12V o 24V DC (según la fuente de energía del sistema).
* **Velocidad de rotación:** entre 10 y 100 RPM, para garantizar un movimiento suave y seguro.
* **Torque:** superior a 30 kg·cm, dependiendo del peso de la compuerta y la fuerza requerida.
* **Consumo de corriente:** de 1 a 5 A, acorde con la capacidad del driver seleccionado.
* **Tipo de reductora:** planetaria o de tornillo sin fin (worm gear), por su eficiencia y capacidad de autobloqueo.

**Modelo propuesto:**

Motor DC con reductora 12V, 60W, 100 RPM tipo worm gear, adecuado para la apertura y cierre de compuertas pequeñas o medianas en sistemas de riego automatizado. Este tipo de motor ofrece un excelente equilibrio entre potencia, torque y facilidad de control.

**3. Módulo Driver L298N**

El L298N es un módulo electrónico basado en un circuito integrado de doble puente H, diseñado para controlar motores DC o motores paso a paso mediante señales digitales de control. En este proyecto, el L298N permite regular la dirección y velocidad de giro del motor DC, en función de las órdenes enviadas desde un microcontrolador (por ejemplo, Arduino UNO o ESP32).

**Características técnicas principales:**

* Tensión de alimentación del motor: 5V a 35V DC.
* Corriente máxima de salida: 2A por canal.
* Control de sentido de giro mediante entradas lógicas (IN1–IN4).
* Control de velocidad a través de modulación por ancho de pulso (PWM).
* Incluye disipador de calor y pines de habilitación (ENA/ENB).

El módulo L298N es ampliamente utilizado en proyectos de automatización agrícola y sistemas embebidos por su bajo costo, facilidad de conexión y compatibilidad con plataformas educativas. Para aplicaciones que demanden mayor corriente, se recomienda considerar controladores de mayor capacidad como el BTS7960 (43A).

**4. Integración del sistema de control**

En el sistema de riego automatizado, el motor DC y el driver L298N se integran con un microcontrolador y sensores que determinan las condiciones de apertura o cierre de la compuerta según la humedad del suelo, nivel del agua o programación horaria.

|  |  |  |
| --- | --- | --- |
| **Componente** | **Función principal** | **Parámetros relevantes** |
| Motor DC con reductora | Generar movimiento de apertura/cierre de la compuerta. | 12V, 60W, 100 RPM, torque ≥ 30 kg·cm |
| Driver L298N | Controlar sentido y velocidad del motor DC. | 5–35V, 2A por canal |
| Microcontrolador (Arduino UNO / ESP32) | Procesar señales de sensores y enviar comandos al driver. | Compatible con PWM y L298N |
| Fuente de alimentación | Suministrar energía estable al sistema. | 12V DC, 5–10A |
| Sensores de nivel o humedad | Activar o detener el movimiento según condiciones de riego. | Sensor capacitivo o resistivo |
| Fin de carrera | Detectar límites de apertura/cierre de la compuerta. | Mecánico o magnético |

El sistema puede programarse para que la compuerta se abra automáticamente cuando el nivel de humedad del suelo sea bajo o el nivel del agua sea suficiente, cerrándose cuando las condiciones de riego sean óptimas.

**5. Criterios de adquisición**

Durante la adquisición de los componentes, se consideraron los siguientes criterios técnicos:

* **Compatibilidad eléctrica y lógica** con el microcontrolador y sensores del sistema.
* **Disponibilidad en el mercado local o plataformas en línea** (Amazon, MercadoLibre, AliExpress).
* **Relación costo–beneficio**, priorizando componentes de bajo consumo y mantenimiento mínimo.
* **Resistencia y durabilidad** ante condiciones ambientales (temperatura, humedad).

**6. Conclusión**

La selección del motor DC con reductora y el driver L298N resulta idónea para el control automatizado de compuertas en un sistema de riego inteligente. Estos componentes permiten un control preciso del movimiento, facilitando la regulación del flujo de agua con bajo consumo energético y alta fiabilidad.

Asimismo, su integración con un microcontrolador y sensores ambientales posibilita la automatización completa del sistema, contribuyendo a una gestión más eficiente y sostenible del recurso hídrico en entornos agrícolas.

**Historia de Usuario 4.2: Calefacción automática**

**3.1.1 Investigar y adquirir el Calefactor y sus componentes de control.**

**1. Introducción**

En el marco del proyecto de riego automatizado, la incorporación de un calefactor tiene como objetivo mantener condiciones térmicas adecuadas en el sistema, especialmente en zonas donde las bajas temperaturas pueden afectar el funcionamiento de los sensores, el flujo del agua o el crecimiento de las plantas.

