

# Caso de Estudio

Andrea Camila Manangón Palacios  
*Ingeniería en Electrónica y Automatización*  
*Universidad de las Américas*  
Quito, Ecuador  
andrea.manangon@udla.edu.ec

Juan Andrés Serrano Granja  
*Ingeniería en Electrónica y Automatización*  
*Universidad de las Américas*  
Quito, Ecuador  
juan.serrano.granja@udla.edu.ec

**Abstract**—En este documento se presenta un estudio de caso enfocado en el monitoreo de la temperatura y humedad de una biblioteca, con el fin de reconocer variaciones en estos valores y en caso de ser necesario, corregirlos evitando así el deterioro de los libros. Para la elaboración de este proyecto se creó un sistema que permita monitorear estos valores en el transcurso del día mediante sensores DHT11. El uso de la base de datos Firebase, javascript y una interfaz en HTML fue fundamental para almacenar los valores y poderlos reflejar a los empleados para facilitar su posterior análisis. Estos sistemas son de gran utilidad al permitir tener un registro de los valores y debido a su fácil implementación pueden ser adaptados en espacios más grandes o incluso en diferentes áreas.

**Index Terms**—Firebase, HTML, javascript, DHT11, temperatura, humedad

## I. INTRODUCCIÓN

El cuidado y preservación de libros es dificultoso porque los materiales de los que están fabricados como el papel y cartón son delicados. Al ser expuestos a condiciones ambientales muy irregulares pueden deteriorarse con mucha facilidad. Dos factores esenciales que deben ser monitoreados son la temperatura y humedad con el fin de procurar mantener un ambiente controlado. Los sistemas de monitoreo son una herramienta muy eficaz y útil para poder observar cambios en determinadas condiciones o situaciones que requieran un seguimiento constante. Mediante estos se puede realizar análisis ya sea para mantener o mejorar algún aspecto específico.

Para comprender las diferentes funcionalidades del programa se deben tener en cuenta los siguientes conceptos fundamentales.

### A. Base de datos

Es la forma más usada en el mundo para recopilar y organizar información específica que requiera tener un respaldo. Sus inicios se remontan a 1960 en donde se usaron las primeras bases de datos de navegación, en donde se incluyen las de jerarquía y las de red sin embargo estos eran bastante sencillos y limitados. Alrededor de 1980 se popularizaron las bases de datos relacionales que consistían en organizar la información en forma de tablas con filas y columnas. Posteriormente en 1990 aparecieron las bases orientadas a objetos.

A partir de esa época este campo se expande constantemente adaptándose a las nuevas tecnologías y necesidades como son

una mayor velocidad de procesamiento y uso del internet. Dentro de ese grupo aparecen las bases de datos NoSQL, las de datos en la nube y las de autogestión. [1]

### B. Firebase

Es una plataforma en la nube de Google que sirve para el desarrollo de aplicaciones web y móvil. [2] Cuenta con un gran número de funciones pero para este caso de estudio se usaron las real-time database. Esta base de datos es NoSQL y ayudan a guardar los datos en tiempo real al alojarlos en la nube. Con esta aplicación se guardan los datos de la cantidad de personas que ingresan a la biblioteca y los valores de humedad y temperatura de cada sector.



Fig. 1. Firebase de Google

### C. HTML

Conocido también por sus siglas en inglés Lenguaje de Marcado de Hipertexto. Es la base para todos los sitios web, debido a que permite definir el contenido y la estructura básica. Es utilizado en conjunto con CSS, que permite definir diseños, colores, fondos, entre otros y javascript. Tiene sus inicios en 1980 cuando surgió la idea de tener un sistema de hipertexto que permita guardar la información y accederla en cualquier parte del mundo. De esta manera, [3]



Fig. 2. HTML

### D. Javascript

Es más utilizado como scripting para páginas web. Es un lenguaje de programación que se basa en prototipos y tiene soporte para programación orientada a objetos. [4]



Fig. 3. Javascript

### E. Objetivo general

Crear un sistema de monitoreo de temperatura y humedad con la ayuda de recursos tecnológicos como Firebase, HTML, Javascript con el fin de recopilar la información necesaria y poder realizar un análisis de los sectores que presentan datos anormales que pueden estar influyendo en el deterioro de los libros.

