

Universidad San Carlos de Guatemala
Facultad de Ingeniería -FIUSAC-
Escuela de Ciencias y Sistemas
Curso Arquitectura de Computadoras
y Ensambladores 2 - Sección: "B"



FIUSAC
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Fase 1 - PROYECTO ÚNICO -

**Sistema Inteligente de monitoreo ambiental para
Centros de Datos mediante IoT**

DOCUMENTACIÓN

Adler Alejandro Pérez Asensio	202200329
Andrés Alejandro Quezada Cabrera	202200174
Angel Samuel González Velásquez	202200263
Irving Fernando Alvarado Asensio	202200349

Catedrático: Ing. Jurgén Ramírez
Auxiliar: Axel Calderón / Danny Cuxum

Guatemala, 22 de febrero de 2025

Introducción

El avance de la tecnología y la creciente dependencia de los centros de datos hacen imprescindible el monitoreo preciso de las condiciones ambientales dentro de los cuartos de servidores. Factores como la temperatura, la humedad, la calidad del aire y la corriente eléctrica pueden afectar directamente el rendimiento y la vida útil del hardware, lo que resalta la necesidad de un sistema de monitoreo eficiente.

Este proyecto propone el desarrollo de un Sistema Inteligente de Monitoreo Ambiental basado en tecnología IoT, capaz de medir y registrar variables ambientales en tiempo real. Utilizando sensores especializados y un microcontrolador Arduino, el sistema recopilará información clave y la visualizará en una pantalla LCD y un dashboard en Processing, permitiendo la toma de decisiones basada en datos.

A través del enfoque del Smart Connected Design Framework, se abordará la implementación del sistema desde sus diferentes capas: percepción, red, procesamiento y aplicación. Cada una de estas capas será documentada detalladamente para explicar cómo los sensores capturan los datos, cómo se procesan y almacenan, y cómo se presentan a los usuarios de manera efectiva.

Este documento describe la estructura y el funcionamiento del sistema, los componentes utilizados, el flujo de información, los beneficios del monitoreo ambiental en centros de datos y el impacto de la solución propuesta. Además, se incluyen bocetos, diagramas y explicaciones técnicas que permitirán comprender la implementación y facilitar futuras mejoras.

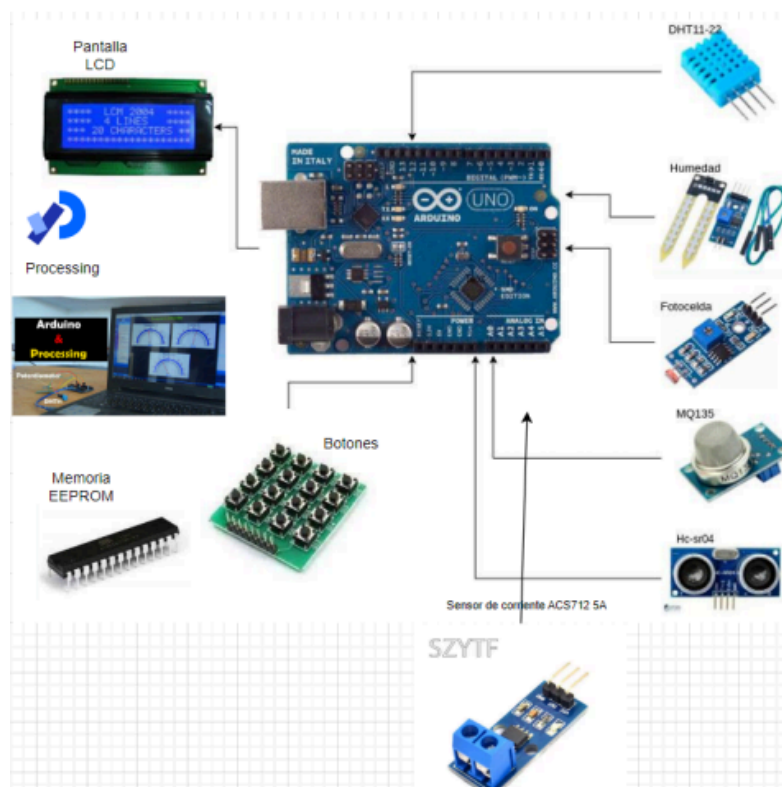
Descripción del Sistema

El sistema consiste en la integración de una placa Arduino con múltiples sensores para monitorear condiciones dentro de un Centro de Datos. La placa se encarga de realizar la lectura de los sensores, procesar la información y enviar los datos relevantes a una interfaz visual desarrollada en Processing.

A través de este dashboard, el usuario puede visualizar en tiempo real métricas clave del entorno, lo que facilita la supervisión y toma de decisiones. El sistema no solo captura y transmite datos, sino que también los interpreta para proporcionar información útil y mejorar la gestión del Data Center.

- **Componentes utilizados**

- ☐ Arduino
- ☐ DHT11
- ☐ HC-SR04
- ☐ Fotorresistencia
- ☐ ACS712 (5A)
- ☐ Pantalla LCD
- ☐ LEDs
- ☐ Jumpers
- ☐ Protoboard



Capas del Smart Connected Design Framework

Smart App

La Smart App en Processing recibe en tiempo real los datos de los sensores conectados a Arduino y los visualiza en un dashboard interactivo, mostrando información clave como temperatura, humedad, calidad del aire y consumo de corriente. La aplicación permite a los usuarios monitorear las condiciones del cuarto de servidores, proporcionando alertas visuales cuando los valores superan umbrales críticos y ofreciendo una interfaz sencilla para interactuar con los datos, consultar valores históricos y tomar decisiones en tiempo real. Es una parte esencial del proyecto, ya que permite visualizar los datos tal cual van llegando a los sensores. Y también abre paso a la siguiente capa de este marco de trabajo.

Analytics

Procesamiento de datos:

La capa de **Analytics** procesa los datos de los sensores para detectar patrones y tendencias en tiempo real. Analiza variables como temperatura, humedad y consumo de energía para garantizar un ambiente adecuado.

Detección de anomalías:

Se establece un umbral para cada parámetro, y si los datos superan estos límites, el sistema genera alertas. Esto permite identificar problemas potenciales antes de que afecten el rendimiento.

Optimización continua:

El análisis de datos históricos ayuda a mejorar la eficiencia del sistema, identificando áreas de mejora en el uso de energía y el control climático.

Predicción de condiciones:

Mediante modelos predictivos, el sistema puede anticipar problemas como un aumento de temperatura o consumo, permitiendo tomar acciones preventivas.

