



Resumen

La humedad es la cantidad de agua presente en el aire de un determinado espacio. En lugares reducidos como baños, el vapor generado por agua caliente sin una adecuada ventilación origina la aparición de manchas negras en techos y paredes, proliferación de moho, corrosión y daño en equipos mecánicos o electrónicos.

Los efectos de la humedad y la temperatura pueden provocar daños estéticos, estructurales y en la salud, especialmente a personas con problemas respiratorios.

La necesidad de regular los ambientes se convierte en la mejor forma de asegurar el confort y el bienestar de nuestra salud. Y para esta tarea, el sensor de humedad y temperatura resultan elementos claves en cualquier espacio de domótica.

ÍNDICE

Glosario.....	3
Introducción.....	5
Metodología.....	7
Códigos implementados	12
Código.py	13
Database.py	17
Comunicación.py	19
Proteus.....	20
Base de Datos.....	20
Sensor_bateria.sql.....	20
Sensor_humedad.sql.....	22
Resultados	23
Circuito electrónico.....	23
BASE DE DATOS EN MYSQL	25
CÓDIGO IMPLEMENTADO EN VSCODE.....	25
GRÁFICOS DE RELACIÓN.....	27
Conclusión.....	29
Webgrafía	31
ANEXOS	32

Glosario

Humedad: Cantidad de vapor de agua presente en el aire o en un material. Se mide generalmente como un porcentaje de la cantidad máxima de vapor que el aire puede contener a una temperatura determinada (humedad relativa).

Relé: Dispositivo electromecánico que actúa como interruptor controlado eléctricamente. Se utiliza para abrir o cerrar circuitos eléctricos mediante una señal de bajo voltaje.

Arduino Uno: Placa de desarrollo basada en un microcontrolador ATmega328P. Es utilizada para proyectos de electrónica y programación, permitiendo el control de sensores y actuadores.

VCC: Voltaje de alimentación o de entrada en un circuito. Generalmente representa el voltaje positivo necesario para que un componente electrónico funcione.

DATA: Señal o línea utilizada para transmitir información digital entre dispositivos o componentes electrónicos.

IN: Terminal o pin de entrada de un dispositivo electrónico, a través del cual se recibe una señal o corriente.

COM: Terminal común en un relé o interruptor. Es el punto de conexión que puede conectarse a otros terminales, como NO (Normalmente Abierto) o NC (Normalmente Cerrado).

GND: Sigla de "Ground" o tierra. Es la referencia de voltaje en un circuito eléctrico y sirve como punto de retorno para la corriente.

NO (Normalmente Abierto): Contacto de un relé que permanece desconectado en su estado de reposo. Solo se conecta cuando se activa el relé.

Resistor: Componente electrónico que limita el paso de corriente en un circuito. Se mide en ohmios y es utilizado para proteger otros componentes.

LED: Sigla de "Light Emitting Diode" (Diodo Emisor de Luz). Componente que emite luz al ser atravesado por una corriente eléctrica en el sentido adecuado.

MYSQL: sistema de gestión de bases de datos relacional (RDBMS) basado en SQL (Structured Query Language). En este proyecto, MySQL fue empleado para gestionar y almacenar los datos generados por la aplicación.

GIT: Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear cambios en el código fuente a lo largo del tiempo. Git facilita el trabajo en equipo al permitir que varios colaboradores trabajen en el mismo proyecto de manera simultánea, con la capacidad de fusionar y revertir cambios.

GITHUB: GitHub es una plataforma basada en la web que permite a los desarrolladores almacenar y gestionar su código fuente utilizando Git, un sistema de control de versiones. GitHub ofrece herramientas colaborativas como control de versiones distribuido, gestión de repositorios y automatización de flujos de trabajo.

VSCODE: Editor de código fuente gratuito y multiplataforma desarrollado por Microsoft. Durante este proyecto, VSCode fue utilizado como entorno principal de desarrollo.

MATPLOTLIB: Biblioteca de Python utilizada para crear visualizaciones y gráficos 2D de alta calidad. En este proyecto, Matplotlib fue empleada para crear gráficos a partir de los datos de la aplicación, facilitando la representación visual de variables como la humedad y la temperatura dentro de VSCode.