### F. Objetivos específicos

- Armar un circuito con los componentes necesarios que permitan monitorear la entrada y salida de personas, así como la humedad y temperatura.
- Crear un sistema de validación para poder limitar el ingreso al sistema de la biblioteca.
- Crea una interfaz gráfica que permita reflejar en tiempo real los datos que se van recopilando.
- Utilizar bases de datos para tener un registro de los datos que se generan al momento de implementar el sistema.

## II. DESARROLLO TÉCNICO

Para poder implementar los distintos aspectos de este sistema de monitoreo, fueron necesarias varias secciones de funcionalidad, específicamente:

- Monitoreo de temperatura y humedad
- Sistema de Validación
- Comunicación con ESP-NOW
- Entrada de datos Javascript
- Interfaz HTML
- Conexión con Firebase

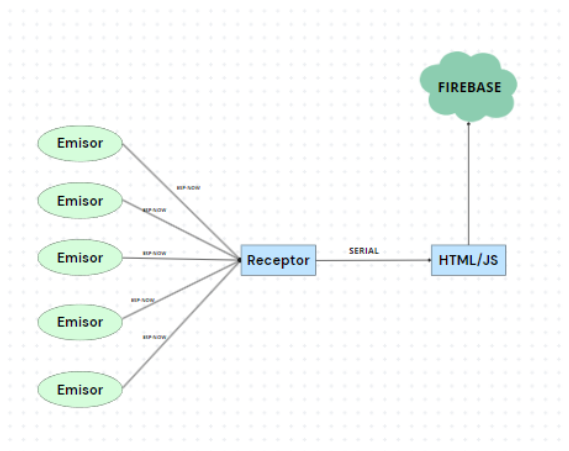


Fig. 4. Estructura de funcionamiento

### A. Monitoreo de temperatura y humedad

Para generar la lectura de estas variables, se utilizó el sensor DHT11, debido a su fácil manejo y relativo bajo costo, se utilizaron 6 sensores en total, uno dentro de cada zona específica de la biblioteca, ya que esto permite sectorizar las áreas de problema específicas, así como poder cubrir todo el volumen de estudio. Para esto fue necesario posicionar varias placas de desarrollo ESP32, cada una con su sensor específico, que emita los datos para estos poder ser guardados en la base de datos para ser analizados.

```
float h = dht.readHumidity();
float t = dht.readTemperature();
```

utilizando la biblioteca DHT integrada en el IDE de Arduino, es posible almacenar los valores de temperatura y humedad de cada sensor usando las funciones demostradas anteriormente.

### B. Sistema de Validación

Para asegurar que la lectura se realice de la forma mas efectiva posible, fue necesario utilizar un sistema de validación, que solo permita el envío de datos cuando se ingresa una clave específica. Esta funcionalidad se logró utilizando un teclado de membrana, así como una pantalla LCD, para poder manejar la validación de forma local.

En Arduino, existen librerías ya establecidas que permiten la fácil implementación de estos periféricos, de manera concreta, se destacan "LiquidCrystal-I2C", que maneja la LCD utilizando el protocolo I2C, y "Keypad" que maneja el teclado. A continuación, se puede encontrar la lógica de validación implementada:

```
if (!acceso) {
  if (key) {
    if (key == '<') {
      // Clear the input
      input = "";
      lcd.setCursor(0, 1);
      lcd.print(" ");
    } else if (key == '>') {
      if (input == pass) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Bienvenido");
        delay(2000);
        lcd.clear();
        acceso = true;
        pararLect = false;
        Serial.println("block");
      } else {
        intentos++;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Incorrecto");
        delay(2000);
        lcd.clear();
        if (intentos >= 3) {
          while (true) {
            lcd.setCursor(0, 0);
            lcd.print("Bloqueado");
            delay(5000);
            lcd.clear();
            lcd.print("Clave acceso:");
            input = "";
            intentos = 0;
            break;
          }
        }
      }
    }
  }
}
```

```

    } else {
        lcd.setCursor(0, 0);
        lcd.print("Clave acceso:");
    }
    input = "";
}
} else {
    if (input.length() < 16) {
        input += key;
        lcd.setCursor(input.length() - 1, 1);
        lcd.print(key);
    }
}
} else {
    if (key == ':') {
        pararLect = true;
        acceso = false;
        Serial.println("block");
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Lectura detenida");
        delay(2000);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Clave acceso:");
        input="";
    }

    if (!pararLect) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Leyendo datos...");
        lcd.setCursor(0, 1);
        lcd.print("* para apagar");
        if (enviarData){
            enviarData = false;

            float h = dht.readHumidity();
            float t = dht.readTemperature();
        }
    }
}