Almacenamiento y análisis histórico:

Los datos históricos se almacenan para realizar análisis comparativos, ayudando a tomar decisiones informadas a largo plazo

Sensors

1. Sensor de temperatura y humedad (DHT11)
 - Función: Miden la temperatura y la humedad relativa del ambiente.
 - Funcionamiento: Utilizan un termistor para detectar la temperatura y un sensor capacitivo para la humedad. Envían los datos en un formato digital que el Arduino interpreta.
 - Aplicación en el sistema: Detectan condiciones que pueden afectar el rendimiento de los servidores, generando alertas si los valores exceden los umbrales críticos.
2. Sensor de movimiento y proximidad (HC-SR04)
 - Función: Detecta la presencia de personas en el cuarto de servidores.
 - Funcionamiento: Emite un pulso ultrasónico y mide el tiempo que tarda en reflejarse, calculando la distancia de los objetos cercanos.
 - Aplicación en el sistema: Registra accesos y presencia en el cuarto de servidores para mejorar la seguridad y el control de acceso.
3. Sensor de iluminación (Fotorresistencia)
 - Fotorresistencia:
 - Función: Mide la cantidad de luz en el entorno.
 - Funcionamiento: Su resistencia varía dependiendo de la intensidad de luz incidente.
 - Aplicación en el sistema: Puede activar o desactivar luces automáticamente, optimizando el consumo energético.
4. Sensor de calidad del aire (MQ135)
 - Función: Mide la concentración de CO₂ y otros gases en el ambiente.
 - Funcionamiento: Utiliza un material semiconductor que cambia su resistencia en función de la concentración de gases presentes.
 - Aplicación en el sistema: Ayuda a evaluar la ventilación del cuarto de servidores y prevenir acumulaciones de gases nocivos.
5. Sensor de corriente eléctrica (ACS712 5A)
 - Función: Mide el consumo eléctrico de los servidores.
 - Funcionamiento: Usa un sensor de efecto Hall para detectar la corriente que fluye a través de él sin necesidad de contacto directo.
 - Aplicación en el sistema: Permite monitorear el suministro de energía, detectando caídas o sobrecargas que puedan afectar los equipos.

Actuadores utilizados: Los actuadores permiten al sistema responder a las condiciones detectadas por los sensores, proporcionando alertas visuales y textuales.

1. Pantalla LCD

- Función: Muestra información en tiempo real sobre las mediciones de los sensores.
- Funcionamiento: Recibe datos desde el Arduino y los representa en un formato visual comprensible.
- Aplicación en el sistema: Muestra mensajes de advertencia en caso de valores anormales de temperatura, humedad o energía.

2. LED's de advertencia (Verde, Amarillo y Rojo)

- Función: Indican visualmente el estado del cuarto de servidores según las mediciones.
- Funcionamiento: Se encienden de distintos colores según los valores detectados por los sensores.
- Aplicación en el sistema:
 - LED Verde: Estado normal.
 - LED Amarillo: Precaución (valores cercanos a los límites permitidos).
 - LED Rojo: Alerta (niveles críticos que requieren atención inmediata).

3. Botones físicos

- Función: Permiten la interacción con el sistema para visualizar información y registrar datos históricos.
- Funcionamiento: Cuando un usuario presiona un botón, el Arduino ejecuta una acción específica.
- Aplicación en el sistema:
 - Un botón para mostrar los valores de los sensores en tiempo real.
 - Un botón para guardar datos en la memoria EEPROM.
 - Un botón para recuperar y mostrar información almacenada.

Conectividad

Esta capa permite la comunicación entre Arduino y el sistema de monitoreo en Processing, asegurando la transmisión en tiempo real de los datos capturados por los sensores. La conexión se establece mediante comunicación serial (UART) a través de un cable USB, con un baud rate de 9600 bps para garantizar estabilidad y precisión.

El Arduino envía datos en un formato estructurado, usando una cadena de texto con separadores definidos, por ejemplo:

25.3,60,300,400,2.1

donde cada valor representa una medición específica (temperatura, humedad, luz, CO₂ y corriente). Processing recibe esta información, la procesa y la visualiza en el dashboard.

En futuras mejoras, la comunicación podría migrar a un protocolo MQTT, permitiendo el envío de datos a una plataforma remota mediante módulos WiFi como ESP8266 o ESP32, optimizando la gestión y almacenamiento de la información en la nube.

Infraestructura

Esta capa se refiere a la base física y tecnológica que soporta el sistema de monitoreo. Esto incluye el hardware como los sensores, Arduino, y actuadores (pantalla LCD, LEDs), así como la conectividad y el almacenamiento de datos. La infraestructura también abarca el entorno de desarrollo y los recursos necesarios para procesar y visualizar la información, asegurando que el sistema sea escalable y pueda manejar múltiples dispositivos o ubicaciones si es necesario.

Implementación del Código

Explicación del código en Arduino

Las librerías a instalar son:

- DHT sensor library: Para usar sensores DHT11 o DHT22.
- LiquidCrystal I2C: Para controlar la pantalla LCD con interfaz I2C.
- La librería EEPROM ya está incluida por defecto en el entorno de Arduino.

setup(): Inicializa los componentes del sistema, como los sensores DHT11, la pantalla LCD, y configura los pines para los sensores de distancia, corriente, luz, y los botones de control. También establece el modo de los LED's y los pines de entrada y salida.

```

void setup() {
  Serial.begin(9600); // Inicializa la comunicación serial a 9600 baudios
  dht.begin();        // Inicializa el sensor DHT11
  lcd.init();          // Inicializa la pantalla LCD
  lcd.backlight();     // Enciende la luz de fondo de la LCD
  lcd.clear();         // Limpia la pantalla LCD
  lcd.setCursor(0, 0); // Posiciona el cursor en la fila 0, columna 0 de la LCD
  lcd.print("Bienvenidos");

  // Configuración de pines de entrada y salida
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(LDR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(MODO_NORMAL_PIN, INPUT);
  pinMode(MODO_GUARDAR_PIN, INPUT);
  pinMode(MODO_MOSTRAR_PIN, INPUT);

  pinMode(LED_OK, OUTPUT);
  pinMode(LED_MID, OUTPUT);
  pinMode(LED_BAD, OUTPUT);
}

```

loop(): Esta es la función principal que se ejecuta continuamente. Controla el flujo del programa según el modo seleccionado (tiempo real, guardar, mostrar datos guardados). Realiza las lecturas de los sensores y actualiza los datos cada 500 ms. Dependiendo del modo, muestra los datos en el LCD o los guarda en la memoria EEPROM.

```

void loop() {
  manejarBotones();

  // Enviar datos por Serial cada 0.5s
  if (millis() - ultimaActualizacionSerial >= 500) {
    ultimaActualizacionSerial = millis();
    enviarDatosSerial();
  }

  switch (modoActual) {
    case TIEMPO_REAL:
      if (alertaActiva) {
        if (millis() - tiempoAlerta >= 3000) { // Prioridad de 3s a alertas
          alertaActiva = false;
        }
      } else if (millis() - ultimaActualizacionLCD >= 500) {
        ultimaActualizacionLCD = millis();
        mostrarValoresTiempoReal();
      }
      break;

    case GUARDAR:
      guardarValoresEEPROM();
      modoActual = TIEMPO_REAL; // Volver automáticamente al modo normal
      break;

    case MOSTRAR_GUARDADOS:
      mostrarValoresEEPROM();
      break;
  }
}