Introducción

Los sensores de humedad y temperatura tienen multitud de aplicaciones en el mundo de la jardinería, la agricultura, construcción y meteorología, tales como:

- Comprobar porcentajes de humedad presente en el suelo, ajustando eficazmente el riego para que se realice de manera adecuada.
- Medir la humedad de los materiales de construcción, permitiendo detectar el momento en el que han terminado de fraguar para seguir trabajando con ellos.
- Prevenir inundaciones y fugas de agua en distintos ambientes.

Sin embargo, cada vez son incluidos con mayor asiduidad en los sistemas de climatización domésticos permitiendo establecer con precisión el porcentaje de humedad ambiental y temperatura. Por lo tanto, el objetivo general del presente trabajo es implementar un sistema IoT para el monitoreo de variables de temperatura y humedad en ambientes reducidos a través de un sensor de tamaño pequeño. Por objetivo específico se plantea el diseño e implementación de un sistema de captura y visualización de datos que permita el monitoreo permanente de las variables.

Debido a que el campo de aplicación del presente proyecto es en hogares, se plantea que el mismo sea rentable y de fácil implementación, que el consumo energético sea reducido y mantenga al mismo tiempo la capacidad de obtener mediciones precisas y fiables, siendo las razones que justifican su aplicación:

- Su sencillez en sus mecanismos de funcionamiento.
- Fácil de calibrar, económicos y duraderos.
- Capacidad de controlar y ajustar los niveles de humedad y temperatura deseados.

- Medir la temperatura y avisar ante aumentos bruscos que superen los umbrales de confort.
- Alertar inmediatamente cuando el nivel de humedad alcanza niveles que no son recomendables para la salud.

Metodología

De acuerdo con lo solicitado por la empresa “Dispositivos Inteligentes SRL”, se llevó a cabo la elección de los componentes de un dispositivo que recaudará información y en base a ella accionará un actuador. El mismo fue elegido luego de una exhaustiva búsqueda e investigación que se realizó en las inmediaciones de la ciudad de Córdoba Capital.

La investigación fue motivada por una necesidad que se encuentra en los hogares de esta provincia, debido a su clima en las estaciones primaverales y de verano existe un factor que genera daños a la salud y a las instalaciones inmobiliarias, hablamos de la humedad.

Por lo que se hará uso del sensor DHT11 que mide la humedad y la temperatura, un extractor de aire, un LED, un Relé, una batería y un convertidor.

La elección del uso del relé se realizó para no comprometer la seguridad de la ESP32. El mismo controlará al extractor que estará alimentado directamente desde una toma de corriente de 220V, facilitando las conexiones y configuraciones que se requieran en el circuito para que el dispositivo funcione con éxito.

A continuación, se describen los detalles de los componentes y las conexiones eléctricas:

❖ Componentes:

- Microcontrolador ESP32 (Microcontrolador que recibe y procesa los datos del sensor y en base a los resultados acciona un actuador)
- Sensor DHT11 (mide temperatura y humedad)
- Actuador (extractor de aire que elimina el exceso de humedad en el

baño)

- LED (indicador visual del estado)
- Relé de 5V(Controla el encendido y apagado del extractor de aire)
- Batería LiFePO4 (proporciona energía)

❖ Conexiones eléctricas:

Para las simulaciones del proyecto se utilizó el software de Proteus y Arduino Ide. Ambos fueron de gran utilidad a la hora del análisis de viabilidad del proyecto. Debido a las limitaciones propias de los simuladores se utilizó Arduino Uno en lugar de ESP32.

A continuación, se describen las conexiones realizadas.

- Sensor DHT11 al ESP32:
 - Pin de datos (DATA): Se conecta a un pin digital del ESP32. Este es el pin que usará el ESP32 para leer la información de humedad y temperatura.
 - Pin VCC (Alimentación): Se conecta al pin de 3.3V del ESP32.
 - Pin GND (Tierra): Se conecta al pin GND del ESP32.

- Actuador (Extractor de aire 220V)-Resistor:

Por sus características energéticas se conectará a una fuente de alimentación externa que estará ubicado en el baño para que cumpla su función.

- Relé-Conexiones al extractor y a la ESP32:
 - El pin IN del módulo de relé se conecta al pin GPIO 7 del ESP32.
 - El pin VCC del relé se conecta a 5V.
 - El pin GND del relé se conecta a GND del ESP32.