```

Se puede observar que se utiliza una variable de validación, que se activa cuando los valores agrupados ingresados por teclado son iguales al valor llamado pass, esto activa la lectura de temperatura y humedad, y se mantiene activo hasta que el usuario presiona la tecla \*, es importante notar que para la configuración del teclado se utilizó el siguiente esquema:

```

{ '1', '2', '3', '<' },
{ '4', '5', '6', '=' },
{ '7', '8', '9', '>' },
{ ':', '0', ';', '?' }

```

### C. Comunicación con ESP-NOW

ESP-NOW es un protocolo de comunicación que permite comunicar varios ESP32 de forma bidireccional, esto es útil ya que permite conectar todos los sensores a un ESP32 receptor, que agrupa los valores medidos en cada sector, para poder enviarlo a la base de datos, este protocolo se implementa en el código a continuación:

```

typedef struct struct_message {
    int id;
    float t;
    float h;
}struct_message;

```

El primer paso es crear una estructura de datos, que contenga el id de la placa emisora, y todos los valores que se desee enviar, esta estructura se maneja en todas las ESP32, incluida la receptora.

```

struct_message myData;
struct_message board1;
struct_message board2;
struct_message board3;
struct_message board4;
struct_message board5;
struct_message boardStruct[4] = {board1, board2, board3, board4, board5};

void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
    char macStr[18];

    memcpy(&myData, incomingData, sizeof(myData));
    // Update the structures with the new incoming data
    boardStruct[myData.id-1].t = myData.t;
    boardStruct[myData.id-1].h = myData.h;
}

```

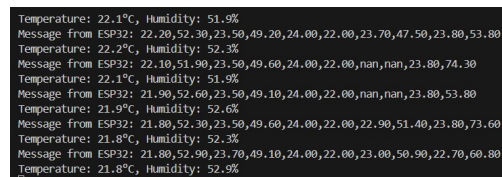
En esta sección se organizan los datos recibidos de cada placa, y se los agrupa en un array, para así poder acceder a los mismos.

```

Serial.print(t);
Serial.print(",");
Serial.print(h);
Serial.print(",");
Serial.print(boardStruct[0].t);
Serial.print(",");
Serial.print(boardStruct[0].h);
Serial.print(",");
Serial.print(boardStruct[1].t);
Serial.print(",");
Serial.print(boardStruct[1].h);
Serial.print(",");
Serial.print(boardStruct[2].t);
Serial.print(",");
Serial.print(boardStruct[2].h);
Serial.print(",");
Serial.print(boardStruct[3].t);
Serial.print(",");
Serial.println(boardStruct[3].h);
}

```

Finalmente, se imprimen los datos separados por comas en el monitor serial, de esta forma estos se pueden transmitir a otro programa para ser procesados.



```

Temperature: 22.1°C, Humidity: 51.9%
Message from ESP32: 22.20,52.30,23.50,49.20,24.00,22.00,23.70,47.50,23.80,53.80
Temperature: 22.2°C, Humidity: 52.3%
Message from ESP32: 22.10,51.90,23.50,49.60,24.00,22.00,nan,nan,23.80,74.30
Temperature: 22.1°C, Humidity: 51.9%
Message from ESP32: 21.90,52.60,23.50,49.10,24.00,22.00,nan,nan,23.80,53.80
Temperature: 21.9°C, Humidity: 52.6%
Message from ESP32: 21.80,52.30,23.50,49.60,24.00,22.00,22.90,51.40,23.80,73.60
Temperature: 21.8°C, Humidity: 52.3%
Message from ESP32: 21.80,52.90,23.70,49.10,24.00,22.00,23.00,50.90,22.70,60.80
Temperature: 21.8°C, Humidity: 52.9%