```


manejarBotones(): Monitorea el estado de los botones de control (modo normal, guardar y mostrar datos). Dependiendo de cuál botón es presionado, cambia el modo del sistema (tiempo real, guardar o mostrar datos almacenados).

```
void manejarBotones() {
    if (digitalRead(MODO_NORMAL_PIN) == HIGH) {
        modoActual = TIEMPO_REAL;
        alertaActiva = false; // Salir de cualquier alerta
    }
    if (digitalRead(MODO_GUARDAR_PIN) == HIGH) {
        modoActual = GUARDAR;
    }
    if (digitalRead(MODO_MOSTRAR_PIN) == HIGH) {
        modoActual = MOSTRAR_GUARDADOS;
    }
}
```

enviarDatosSerial(): Cada 500 ms, esta función recopila los datos de los sensores de temperatura, humedad, distancia, luz, calidad del aire y corriente, y los envía al puerto serial para ser visualizados en el monitor serial.

```
void enviarDatosSerial() {
    float distancia = obtenerDistancia();
    int calidadAire = analogRead(MQ135_PIN);
    calidadAire = calcularCo2(calidadAire);
    int luz = digitalRead(LDR_PIN);
    float humedad = dht.readHumidity();
    float temperatura = dht.readTemperature();
    float corriente = obtener_corriente();

    Serial.println(String(temperatura) + "," + String(humedad) + "," + String(distancia) + "
```

obtener_corriente(): Esta función lee el voltaje de un sensor de corriente y calcula el valor de la corriente, basándose en la lectura del ADC. Además, se calcula el valor máximo y mínimo de corriente durante las mediciones para proporcionar una lectura precisa.

```
float obtener_corriente() {
    float voltajeSensor;
    float corriente = 0;
    float Imax = 0;
    float Imin = 0;

    voltajeSensor = analogRead(CORRIENTE_PIN) * (5.0 / 1023.0);
    corriente = 0.9 * corriente + 1 * ((voltajeSensor - 2.5) / sensibilidad);
    if (corriente > Imax) {
        Imax = corriente;
    }
    if (corriente < Imin) {
        Imin = corriente;
    }
    return (((Imax - Imin) / 2)) * 10;
}
```

calcularCo2(): Recibe el valor analógico del sensor MQ135, que mide la calidad del aire, y lo procesa para estimar la concentración de CO₂ en el ambiente. Esta función aún es simple, pero puede ser extendida para hacer un análisis más detallado.

```
float calcularCo2(int valorADC) {  
    return valorADC;  
}
```

mostrarValoresTiempoReal(): Muestra los datos de los sensores en el LCD en tiempo real. Si se detectan errores (como una lectura incorrecta de temperatura o humedad), muestra un mensaje de alerta. También se manejan las alertas de los valores críticos de temperatura, humedad y corriente.

```
void mostrarValoresTiempoReal() {  
    float distancia = obtenerDistancia();  
    int calidadAire = analogRead(MQ135_PIN);  
    calidadAire = calcularCo2(calidadAire);  
    int luz = digitalRead(LDR_PIN);  
    float humedad = dht.readHumidity();  
    float temperatura = dht.readTemperature();  
    float corriente = obtener_corriente();  
  
    if (isnan(humedad) || isnan(temperatura)) {  
        mostrarAlerta("DHT11 Error");  
        return;  
    }  
  
    manejarAlertas(temperatura, humedad, corriente);  
  
    if (!alertaActiva) {  
        lcd.clear();  
        lcd.setCursor(0, 0);  
        lcd.print("T:" + String(temperatura) + " H:" + String(humedad));  
        lcd.setCursor(0, 1);  
        lcd.print("CO2:" + String(calidadAire) + " C:" + String(corriente));  
    }  
}
```

mostrarAlerta(): Muestra un mensaje de alerta en el LCD, indicando un error o una condición crítica, y activa el modo de alerta, lo que hace que el sistema deje de mostrar los valores normales y ponga énfasis en la alerta. La alerta se mantiene activa durante 3 segundos.

```
void mostrarAlerta(String mensaje) {  
    lcd.clear();  
    lcd.setCursor(0, 0);  
    lcd.print(mensaje);  
    alertaActiva = true;  
    tiempoAlerta = millis();  
}
```

obtenerDistancia(): Utiliza el sensor de ultrasonido para medir la distancia en centímetros entre el sensor y un objeto. Se envía una señal de activación (TRIG) y se mide el tiempo de respuesta del eco para calcular la distancia.

```
float obtenerDistancia() {  
    digitalWrite(TRIG_PIN, LOW);  
    delayMicroseconds(2);  
    digitalWrite(TRIG_PIN, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(TRIG_PIN, LOW);  
  
    long duration = pulseIn(ECHO_PIN, HIGH);  
    return duration * 0.034 / 2;  
}
```

guardarValoresEEPROM(): Guarda los valores de los sensores (temperatura, humedad, distancia, calidad del aire, corriente y luz) en la memoria EEPROM para su posterior visualización. Cada conjunto de datos se guarda en una dirección de memoria consecutiva.

```
void guardarValoresEEPROM() {  
    float distancia = obtenerDistancia();  
    float calidadAire = analogRead(MQ135_PIN);  
    calidadAire = calcularCo2(calidadAire);  
    float luz = digitalRead(LDR_PIN);  
    float humedad = dht.readHumidity();  
    float temperatura = dht.readTemperature();  
    float corriente = obtener_corriente();  
  
    EEPROM.put(eepromAddress, temperatura);  
    EEPROM.put(eepromAddress + sizeof(float), humedad);  
    EEPROM.put(eepromAddress + 2 * sizeof(float), distancia);  
    EEPROM.put(eepromAddress + 3 * sizeof(float), calidadAire);  
    EEPROM.put(eepromAddress + 4 * sizeof(float), corriente);  
    EEPROM.put(eepromAddress + 5 * sizeof(float), luz);  
  
    lcd.clear();  
    lcd.setCursor(0, 0);  
    lcd.print("Valor guardado");  
  
    eepromAddress += 6 * sizeof(float);  
    if (eepromAddress >= EEPROM.length()) {  
        eepromAddress = 0;  
    }  
}
```

mostrarValoresEEPROM(): Lee los datos almacenados en la memoria EEPROM y los muestra en el LCD, presentando los datos guardados uno por uno. Después de mostrar cada conjunto de valores, se hace una pausa de 3 segundos antes de mostrar el siguiente registro.

```

void mostrarValoresEEPROM() {
    float temperatura, humedad, distancia, luz, calidadAire, corriente;
    int addr = 0;
    int registro = 1;

    while (addr < eepromAddress && modoActual == MOSTRAR_GUARDADOS) {
        EEPROM.get(addr, temperatura);
        EEPROM.get(addr + sizeof(float), humedad);
        EEPROM.get(addr + 2 * sizeof(float), distancia);
        EEPROM.get(addr + 3 * sizeof(float), calidadAire);
        EEPROM.get(addr + 4 * sizeof(float), corriente);
        EEPROM.get(addr + 5 * sizeof(float), luz);

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("T:" + String(temperatura) + " H:" + String(humedad));
        lcd.setCursor(0, 1);
        lcd.print("CO2:" + String(calidadAire) + " C:" + String(corriente));
        delay(3000);

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("D:" + String(distancia) + " L:" + String(luz));
        lcd.setCursor(0, 1);
        lcd.print("Reg #" + String(registro));
        delay(3000);

        addr += 6 * sizeof(float);
        registro++;
    }
}

