Registro Diario de Avances – Sprint 1

**Fecha:24/09/2025**

**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 de la aplicación TaskManager que permita a los usuarios:

* Registrarse e iniciar sesión.
* Crear nuevas tareas.
* Marcar tareas como completadas.
* Almacenar los datos de manera persistente con localStorage.

# 3. Historias de Usuario y Tareas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| HU-1 | Visualización en tiempo real | 3 pts | 17/09/2025 | 26/09/2025 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

## 

# 4. Evidencias de Investigación de Sensores

**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. [how2electronics.com+1](https://how2electronics.com/interface-capacitive-soil-moisture-sensor-arduino/?utm_source=chatgpt.com)

● 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. [learn.adafruit.com+1](https://learn.adafruit.com/dht/overview?utm_source=chatgpt.com)

#### **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). [Bosch Sensortec+1](https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf?utm_source=chatgpt.com)

● 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). [Seeed Studio](https://www.seeedstudio.com/blog/2020/04/20/dht11-vs-dht22-am2302-which-temperature-humidity-sensor-should-you-use/?srsltid=AfmBOooAY8ZwcqwIGLyD4HtMnIrCW6HS98Y4R_MstCvaIPaiN6CH1l_1&utm_source=chatgpt.com)

## 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). [Random Nerd Tutorials+1](https://randomnerdtutorials.com/guide-for-soil-moisture-sensor-yl-69-or-hl-69-with-the-arduino/?utm_source=chatgpt.com)

### ● Capacitive Soil Moisture Sensor v1.2

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

○ Durabilidad: alta relativa; menos corrosión. [how2electronics.com+1](https://how2electronics.com/interface-capacitive-soil-moisture-sensor-arduino/?utm_source=chatgpt.com)

### ● DHT11

○ Humedad: 20–80 %RH, ±5 %RH.

○ Tiempo lectura: ≤1 Hz. [learn.adafruit.com](https://learn.adafruit.com/dht/overview?utm_source=chatgpt.com)

### ● DHT22 / AM2302

○ Humedad: 0–100 %RH, precisión ±2–5 %RH (mejor que DHT11), -40–80°C. [cdn.sparkfun.com](https://cdn.sparkfun.com/assets/f/7/d/9/c/DHT22.pdf?utm_source=chatgpt.com)

### ● BME280

○ Humedad: precisión típica ±3 %RH; I²C (0x76/0x77) o SPI; bajo consumo; alta estabilidad. [Bosch Sensortec+1](https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf?utm_source=chatgpt.com)

### ● SHT31 / SHT3x / SI7021

○ Humedad: típicamente ±2 %RH (SHT31), buena estabilidad y respuesta. [Seeed Studio](https://www.seeedstudio.com/blog/2020/04/20/dht11-vs-dht22-am2302-which-temperature-humidity-sensor-should-you-use/?srsltid=AfmBOooAY8ZwcqwIGLyD4HtMnIrCW6HS98Y4R_MstCvaIPaiN6CH1l_1&utm_source=chatgpt.com)

# 5. Evidencias de Código

## 4.1 Historia de Usuario 1 — 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);

}

## 4.2 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.

## 4.3 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.

## 4.4 Historia de Usuario 3 — Gestión de Tareas

Archivo: components/NuevaTarea.js

// Componente para agregar nuevas tareas

function NuevaTarea({ agregar }) {

    const [texto, setTexto] = useState("");

    // Agrega la tarea si hay texto ingresado

    const agregarTarea = () => {

      if (texto) {

        agregar(texto);

        setTexto(""); // Limpia el input después de agregar

      }

    };

    return (

      <div>

        <input value={texto} onChange={e => setTexto(e.target.value)} placeholder="Nueva tarea" />

        <button onClick={agregarTarea}>Agregar</button>

      </div>

    );

  }

Archivo: components/Tarea.js

// Componente que representa una tarea individual

function Tarea({ tarea, completar }) {

    return (

      <div>

        {/\* Muestra el texto con tachado si está completada \*/}

        <span style={{ textDecoration: tarea.completada ? "line-through" : "none" }}>

          {tarea.texto}

        </span>

        {/\* Botón para marcar como completada \*/}

        {!tarea.completada && <button onClick={completar}>Completar</button>}

      </div>

    );

  }

Archivo: components/App.js

// Componente principal de la aplicación

function App() {

    // Carga las tareas guardadas desde localStorage al iniciar

    const [tareas, setTareas] = useState(() => {

      const guardadas = localStorage.getItem("tareas");

      return guardadas ? JSON.parse(guardadas) : [];

    });

    // Guarda las tareas en localStorage cada vez que cambian

    useEffect(() => {

      localStorage.setItem("tareas", JSON.stringify(tareas));

    }, [tareas]);

    // Función para agregar una nueva tarea

    const agregar = texto => setTareas([...tareas, { texto, completada: false }]);

    // Función para marcar una tarea como completada

    const completar = index => {

      const nuevas = [...tareas];

      nuevas[index].completada = true;

      setTareas(nuevas);

    };

    return (

      <div>

        <NuevaTarea agregar={agregar} />

        {tareas.map((t, i) => (

          <Tarea key={i} tarea={t} completar={() => completar(i)} />

        ))}

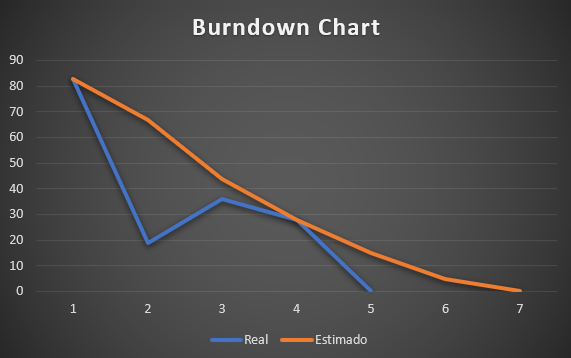
      </div>

    );

  }

# 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. | 8 | 8 | 5 | 3 | 3 | 1 |
| HU-1.2 | Registro histórico | 36 hrs. | 8 | 8 | 8 | 6 | 4 | 2 |
| HU-1.3 | Umbrales personalizables | 19 hrs. |  | 7 | 3 | 4 | 3 | 2 |
| Tiempo de trabajo | | 83 hrs. | 67 | 44 | 28 | 15 | 5 | 0 |



# 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.