```

Fig. 5. Estructura de Salida de datos

### D. Entrada de datos Javascript

Como se observó en la imagen anterior, los datos enviados por monitor serial por el ESP32 pueden ser leídos a través de un servidor en Javascript, para esto, se utiliza la herramienta de Node.js, ya que permite generar aplicaciones en tiempo real y servidores. a continuación se presenta la estructura para escuchar los datos del ESP32:

```

const { SerialPort } = require('serialport');
const { ReadlineParser } =
require('@serialport/parser-readline');

```

```

const http = require('http');
const express = require('express');
const socketIO = require('socket.io');
const app = express();
const server = http.createServer(app);
const io = socketIO(server);

server.listen(3000, function () {
  console.log('Server listening on port', 3000);
});

app.use(express.static(__dirname + '/public'));

const port = new SerialPort({
  path: 'COM5',
  baudRate: 115200,
});

const parser =
port.pipe(new ReadlineParser({ delimiter: '\r\n' }));

port.on('open', () => {
  console.log('Serial port opened');
});

```

Se puede observar que se utiliza la libreria serialport, y toma el valor de COM específico que está utilizando la placa, de esta forma, cada vez que el E32 envía un mensaje al serial, javascript es capaz de captarlo y procesarlo. Se puede notar de la misma forma, que se utiliza un socket, para poder enviar los datos de forma segura entre el backend(javascript), y el programa HTML. el servidor en js envía los datos bajo la siguiente estructura, cada vez que el ESP32 actualiza los valores:

```

io.emit('sensorData', {
  temperatural,
  humedad1,
  temperatura2,
  humedad2,
  temperatura3,
  humedad3,
  temperatura4,
  humedad4,
  temperatura5,
  humedad5,
  temperatura6,
  humedad6,
  contadorPulsadores,
  activo
});

```

Los pulsadores se refieren a valores acumulados de entradas y salidas de las puertas, estos tienen una lógica por separado, que se las puede encontrar en los anexos, pero que no entran en el rango de este análisis.

### E. Interfaz HTML

Para generar esta interfaz, se debe tener en cuenta el aspecto estético, y el aspecto funcional, a continuación se presenta la explicación de ambos:

1) *Estético*: Se utilizó CSS para realizar la estructura estética de cada elemento en la interfaz, específicamente, se notan los siguientes elementos:

- Banner
- Contador de entradas
- Secciones interactivas
- Gráficas

Para el código en CSS, se generaron estructuras estilizadas para cada uno de los componentes mencionados anteriormente, las estructuras container, library—layout, section, charts e



Fig. 6. Estructura servidor web

imagen—fondo manejan los elementos gráficos, luego, stos se implementan a través de divisiones en html para acomodarlos tal y como se muestra en la figura anterior.

Section tiene un comportamiento especial, ya que se utiliza varias veces con diferentes configuraciones, para cada una de las habitaciones en la biblioteca, se itera para crear las diferentes secciones de la siguiente manera, ajustando su posición y tamaño dentro de container:

```

#section1 {top:10px;left:10px;width:180px;height:140px;}
#section2 {top:10px;left:210px;width:180px;height:140px;}
#section3 {top:160px;left:10px;width:380px;height:70px;}
#section4 {top:240px;left:10px;width:180px;height:140px;}
#section5 {top:390px;left:10px;width:180px;height:140px;}
#section6 {top:390px;left:210px;width:180px;height:140px;}