```

manejarAlertas(): Evalúa las lecturas de los sensores para verificar si alguna está fuera de los límites definidos. Si los valores de temperatura, humedad o corriente son anormales, se activa una alerta y se ilumina el LED correspondiente (verde para normal, amarillo para moderado y rojo para crítico).

```

void manejarAlertas(float temperatura, float humedad, float corriente) {
    int errores = 0;
    if (corriente < 0.5 || corriente > 6) {
        errores = errores + 1;
        mostrarAlerta("Corriente ALERTA");
    }

    if (humedad < 20 || humedad > 60) {
        errores = errores + 1;
        mostrarAlerta("Humedad ALERTA");
    }

    if (temperatura < 18 || temperatura > 35) {
        errores = errores + 1;
        mostrarAlerta("Temperatura ALERTA");
    }

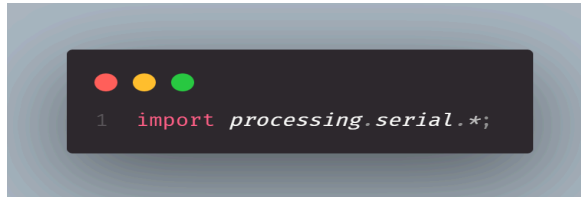
    if (errores > 0 && errores <= 2) {
        digitalWrite(LED_MID, HIGH);
        digitalWrite(LED_OK, LOW);
        digitalWrite(LED_BAD, LOW);
    } else if (errores > 2) {
        digitalWrite(LED_BAD, HIGH);
        digitalWrite(LED_OK, LOW);
        digitalWrite(LED_MID, LOW);
    } else if (errores < 1) {
        digitalWrite(LED_OK, HIGH);
        digitalWrite(LED_MID, LOW);
        digitalWrite(LED_BAD, LOW);
    }
}

```

Explicación del Código Processing

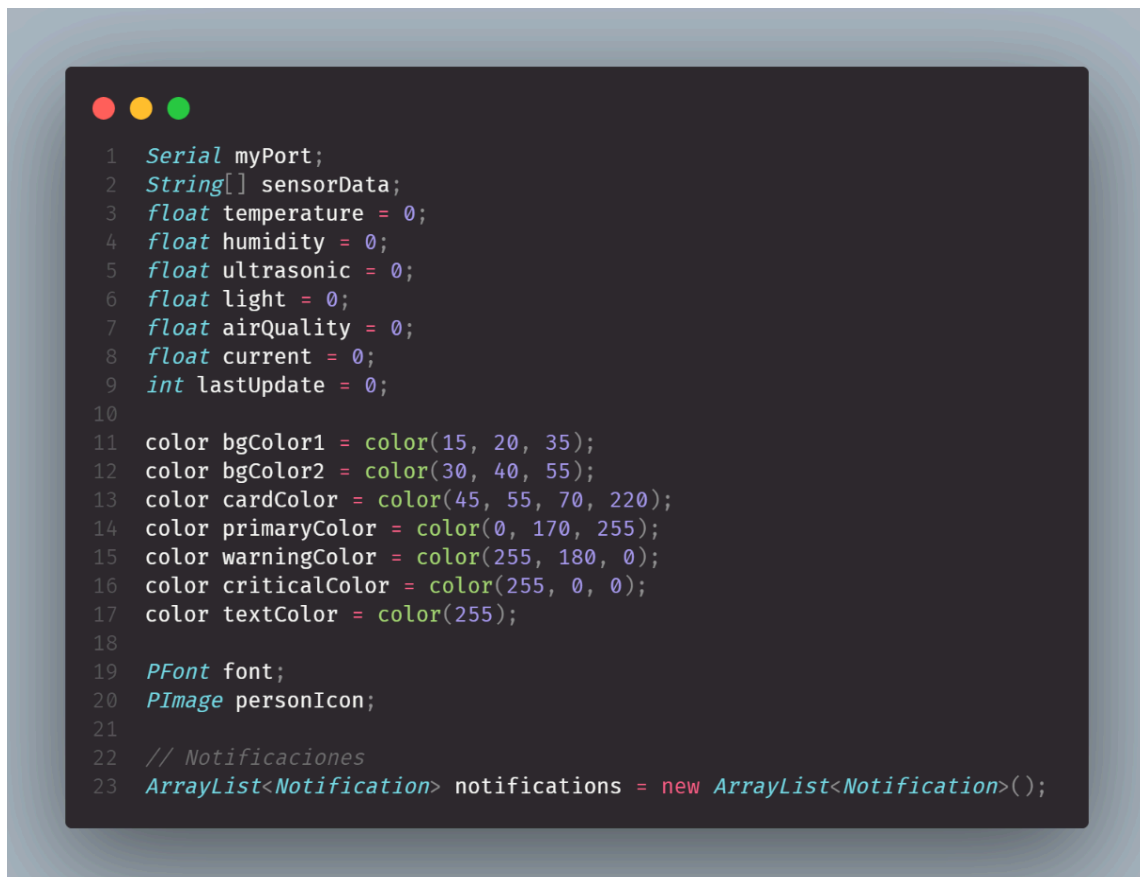
Este programa en Processing recibe datos de sensores de una maqueta a través de una conexión serial con un Arduino. Representa visualmente estos datos mediante indicadores y notificaciones, proporcionando una interfaz gráfica para el monitoreo de un data center.

1. Librerías importadas



Se importa la librería `processing.serial`, que permite la comunicación entre Processing y el Arduino a través del puerto serial.

2. Declaración de variables globales



- **myPort:** Objeto para manejar la comunicación serial.
- **sensorData:** Arreglo para almacenar los datos recibidos desde el Arduino.
- **Variables para almacenar los valores de los sensores:** temperature, humidity, ultrasonic, light, airQuality, current.
- **lastUpdate:** Variable para almacenar el tiempo de la última actualización.
- **bgColor1, bgColor2, cardColor, primaryColor, warningColor, textColor:** Variables para almacenar los valores que definen el color de fondo, cartas, texto y advertencias.
- **font:** Fuente para el texto.
- **personIcon:** Imagen para el indicador de presencia.
- **Notifications:** Se utiliza una lista para almacenar las notificaciones que se muestran en la interfaz.

3. Funciones principales

Función setup()

```

1 void setup() {
2   size(875, 680);
3   noStroke();
4   smooth();
5   font = createFont("Roboto-Regular.ttf", 24);
6   textFont(font);
7   personIcon = loadImage("person_icon.png");
8   String portName = Serial.list()[0]; // Cambiar índice si es necesario
9   myPort = new Serial(this, portName, 9600);
10  myPort.bufferUntil('\n');
11 }

```

- Define el tamaño de la ventana.
- Carga la fuente y la imagen del icono.
- Establece la comunicación serial con el Arduino.
- Configura el puerto para recibir datos hasta encontrar un salto de línea ('\n').

Función draw() (bucle principal)

```
1 void draw() {
2   drawGradientBackground();
3
4   drawTitle();
5
6   int cardWidth = 250; // Ancho de cada carta
7   int cardHeight = 250; // Alto de cada carta
8   int margin = 25; // Margen entre cartas
9
10  // Primera fila: 3 cartas
11  drawGauge(margin, 100, cardWidth, cardHeight, temperature, "Temperatura", "°C", 0, 60);
12  drawGauge(margin + cardWidth + margin, 100, cardWidth, cardHeight, humidity, "Humedad", "%", 0, 80);
13  drawGauge(margin + 2 * (cardWidth + margin), 100, cardWidth, cardHeight, airQuality, "CO2 en el ambiente", "PPM", 0, 1000);
14
15  // Segunda fila: 2 cartas
16  drawCurrentBar(margin, 100 + cardHeight + margin, cardWidth, cardHeight, current, 0, 5);
17  drawLightIndicator(margin + cardWidth + margin, 100 + cardHeight + margin, cardWidth, cardHeight, light);
18  drawPresenceIndicator(margin + 2 * (cardWidth + margin), 100 + cardHeight + margin, cardWidth, cardHeight);
19
20  drawDateTime();
21
22  // Dibujar notificaciones
23  drawNotifications();
24 }
```

- Dibuja el fondo degradado.
- Muestra el título de la interfaz.
- Llama a funciones para dibujar indicadores de temperatura, humedad, calidad del aire, corriente, luz y presencia.
- Muestra la fecha y las notificaciones.