El calefactor permite elevar o estabilizar la temperatura en áreas específicas del sistema, como depósitos de agua, invernaderos o conducciones, asegurando la eficiencia del riego y la protección de los componentes electrónicos. Su funcionamiento requiere de un sistema de control térmico, compuesto por sensores de temperatura, módulos de potencia y controladores lógicos que regulan su activación.

**2. Calefactor: Principio de funcionamiento y tipos**

Un calefactor es un dispositivo eléctrico que convierte la energía eléctrica en energía térmica mediante el efecto Joule, el cual establece que cuando una corriente eléctrica atraviesa un material resistivo, se produce calor.

Existen diversos tipos de calefactores utilizados en sistemas automatizados:

|  |  |  |
| --- | --- | --- |
| **Tipo de calefactor** | **Aplicación principal** | **Características** |
| **Resistencia calefactora (cerámica o metálica)** | Calentamiento de agua o aire. | Eficiente, económica, fácil de controlar. |
| **Calefactor tipo cartucho** | Aplicaciones industriales y de precisión. | Alta densidad de potencia, tamaño compacto. |
| **Banda calefactora o flexible** | Superficies cilíndricas o depósitos. | Distribución uniforme del calor. |
| **Calefactor PTC (coeficiente positivo de temperatura)** | Control automático de temperatura. | Autolimitante, segura y duradera. |

**Tipo seleccionado:**

Para el presente proyecto, se selecciona una resistencia calefactora tipo PTC de 12V DC, adecuada para aplicaciones de baja potencia en sistemas de riego automatizado. Su diseño compacto y capacidad de autorregulación la hacen ideal para mantener la temperatura del agua o del entorno de sensores sin riesgo de sobrecalentamiento.

**3. Componentes de control del calefactor**

El control del calefactor requiere de un sistema que permita regular su encendido y apagado según la temperatura deseada. Este sistema se compone de los siguientes elementos:

|  |  |  |
| --- | --- | --- |
| **Componente** | **Función principal** | **Ejemplo de modelo** |
| **Sensor de temperatura** | Detectar la temperatura del agua o ambiente. | Sensor DHT22 o DS18B20. |
| **Módulo de control (microcontrolador)** | Procesar las señales del sensor y decidir cuándo activar el calefactor. | Arduino UNO / ESP32. |
| **Módulo de potencia (relevador o MOSFET)** | Permitir el paso de corriente al calefactor bajo control del microcontrolador. | Módulo relé 10A o MOSFET IRF520. |
| **Fuente de alimentación** | Proporcionar energía estable al sistema de calefacción. | 12V DC / 10A. |

**Principio de control:**

El sensor de temperatura monitorea el entorno y envía la lectura al microcontrolador. Este compara la lectura con un umbral predefinido (por ejemplo, 20 °C).

* Si la temperatura es inferior al valor establecido, el microcontrolador activa el calefactor mediante el módulo relé.
* Cuando se alcanza la temperatura óptima, el sistema interrumpe la corriente, manteniendo así un equilibrio térmico eficiente.

Este principio de control puede ser binario (ON/OFF) o, en versiones más avanzadas, modulado mediante PWM o control PID para un ajuste más preciso de la temperatura.

**4. Integración del sistema de calefacción**

En el sistema de riego automatizado, el calefactor se integra con el resto de los módulos de control ambiental. Su función es prevenir la condensación, el congelamiento o la baja temperatura del agua en los tanques y conducciones, garantizando el funcionamiento óptimo del sistema.

|  |  |  |
| --- | --- | --- |
| **Elemento** | **Descripción** | **Parámetros técnicos** |
| Calefactor PTC 12V | Elemento resistivo autolimitante para elevar la temperatura del agua o aire. | 12V DC, 100 W, autorregulación 70 °C |
| Sensor DS18B20 | Sensor de temperatura digital de alta precisión. | Rango -55 °C a +125 °C, precisión ±0.5 °C |
| Módulo relé 10A | Actuador de potencia controlado por microcontrolador. | Capacidad de conmutación: 10A / 250V AC o 10A / 30V DC |
| Microcontrolador Arduino UNO | Unidad lógica de procesamiento y control. | Entradas digitales/analógicas, comunicación serial |
| Fuente 12V DC | Fuente de poder del sistema calefactor y control. | 12V, 10A |

**5. Criterios de adquisición**

Para la selección y compra de los componentes, se consideraron los siguientes criterios técnicos y logísticos:

* **Compatibilidad eléctrica:** Todos los componentes funcionan a 12V DC, garantizando integración con el sistema de riego.
* **Seguridad térmica:** Selección de calefactor PTC por su capacidad de autorregulación y protección contra sobrecalentamiento.
* **Eficiencia energética:** Baja demanda de corriente y rápida respuesta térmica.
* **Disponibilidad comercial:** Componentes adquiribles en proveedores locales o en línea (Amazon, MercadoLibre, AliExpress).
* **Costo accesible:** Adecuado para proyectos académicos y prototipos de ingeniería.