```

Para la parte funcional del código, primero se utilizan las etiquetas let, const y var para definir los elementos a colocar, como las gráficas ed humedad y temperatura, así como ajustar los valores iniciales de los cuartos, y finalmente generar una variable en donde aperturar el socket creado por el código en Javascript. de la misma manera, se generan funciones que permitan el uso de la información recibida, como es el caso de toggleSection(), que activa visualmente los datos de cada sección, actualizarSecciones(), que actualiza los datos de temperatura y humedad de cada habitación, y finalmente loadSectionGraph, que actualiza en tiempo real los gráficos de temperatura y humedad de cada habitación cuando se da click a su botón respectivo, esta última igualmente tiene funcionalidad con firebase, por lo que se explora más adelante.

```

function actualizarSecciones() {
  for (let sectionId in lecturas) {
    const seccionRef =
    firebase.database().ref(Cuartos/${sectionId});
    seccionRef.on("value", (snapshot) => {
      const data = snapshot.val();
      const tempKeys = Object.keys(data.temp || {});
      const humKeys = Object.keys(data.hum || {});
      const lastTempKey = tempKeys[tempKeys.length - 1];
      const lastHumKey = humKeys[humKeys.length - 1];

      const temp = data.temp?.[lastTempKey] || "- -";
      const hum = data.hum?.[lastHumKey] || "- -";

      if (estadoSecciones[sectionId]) {
        const sectionElement =
        document.getElementById(sectionId);
        sectionElement.innerText =
        Sección ${sectionId.replace('section',
        '')}\n${temp}°C,

```

```

    ${hum}%};
  }
}, (error) => {
  console.error(Error al leer datos de la sección
    ${seccionId}:, error);
});
}
}

```

En el código anterior se muestra la funcionalidad de actualizarSecciones, se puede notar que esta función llama a cada valor de temperatura de data, que representa los valores enviados por el socket, y se los utiliza para actualizar la parte visual de cada sección, de la misma manera, en el caso de haber un error, se utiliza la consola para registrar los movimientos.

### F. Conexión con Firebase

Para poder vincular los datos a la nube, se utiliza Firebase, donde se creó un elemento llamado "Realtime Database", el mismo que permite almacenar hasta 1GB de información. En la práctica, esta base de datos se utiliza para almacenar 3 tipos de variables:

- Temperatura y Humedad de cada sección
- Contadores de entradas y salidas
- Estado del sistema de monitoreo

Para entender el funcionamiento de esta vinculación, se explica a continuación la implementación para almacenar los datos de las habitaciones.

```

const firebaseConfig = {
  apiKey: "AIzaSyAgjuRFQ_tcdZCCkjrA838yc2uwq2BJ3RY",
  authDomain: "monitoreo-5d03a.firebaseio.com",
  databaseURL:
    "https://monitoreo-5d03a-default-rtdb.firebaseio.com",
  projectId: "monitoreo-5d03a",
  storageBucket: "monitoreo-5d03a.firebaseio.com",
  messagingSenderId: "665390581550",
  appId: "1:665390581550:web:a597bf82743a9a48529cbf",
  measurementId: "G-JMJDTZ98JX"
};

```

Para inicializar la vinculación, se utiliza la llave de acceso de Firebase, esta llave permite que cualquier sistema se conecte a la misma base de datos, esto es útil para poder expandir el proyecto en el caso de que sea necesario.

Una vez inicializada la conexión, se crea la función guardarDatosSeccion, en donde se adjunta una referencia general denominada "Cuartos".

```

var cuartosRef = firebase.database().ref("Cuartos");
var seccionRef = cuartosRef.child(seccionId);

```

Esto permite crear una sección en firebase donde se almacene cada tipo de dato por separado, con etiquetas pertinentes. Se activa un contador que, cada vez que se recibe un dato de la ESP32, crea un nuevo child con el número actual del contador como nombre, donde se almacena el valor, como se muestra en el siguiente código:

```

seccionRef.child("temp").child(contador).set(nuevaTemp)
  .then(() => console.log(Temperatura guardada para
    ${seccionId}:, nuevaTemp))
  .catch((error) => console.error(Error al guardar
    temperatura para ${seccionId}:, error));

```

De esta forma, el código permite generar datos en firebase que se puedan actualizar en tiempo real, y se puede de la misma forma visualizarlos en HTML en forma de gráfico en el tiempo.



Fig. 7. Estructura base de datos

Una vez implementado el sistema completo, se procedió a realizar las mediciones en las distintas habitaciones para entender el comportamiento de la temperatura y humedad, y diagnosticar la problemática, en los anexos se puede encontrar el video de la toma de datos respectivos.