Función serialEvent(Serial myPort)

```
1 void serialEvent(Serial myPort) {
2   String data = myPort.readStringUntil('\n');
3   if (data != null) {
4     data = trim(data);
5     sensorData = split(data, ',');
6
7     // Asigna los valores a las variables
8     if (sensorData.length == 6) {
9       temperature = float(sensorData[0]);
10      humidity = float(sensorData[1]);
11      ultrasonic = float(sensorData[2]);
12      light = float(sensorData[3]);
13      airQuality = float(sensorData[4]);
14      current = float(sensorData[5]);
15      checkCriticalLevels();
16    }
17  }
18 }
```

- Captura los datos recibidos por el puerto serial.
- Separa los valores usando, como delimitador.
- Convierte los valores en flotantes y los almacena en las variables de sensores.
- Llama a `checkCriticalLevels()` para verificar si hay condiciones de alerta.

Funciones de dibujo

```

1 void drawGauge(float x, float y, float w, float h, float value, String label, String unit, float minVal, float maxVal) {
2 }
3
4 void drawLightIndicator(float x, float y, float w, float h, float lightValue) {
5 }
6
7 void drawCurrentBar(float x, float y, float w, float h, float value, float minVal, float maxVal) {
8 }
9
10 void drawPresenceIndicator(float x, float y, float w, float h) {
11 }
12
13 void drawTitle() {
14 }
15
16 void drawDateTime() {
17 }
18
19 void checkCriticalLevels() {
20 }
21
22 void addNotification(String message, color notificationColor) {
23 }
24
25 void drawNotifications() {
26 }

```

- **drawGauge():** Dibuja un medidor circular para valores como temperatura, humedad y calidad del aire.
- **drawLightIndicator():** Dibuja un indicador de luz con un círculo que cambia de brillo.
- **drawCurrentBar():** Dibuja una barra de consumo eléctrico con animación.
- **drawPresenceIndicator():** Dibuja un indicador de presencia basado en el sensor ultrasónico.
- **drawGradientBackground():** Dibuja un fondo con gradiente.
- **drawTitle():** Dibuja el título de la aplicación.
- **drawDateTime():** Dibuja la fecha y hora actual.
- **checkCriticalLevels():** Verifica si los valores de los sensores están fuera de los rangos normales y agrega notificaciones.
- **addNotification():** Agrega una notificación a la lista.
- **drawNotifications():** Dibuja las notificaciones en la pantalla.

4. Clase Notifications

```
1  class Notification {
2      String message;
3      color notificationColor;
4      int timer;
5      float x, y;
6      float targetX;
7      boolean isClosing = false;
8
9      Notification(String message, color notificationColor) {
10         this.message = message;
11         this.notificationColor = notificationColor;
12         this.timer = millis();
13         this.x = width;
14         this.targetX = width - 400;
15     }
16
17     void display(int index) {
18         x = lerp(x, targetX, 0.1);
19
20
21         float notificationY = 20 + index * 60;
22
23         fill(0, 50);
24         noStroke();
25         rect(x + 3, notificationY + 3, 380, 50, 10);
26
27         fill(notificationColor);
28         noStroke();
29         rect(x, notificationY, 380, 50, 10);
30
31         // Dibujar icono
32         fill(255);
33         textSize(24);
34         textAlign(CENTER, CENTER);
35         text("△", x + 30, notificationY + 25);
36
37         // Dibujar texto
38         fill(textColor);
39         textSize(18);
40         textAlign(LEFT, CENTER);
41         text(message, x + 20, notificationY + 25);
42
43         // Botón de cerrar
44         fill(255, 100);
45         rect(x + 340, notificationY + 10, 30, 30, 5);
46         fill(0);
47         textSize(20);
48         textAlign(CENTER, CENTER);
49         text("×", x + 355, notificationY + 25);
50
51         // Verificar si se hace clic en el botón de cerrar
52         if (mousePressed && mouseX > x + 340 && mouseX < x + 370 && mouseY > notificationY + 10 && mouseY < notificationY + 40) {
53             isClosing = true;
54             targetX = width; // Mover fuera de la pantalla
55         }
56
57         // Eliminar si está fuera de la pantalla
58         if (isClosing && x > width - 10) {
59             notifications.remove(this);
60         }
61     }
62
63     boolean isExpired() {
64         return millis() - timer > 3000; // Duración de 3 segundos
65     }
66 }
67
```


Atributos:

- message: Mensaje de la notificación.
- notificationColor: Color de fondo de la notificación.
- timer: Guarda el tiempo en que la notificación fue creada.
- x, y: Posición de la notificación en pantalla.
- targetX: Posición objetivo a la que debe desplazarse la notificación.
- isClosing: Indica si la notificación está en proceso de cierre.

Constructor:

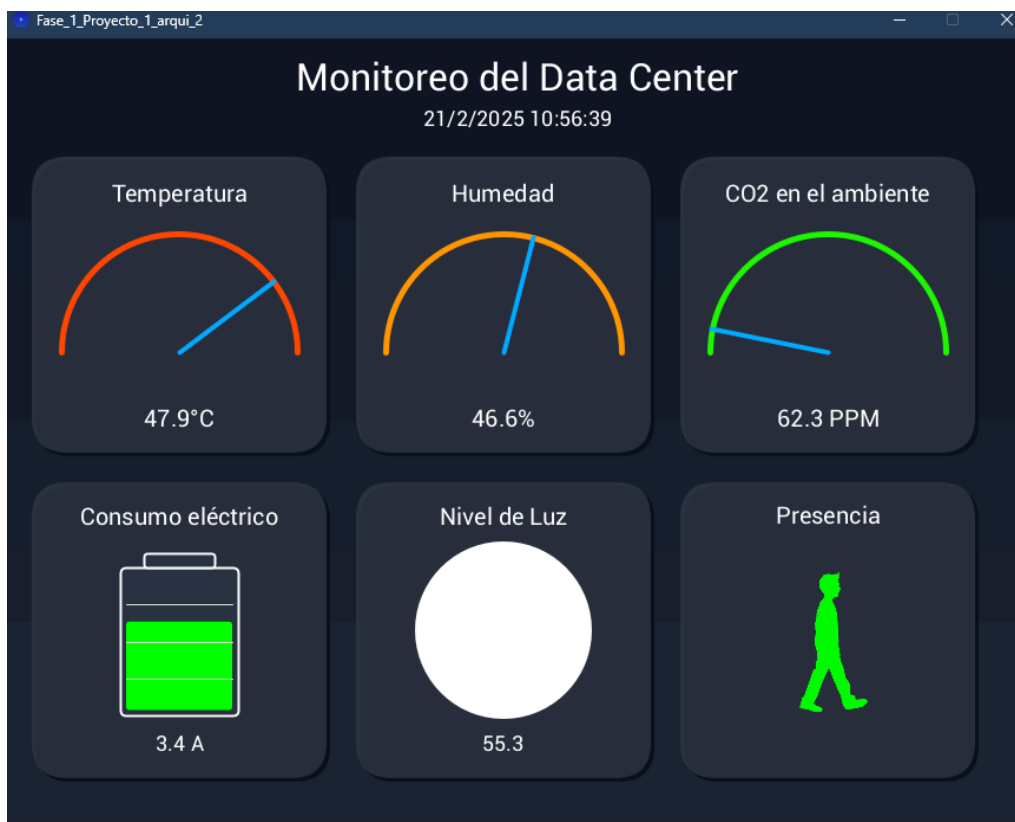
- Recibe el mensaje y color de la notificación.
- Inicializa x en el borde derecho de la pantalla y targetX en una posición más visible (400 píxeles hacia la izquierda).
- Guarda el tiempo de creación con millis().