- En el lado de alta potencia del relé:
 - ♦ El terminal COM del relé se conecta a la fase de la fuente externa de 220V que alimenta el extractor.
 - ♦ El terminal NO (normalmente abierto) del relé se conecta a uno de los cables del extractor, mientras que el otro cable del extractor se conecta a neutro.
- Conexión del LED al ESP32:
 - El pin positivo del LED se conecta a una resistencia (220Ω) y luego al pin GPIO 6 del ESP32.
 - El pin negativo del LED se conecta a un resistor de 220V y éste al GND del ESP32.
- Conexión de la batería LiFePO4:
 - La batería LiFePO4 se conecta al convertidor boost, que eleva el voltaje de 3.2V a 5V.
 - La salida de 5V del convertidor se conecta al pin VIN del ESP32 y a VCC del módulo de relé.

Para proteger el sistema de la humedad del baño y de otros factores de hará uso de una caja de conexiones plástica que permitirá albergar los componentes electrónicos sin riesgo de daño.

❖ Características técnicas del sensor DHT11:

El DHT11 es un sensor digital de temperatura y humedad relativa de bajo costo y de fácil implementación con cualquier microcontrolador, de bajo consumo de energía y excelente estabilidad a largo plazo.

Integra un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos (no posee salida analógica), trabaja con un rango de medición de temperatura de 0 a 50 °C con precisión de ± 2.0 °C y un rango de humedad de 20% a 90% RH con precisión de 4% RH.

Los ciclos de lectura deben ser como mínimo 1 o 2 segundos, pero debido a que la temperatura y la humedad son variables que no cambian muy rápido en el tiempo no se considera una desventaja para este proyecto.

A nivel de software se dispone de librerías para Arduino con soporte para el protocolo "Single bus". En cuanto al hardware, solo es necesario conectar el pin VCC de alimentación a 3-5V, el pin GND a Tierra (0V) y el pin de datos a un pin digital del microcontrolador.

El protocolo de comunicación entre el sensor y el microcontrolador emplea un único hilo o cable, la distancia máxima recomendable de longitud de cable es de 20m., de preferencia

Especificaciones técnicas:

- Voltaje de Operación: 3V - 5V DC
 - Rango de medición de temperatura: 0a 50 °C
 - Precisión de medición de temperatura: ± 2.0 °C
 - Resolución Temperatura: 0.1°C
 - Rango de medición de humedad: 20% a 90% RH.
 - Precisión de medición de humedad: 5% RH.
 - Resolución Humedad: 1% RH
 - Tiempo de sensado: 1 seg.
 - Interfaz digital: Single-bus (bidireccional)
 - Modelo: DHT11
 - Dimensiones: 16*12*5 mm
 - Peso: 1 gr.
 - Carcasa de plástico celeste
- PINES
- 1- Alimentación: +5V (VCC)
 - 2- Datos (DATA)
 - 3- No Usado (NC)
 - 4- Tierra (GND)

Códigos implementados

Main.py

```
Entrega 2 > main.py > ...
1  """Programa para gestionar la BD e iniciar objetos
2  de la clase codigo"""
3
4  from database import Database
5  from codigo import Sensor
6
7  def main():
8
9      def principal():
10         """metodo inicio programa
11         Args: eleccion(int): Menú
12         """
13         db=Database("localhost", 3306,"root","root","sensor")
14         sensor1=Sensor(db)
15         print("BIENVENIDO")
16         sensor1.opcionesMenuPrincipal()
17
18
19  if __name__=="__main__":
20     main()
21
22
```

Imagen 1:Main.py

Código.py

```
1  import datetime
2  import time
3  import matplotlib.pyplot as plt
4
5  class Sensor():
6
7      def __init__(self, db) -> None:
8          self.humedad=0
9          self.bateria=0
10         self.db=db
11
12     def opcionesMenuPrincipal(self):
13         """funcion que despliega el menu para el usuario"""
14
15         while True:
16
17             print("-----")
18             eleccion=int(input("Seleccione la opción deseada: \n"
19                 " 1. Registro de Humedad y carga de bateria\n"
20                 " 2. Visualización gráfica de registros de humedad \n"
21                 " 3. Mostrar estado de la bateria\n"
22                 " 4. Cerrar conexión\n"))
```

```

23
24         if eleccion==1:
25             self.registrosHumedad()
26         elif eleccion==2:
27             self.visualizacionGrafica()
28         elif eleccion ==3:
29             self.bateriaEstado()
30         elif eleccion==4:
31             self.db.cerrarConexion()
32             break
33         else:
34             print("ingresó un dato incorrecto")
35
36
37
38
39  ✓ def registrosHumedad(self):
40     """funcion que carga valores de humedad del sensor y carga de la bateria a la BD"""
41
42     print("ingreso de los valores de humedad, voltaje bateria y carga de la bateria o escriba 0,0,0 para volver al menú anterior")
43
44     while True:
45
46         def recibir_datos():
47             datos = input("Ingrese informacion\n")