## III. RESULTADOS

A continuación se presentan los gráficos de cada una de las secciones: Se puede observar que en las zonas 3 y

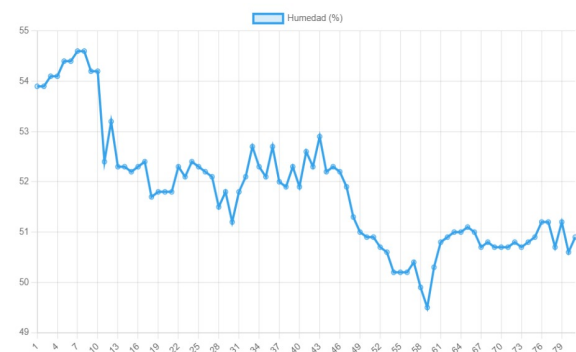
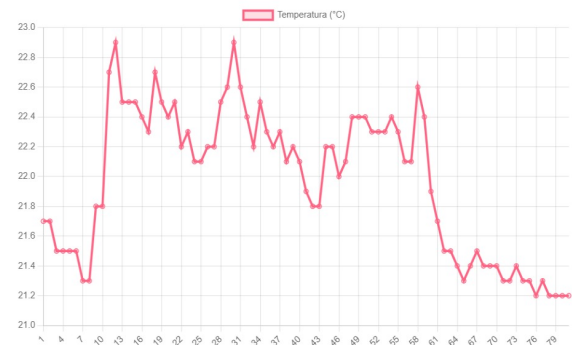


Fig. 8. Resultados zona 1



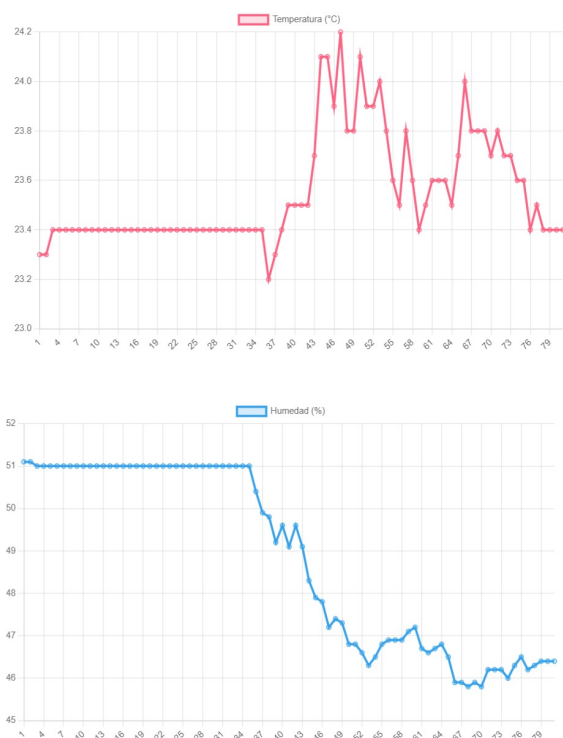


Fig. 9. Resultados zona 2

4, se generaron picos de temperatura, aumentando mas de 4 grados en cada caso, de la misma manera, existe un aumento en el porcentaje de humedad en el mismo intervalo, lo que da a entender que hubo una anomalia de condiciones en estos momentos.

De la misma forma se puede observar que el resto de las zonas mantuvieron una temperatura relativamente constante, ed aproximadamente 23 grados. de estas la habitación mas estable fue la seccion 2, que mantuvo un desarrollo casi lineal durante la mayoría de su trayectoria.

De forma general, se pueden obtener los siguientes datos de desviación y de desarrollo para cada una de las zonas.