Método display(int index):

- Desplaza la notificación suavemente desde x hasta targetX usando lerp().
- Dibuja un rectángulo con el color de fondo de la notificación.
- Muestra un icono de advertencia () y el mensaje.
- Agrega un botón de cierre (×) en la esquina superior derecha.
- Si se hace clic en el botón de cierre, la notificación se mueve fuera de la pantalla (targetX = width).
- Si la notificación ha salido completamente de la pantalla, se elimina de la lista notifications.

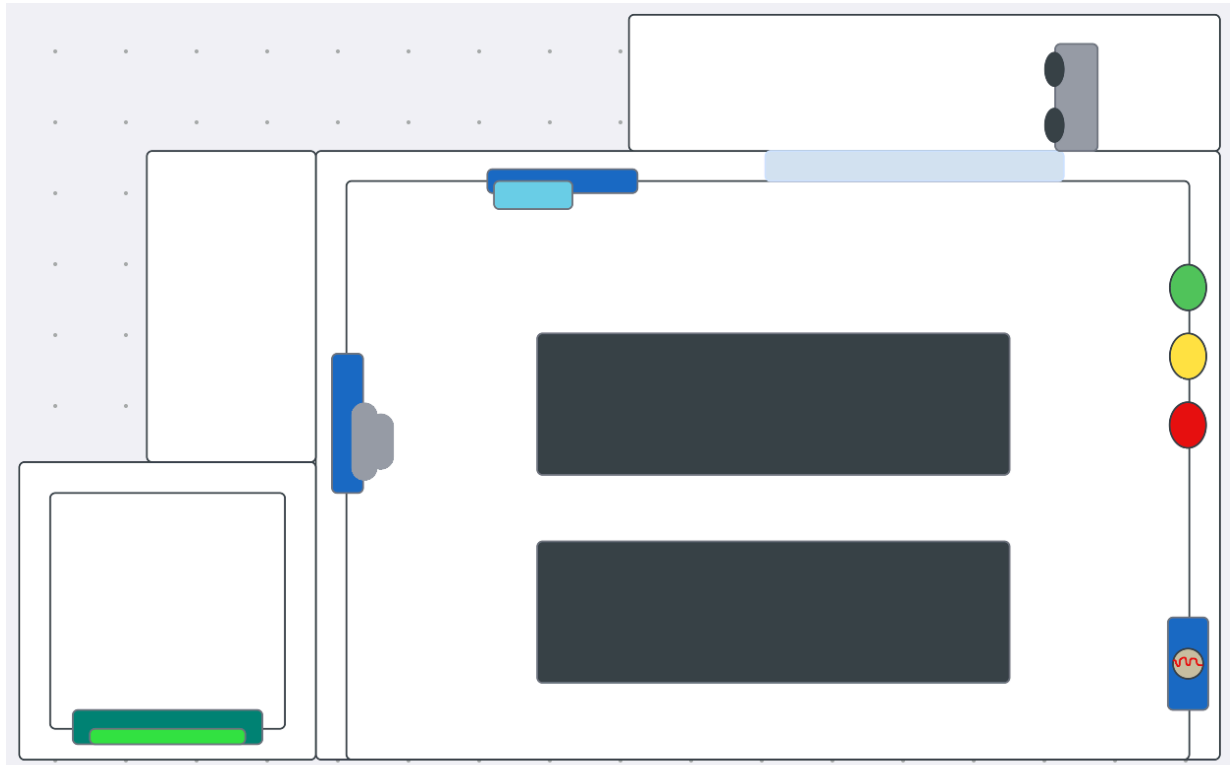
Método isExpired():

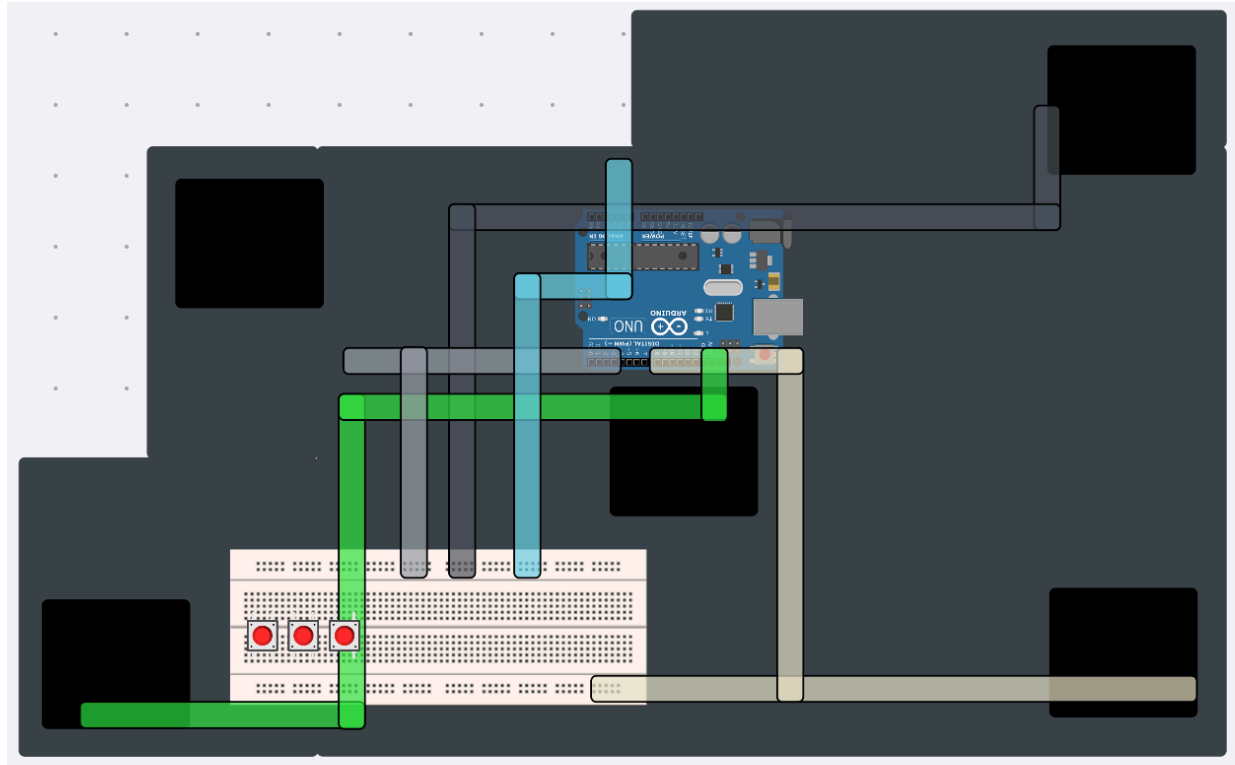
- Retorna true si han pasado más de 3 segundos desde su creación, indicando que debe eliminarse automáticamente.