```

```

48             return datos.split(',')
49
50         datos=recibir_datos()
51
52         valorCargador=0
53
54         if datos:
55             humedad, voltaje, bateria = map(float, datos)
56
57             if (humedad or voltaje or bateria) ==0:
58                 break
59
60
61             print(f'humedad: {humedad}, voltaje: {voltaje}, bateria: {bateria}')
62             time.sleep(3) # Espera 3 segundos antes de la próxima lectura
63
64
65             hora=datetime.datetime.now()
66             sql=("INSERT INTO humedad(fecha, valores) VALUES (%s,%s)")
67             contenido=(hora, humedad)
68             self.db.ejecutarConsultas(sql,contenido)

```

```

69
70         valorCargador=0
71
72         if (bateria <=27.0):
73             valorCargador=1
74             sql1=("INSERT INTO bateria(cargador, carga)VALUES(%s,%s)")
75             datosA=(valorCargador, bateria)
76             self.db.ejecutarConsultas(sql1,datosA)
77         else:
78             sql1=("INSERT INTO bateria(cargador, carga)VALUES(%s,%s)")
79             datosA=(valorCargador, bateria)
80             self.db.ejecutarConsultas(sql1,datosA)
81

```

```

84     # efunciones se refieren al estado de la bateria
85
86
87     def bateriaEstado(self):
88         """metodo para buscar el estado de la bateria"""
89
90         sql=("SELECT cargador FROM bateria ORDER BY idbateria DESC LIMIT 1")
91         sql1=("SELECT carga FROM bateria ORDER BY idbateria DESC LIMIT 1")
92
93         res=self.db.obtenerResultados(sql)
94         res2=self.db.obtenerResultados(sql1)
95
96         car1=res[-1]
97         car=car1[0]
98
99         res1=res2[-1]
100        res=res1[0]
101
102        if car ==0:
103            print("La batería no esta recibiendo carga")
104        elif car==1:
105            print("La batería esta recibiendo carga")
106        if res >=40.0:
107            print("La bateria esta por encima o igual a 60%")
108        elif res<=27.0:
109            print("La carga de la batería esta por debajo o igual a 15%")

```



```
109         print("La carga de la batería esta por debajo o igual a 15%")
110     else:
111         print("La batería no está funcionando correctamente")
112
113
114
115     def visualizacionGrafica(self):
116
117         print("Registros de los valores de humedad")
118         sql = ("SELECT valores FROM humedad1")
119         valores = self.db.obtenerResultados(sql)
120         aplanada = [elemento for tupla in valores for elemento in tupla]
121         print(aplanada)
122
123         sql = ("SELECT idhumedad FROM humedad1")
124         valoresA = self.db.obtenerResultados(sql)
125         print(valoresA)
126         aplanadaA = [elemento for tupla in valoresA for elemento in tupla]
127
128
129         fig, ax = plt.subplots()
130         ax.plot(aplanadaA, aplanada, label='Humedad (%)', color='b')
131         ax.set_ylabel('Humedad (%)')
132         ax.set_xlabel('Tiempo')
133
134
135         plt.show()
136
137         self.opcionesMenuPrincipal()
138
139
```