### CONCLUSIONES

La implementación de un sistema de monitoreo es fundamental para conocer, registrar y analizar las condiciones de temperatura y humedad que existen en cada zona de la biblioteca, con estos datos se pudo evidenciar cuales son las zonas que se encuentran afectadas. En el caso de las zonas 3 y 4 se evidenciaron 2 picos de temperatura altos en donde llegaron a valores que superaron los 4 grados de diferencia. Al haber un aumento de temperatura, los niveles de humedad ascendieron consecuentemente. Estos dos factores son cruciales para los materiales de los libros, en especial el porcentaje de humedad. Al sobrepasar los límites normales este factor puede contribuir al desgaste de las hojas, manchas en las tapas d elos libros y en su interior, amarillamiento en



Fig. 10. Resultados zona 3

los bordes, entro otros. Si estos valores no son controlados con el tiempo, todos los deterioros mencionados anteriormente se aceleran causando la pérdida de libros con mayor frecuencia.

Aparte de estas dos zonas, las demás presentaron variaciones menores a 2 grados lo cual no significa una fluctuación importante que pueda causar algún problema a futuro con los libros. En este caso se recomienda mantener estas condiciones estables para evitar que se pueda producir consecuencias a largo plazo. Sin embargo, si se encuentran libros que presentan desgastes visibles, sería importante revisar otras factores aparte de los ya analizados, como la frecuencia en que los clientes los ocupan, la limpieza, entre otros.

A través de esta implementación, se pudo evidenciar que en efecto existía un fenomeno que afectaba la condición de los libros, es importante entender que estas afectaciones, si bien son difíciles de detectar de forma manual, tienen un impacto grande en la capacidad de la biblioteca de mantener una condición óptima de sus recursos. Para poder determinar de forma certera el origen de esta falla, se recomienda realizar una inspección general de las zonas afectadas, específicamente de conexiones eléctricas, entradas de aire o luz, y ventilación, para de esta forma poder determinar la causa directa.

### REFERENCES

- [1] Oracle, "What is a Database?," *Oracle*. [En línea]. Disponible en: <https://www.oracle.com/mx/database/what-is-database/#relational>. [Accedido: 23-nov-2024].