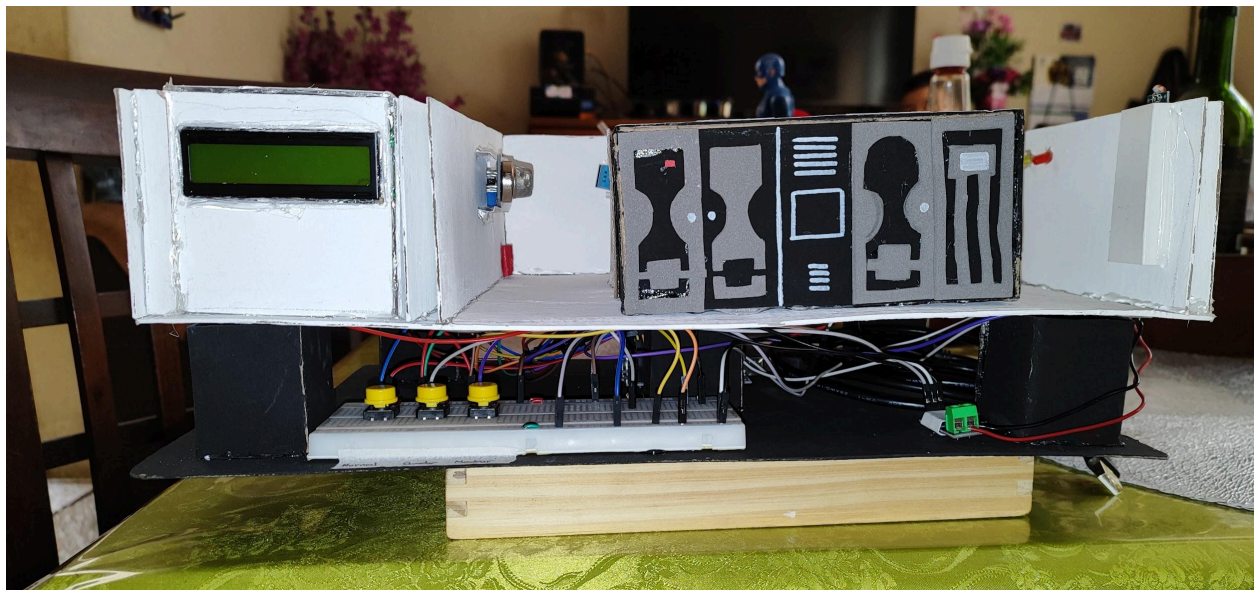
Prototipo Físico

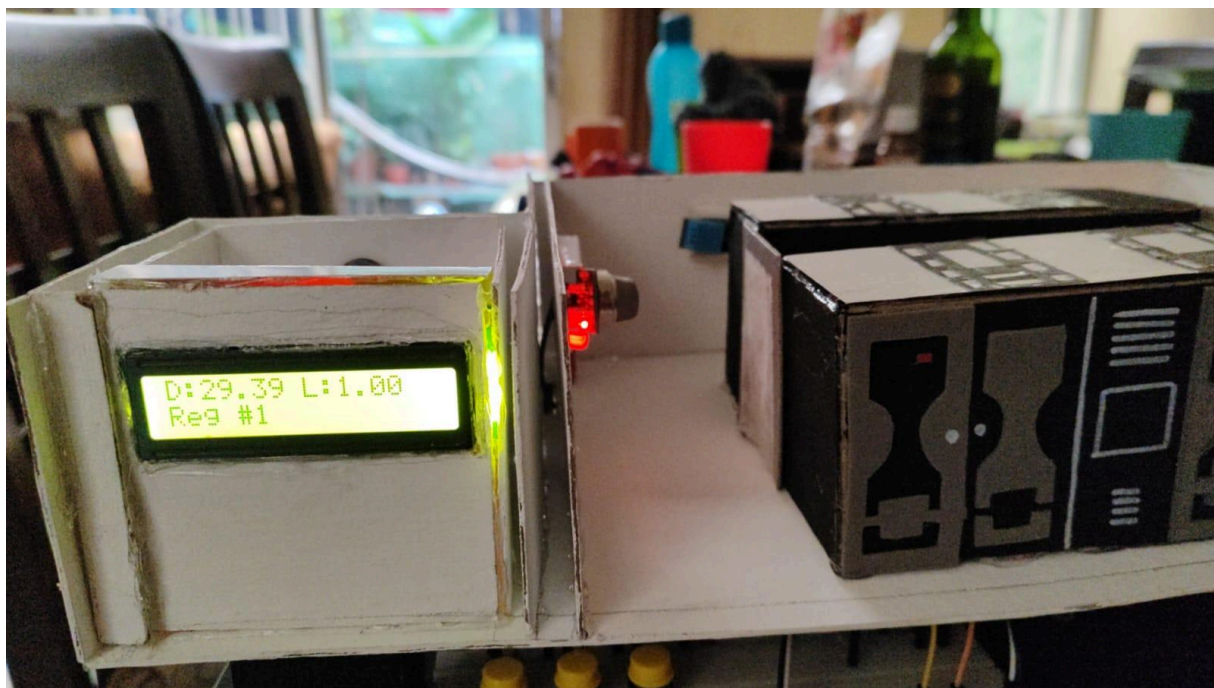
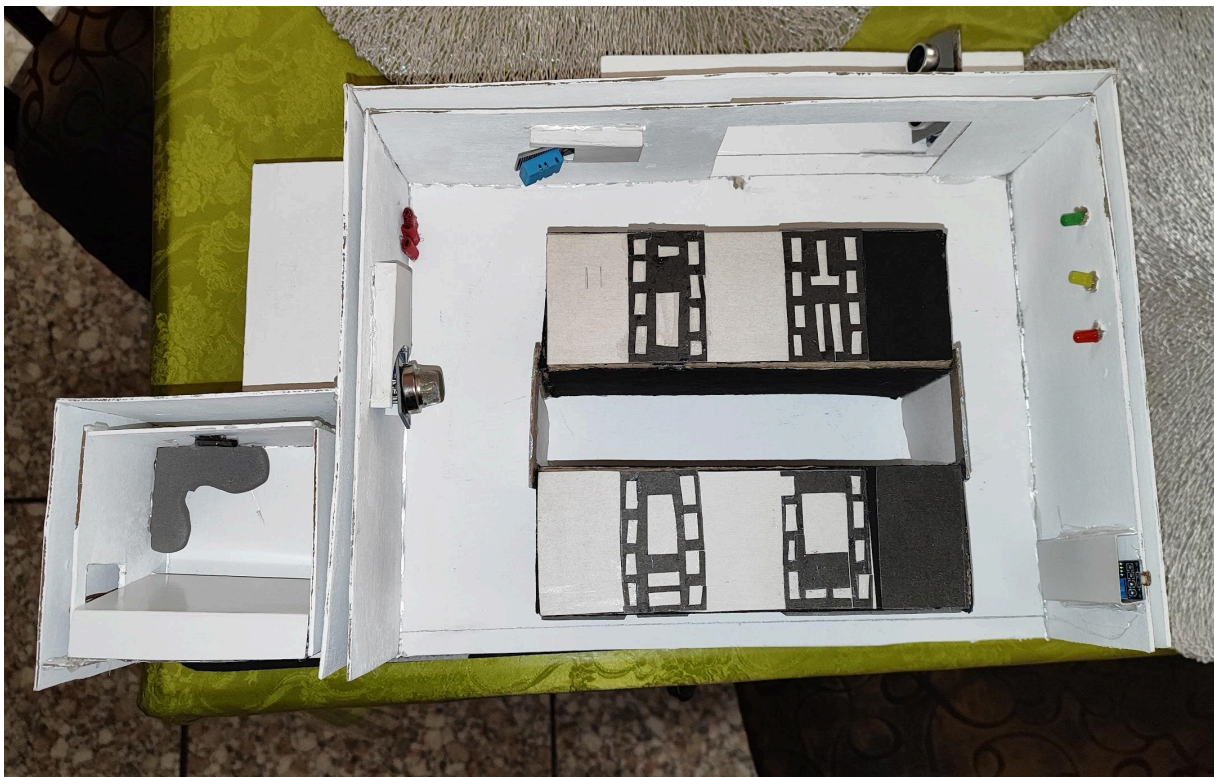
A continuación se presentan los planos y el diseño previo a la realización de la maqueta física, para poder entender en qué parte iría cada sensor y cada parte esencial de la maqueta. También para poder ver cómo sería el cableado físico de cada sensor, y en donde estaría el Arduino Uno y la protoboard en el diseño físico.