**Ejemplos de componentes adquiribles:**

* **Calefactor PTC 12V DC, 100 W.**
* **Sensor de temperatura DS18B20 impermeable.**
* **Módulo relé 1 canal de 10A.**
* **Fuente de poder 12V/10A.**

**6. Conclusión**

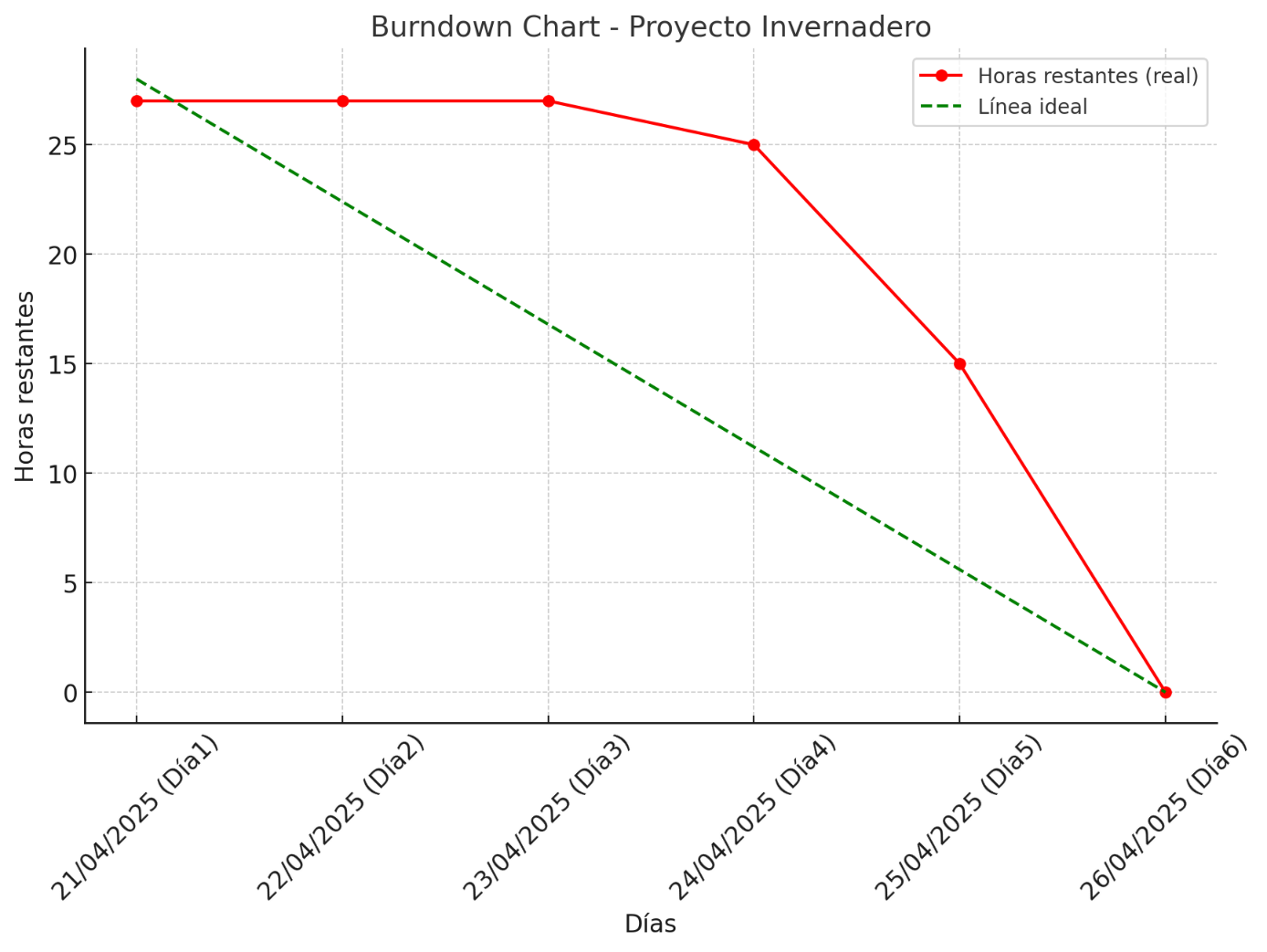
La integración de un calefactor controlado electrónicamente dentro del sistema de riego automatizado permite mantener condiciones térmicas estables y óptimas para el funcionamiento del equipo y la eficiencia del riego.