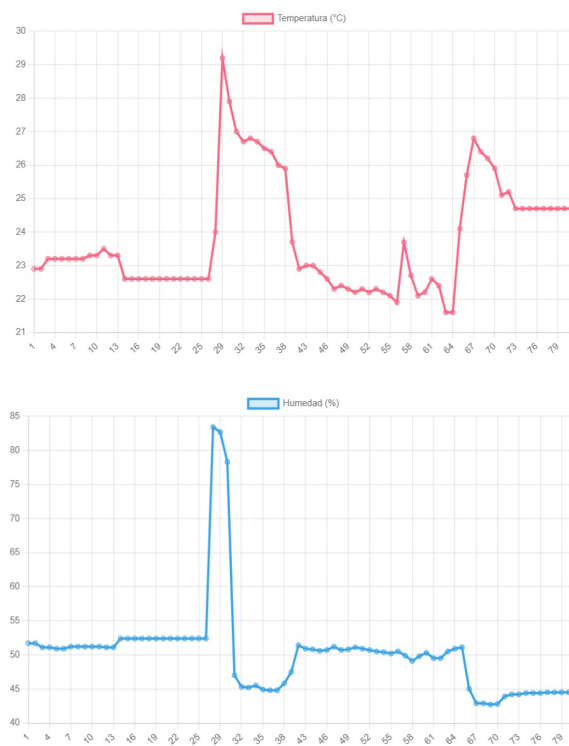


Fig. 11. Resultados zona 4

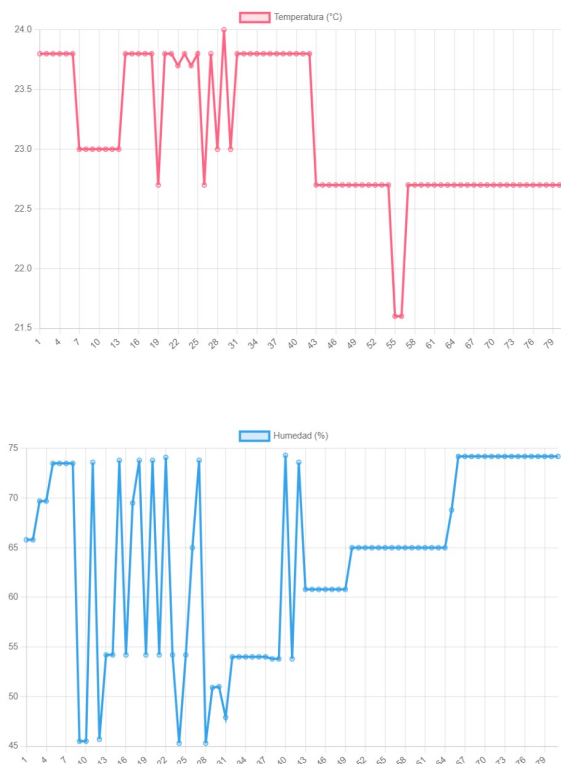


Fig. 12. Resultados zona 5

- [2] Digital55, “¿Qué es Firebase? Funcionalidades, ventajas y conclusiones,” *Digital55*. [En línea]. Disponible en: <https://digital55.com/blog/que-es-firebase-funcionalidades-ventajas-conclusiones/>. [Accedido: 23-nov-2024].
- [3] Vadavo, “HTML: Qué es y para qué sirve,” *Vadavo*. [En línea]. Disponible en: <https://www.vadavo.com/blog/html-que-es-y-para-que-sirve/>. [Accedido: 23-nov-2024].
- [4] Mozilla, “JavaScript,” *MDN Web Docs*. [En línea]. Disponible en: <https://developer.mozilla.org/es/docs/Web/JavaScript>. [Accedido: 23-nov-2024].

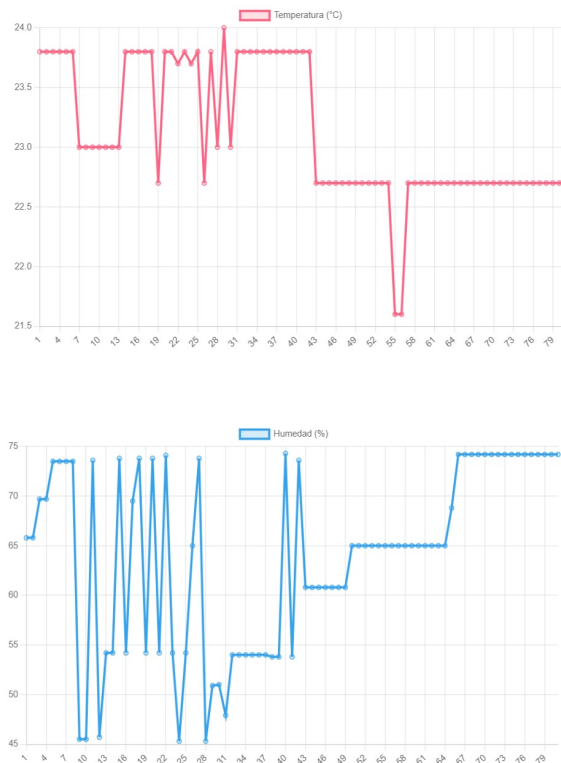


Fig. 13. Resultados zona 6

	t1	h1
Mayor	22.90	54.6
desviación estandar	0.490411	1.198727
Promedio	21.96742	51.80372
Menor	21.20	49.5

Fig. 14. Analisis zona 1

	t2	h2
Mayor	24.20	51.1
desviación estandar	0.222548	2.139749
Promedio	23.54712	48.77201
Menor	23.2	45.8

Fig. 15. Analisis zona 2

	t3	h3
Mayor	34	74
desviación estandar	2.217982	10.71099
Promedio	24.83036	22.0969
Menor	20	18

Fig. 16. Analisis zona 3

	t6	h6
Mayor	24	74.3
desviación estandar	0.57031	9.29876
Promedio	23.12021	63.35147
Menor	21.6	45.3

Fig. 19. Analisis zona 6

	t4	h4
Mayor	29.2	83.4
desviación estandar	1.686259	7.069834
Promedio	23.73069	49.84353
Menor	21.6	42.7

Fig. 17. Analisis zona 4

	t5	h5
Mayor	24	74.3
desviación estandar	0.57031	9.29876
Promedio	23.12021	63.35147
Menor	21.6	45.3

Fig. 18. Analisis zona 5