Imagen2: Código.py

Database.py

Entrega 2 > database.py > ...

```
1  """Clase que permite la conexión con la BD y métodos para ejecutar
2  consultas, con la base de datos
3  """
4  import mysql.connector
5  from mysql.connector import Error
6
7
8
9  class Database:
10
11      def __init__(self, host, port, user, password, db):
12
13          try:
14              self.conexion=mysql.connector.connect(
15                  host=host,
16                  port=port,
17                  user=user,
18                  password=password,
19                  db=db
20              )
21              self.cursor=self.conexion.cursor()
22              print("Conexion exitosa")
23              self.cursor.execute("SELECT database();") #da el nombre de la BD
24              registro=self.cursor.fetchone()
25              print("Conectado a la BD:", registro)
26
27          except Error as ex:
28              print("Error de conexion", ex)
29              self.conexion=None
30              self.cursor=None
31
32      #Metodos para (INSERT, UPDATE, DELETE) datos
33      def ejecutarConsultas(self, consulta, valores=None):
34          self.cursor.execute(consulta, valores)
35          self.conexion.commit()
```

```
36
37     #Metodo para consultar SELECT y retorna resultados
38     def obtenerResultados(self, consulta, valores=None):
39         self.cursor.execute(consulta, valores)
40         return self.cursor.fetchall()
41
42     def cerrarConexion(self):
43         print("        ¡Gracias por utilizar la aplicación!")
44         self.cursor.close()
45         self.conexion.close()
46         print("Se cerró la conexión con la Base de Datos")
47
48     def ultimoRegistro(self):
49         self.cursor.lastrowid
50
51
52     #db=Database("localhost", 3306,"root","root","sensor")
53     #print(db)
54
```

Imagen 3: *Database.py*

Comunicación.py

```
1
2
3
4
5     import serial
6     import time
7
8
9
10    # Configura el puerto serial
11    puerto = 'COM3' # Cambia por '/dev/ttyUSB0' o por algo 'COM3'
12    baudios = 9600
13    #ser = serial.Serial(puerto, baudios, timeout = 1)
14
15
16
17    def recibir_datos():
18        #if ser.in_waiting > 0:
19        #datos = ser.readline().decode('utf-8').strip()
20        datos = input("Ingrese informacion\n")
21        return datos.split(',')
22
23    while True:
24        datos = recibir_datos()
25        if datos:
26            humedad, bateria = map(float, datos)
27            print(f'humedad: {humedad}, temperatura: {bateria}')
28
29    time.sleep(1) # Espera 1 segundo antes de la próxima lectura
30
31
32    # Cierra el puerto serial al finalizar
33    ser.close()
```

Imagen 4: Comunicaciones.py

Proteus

De acuerdo a lo solicitado se adjunta el **ANEXO-BRAVO**, en el cual se puede observar a detalle el código utilizado en ARDUINO IDE.

Base de Datos

Para la realización de la base de datos se utilizó MySQL. Para este programa se computó a base de datos compuesta por dos tablas, sensor_bateria.sql, sensor_humedad.sql. A continuación, se detalla cada uno:

Sensor_bateria.sql

```

18  --
19  -- Table structure for table `bateria`
20  --
21
22  • DROP TABLE IF EXISTS `bateria`;
23  • /*!40101 SET @saved_cs_client      = @@character_set_client */;
24  • /*!40101 SET character_set_client = utf8 */;
25  • CREATE TABLE `bateria` (
26      `idbateria` int(11) NOT NULL AUTO_INCREMENT,
27      `cargador` int(1) DEFAULT NULL,
28      `carga` float DEFAULT NULL,
29      PRIMARY KEY (`idbateria`)
30  ) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=latin1;
31  • /*!40101 SET character_set_client = @saved_cs_client */;
32
33  --
34  -- Dumping data for table `bateria`
35  --
36
37  • LOCK TABLES `bateria` WRITE;
38  • /*!40000 ALTER TABLE `bateria` DISABLE KEYS */;
39  • INSERT INTO `bateria` VALUES (1,1,60),(2,1,42),(3,1,40),(4,0,60),(5,1,77),(6,1,55),(7,1,23),(8,1,44),(9,1,20),(10,0,74),(11,1,95),(12,0,11),(13,1,41),(14,0,77);
40  • /*!40000 ALTER TABLE `bateria` ENABLE KEYS */;
41  • UNLOCK TABLES;
42  • /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