La elección del calefactor PTC 12V junto con un sistema de control basado en sensor de temperatura, módulo relé y microcontrolador proporciona una solución segura, eficiente y de bajo mantenimiento. Este sistema contribuye significativamente a la sostenibilidad y automatización inteligente de los procesos agrícolas, al reducir pérdidas energéticas y garantizar el correcto desempeño del sistema en diversas condiciones ambientales.

# 5. Burndown Chart

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Backlog ID | Historia de Usuario | Tiempo Estimado | 21/04/2025 | 22/04/2025 | 23/04/2025 | 24/04/2025 | 25/04/2025 | 26/04/2025 |
| **Dia1** | **Dia2** | **Dia3** | **Dia4** | **Dia5** | **Dia6** |
| HU-1.1 | Visualización en tiempo real | 28 hrs. | 3 | 5 | 5 | 7 | 7 | 1 |
| HU-1.2 | Registro histórico | 36 hrs. | 0 | 0 | 0 | 4 | 2 | 2 |
| Tiempo de trabajo | | 64 hrs. | 3 | 5 | 5 | 11 | 12 | 3 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Backlog ID | Historia de Usuario | Tiempo Estimado | 21/09/2025 | 22/09/2025 | 23/09/2025 | 24/09/2025 | 25/09/2025 | 26/09/2025 |
| **Dia1** | **Dia2** | **Dia3** | **Dia4** | **Dia5** | **Dia6** |
| HU-1.1.1 | Investigar sensores de humedad compatibles con Arduino | 4 hrs. | 1 | 0 | 0 | 0 | 0 | 3 |
| HU-1.1.2 | Conectar y programar sensor de humedad | 6 hrs. | 0 | 0 | 0 | 0 | 4 | 2 |
| 1.1.3 | Diseñar la interfaz de usuario para mostrar la lectura de humedad | 5 hrs. | 0 | 0 | 0 | 0 | 3 | 2 |
| 1.1.4 | Integrar la lectura del sensor a la interfaz | 6 hrs. | 0 | 0 | 0 | 2 | 2 | 2 |
| 1.1.5 | Probar la visualización de datos en tiempo real | 4 hrs. | 0 | 0 | 0 | 0 | 1 | 3 |
| 1.1.6 | Implementar alerta en caso de fallo del sensor | 3 hrs. | 0 | 0 | 0 | 0 | 0 | 3 |
| Tiempo de trabajo | | 28 hrs. | 1 | 0 | 0 | 2 | 10 | 15 |



# 6. Análisis del Sprint

* Se completaron todas las tareas planificadas:
  + Investigación de sensores de humedad compatibles con Arduino.
  + Conexión y programación del sensor de humedad.
  + Diseño de la interfaz de usuario para mostrar lecturas.
  + Integración de la lectura del sensor en la interfaz.
  + Pruebas de visualización de datos en tiempo real.
  + Implementación de una alerta en caso de fallo del sensor.
* El progreso fue constante y sin bloqueos significativos.
* Las decisiones técnicas sobre integración sensor–interfaz fueron efectivas para este primer Sprint.
* El código se modularizó y se documentó en comentarios para facilitar mantenimiento y futuras mejoras.
* Las estimaciones fueron precisas y se mantuvo un ritmo de trabajo estable.
* Los objetivos del Sprint se cumplieron satisfactoriamente, dejando la base preparada para nuevas funcionalidades.

# 7. Conclusiones

El Sprint 1 concluyó exitosamente con todas las funcionalidades básicas implementadas en torno al sensor de humedad. Se investigaron sensores compatibles con Arduino, se logró la conexión y programación del módulo, y se diseñó la interfaz de usuario para mostrar en tiempo real la lectura de humedad. Asimismo, se integró la lectura del sensor en la interfaz, se validó la visualización de datos en tiempo real y se implementó una alerta ante posibles fallos del sensor.

El equipo demostró capacidad de entrega puntual y una adecuada gestión de riesgos menores. Las métricas indican que la carga de trabajo fue bien distribuida y los objetivos del sprint se cumplieron de manera satisfactoria, quedando la base lista para futuras mejoras y ampliaciones del sistema.