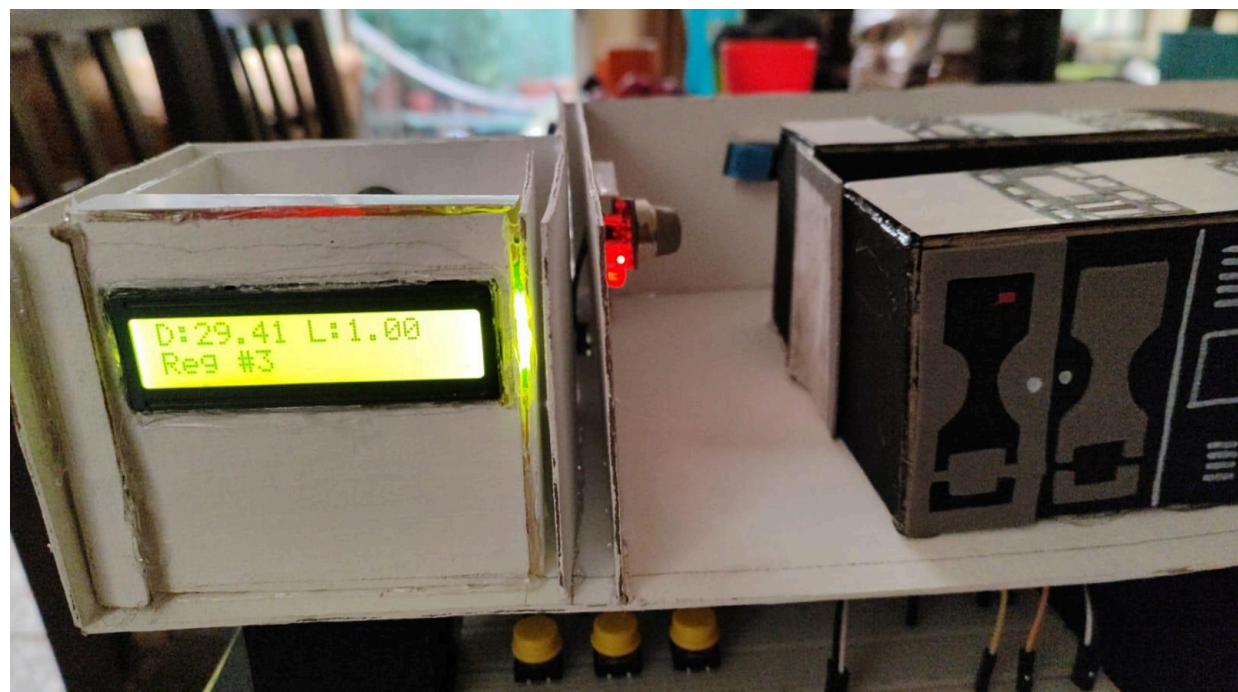
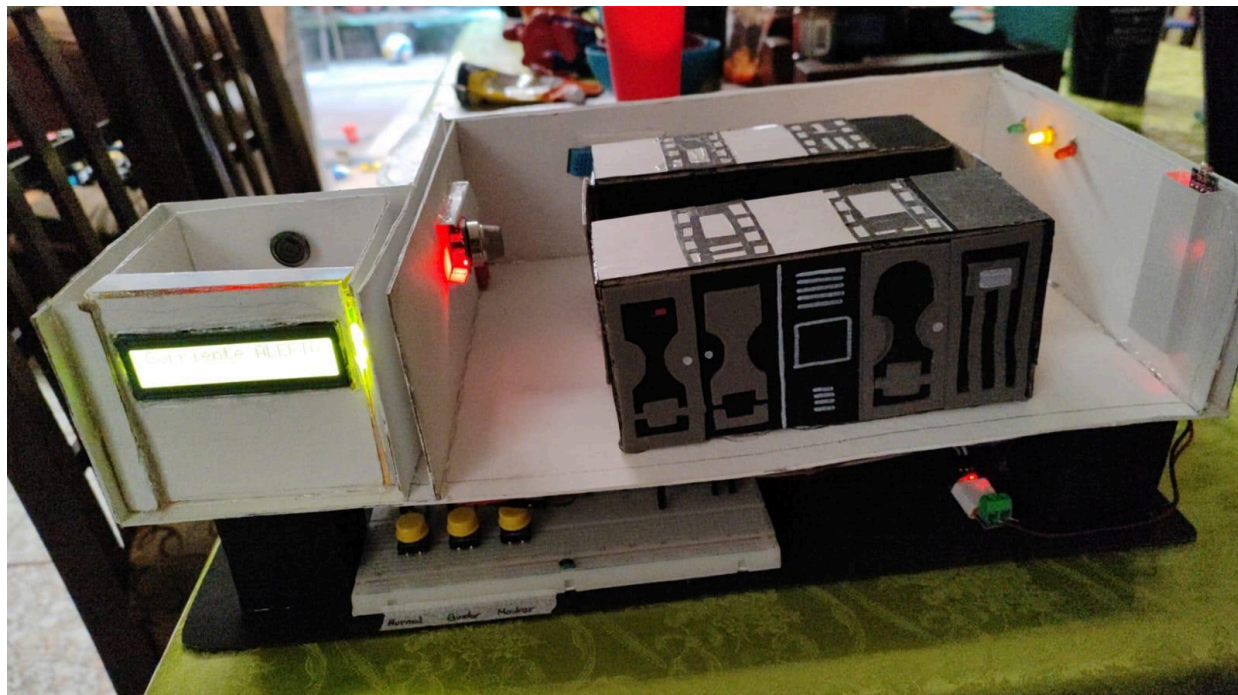




En las siguientes fotografías se muestra la maqueta física finalizada y también se muestra cada parte importante de la maqueta, así como el correcto funcionamiento de esta.







Link del Repositorio

https://github.com/AngelSGonza2107/ARQUI2B_1S2025_G16.git