```

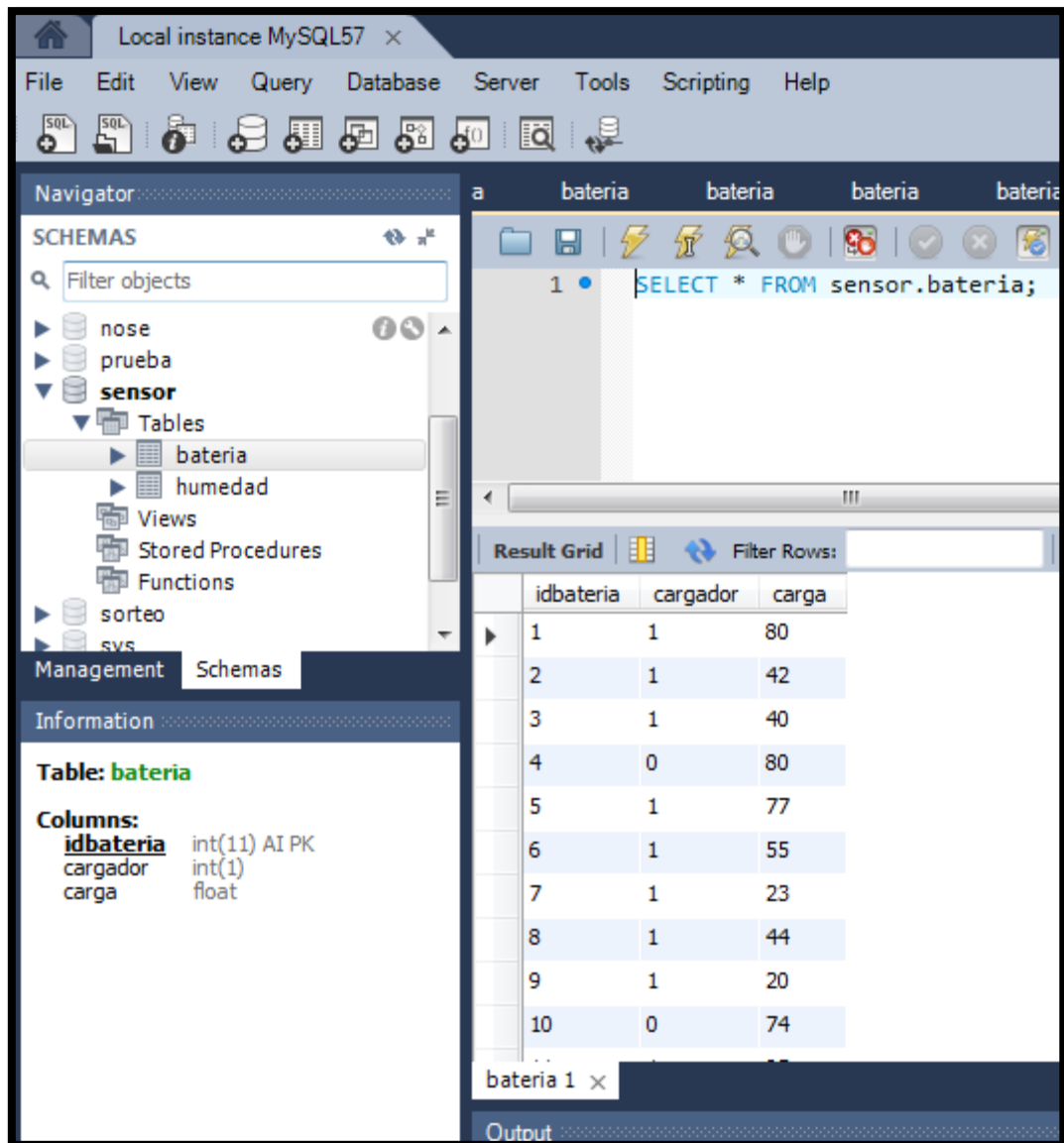


Imagen 4: Detalles del contenido del archivo *Sensor_bateria.sql*

Sensor_humedad.sql

```

17
18 --
19 -- Table structure for table `humedad`
20 --
21
22 • DROP TABLE IF EXISTS `humedad`;
23 • /*!40101 SET @saved_cs_client      = @@character_set_client */;
24 • /*!40101 SET character_set_client = utf8 */;
25 • CREATE TABLE `humedad` (
26   `idhumedad` int(11) NOT NULL AUTO_INCREMENT,
27   `fecha` datetime(6) DEFAULT NULL,
28   `valores` float DEFAULT NULL,
29   PRIMARY KEY (`idhumedad`)
30 ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;
31 • /*!40101 SET character_set_client = @saved_cs_client */;
32
33 --
34 -- Dumping data for table `humedad`
35 --
36
37 • LOCK TABLES `humedad` WRITE;
38 • /*!40000 ALTER TABLE `humedad` DISABLE KEYS */;
39 • INSERT INTO `humedad` VALUES (1, '2024-10-01 23:47:19.336960',10), (2, '2024-10-01 23:47:21.160065',30), (3, '2024-10-01 23:47:22.896164',25);
40 • /*!40000 ALTER TABLE `humedad` ENABLE KEYS */;
41 • UNLOCK TABLES;
42 • /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
43

```

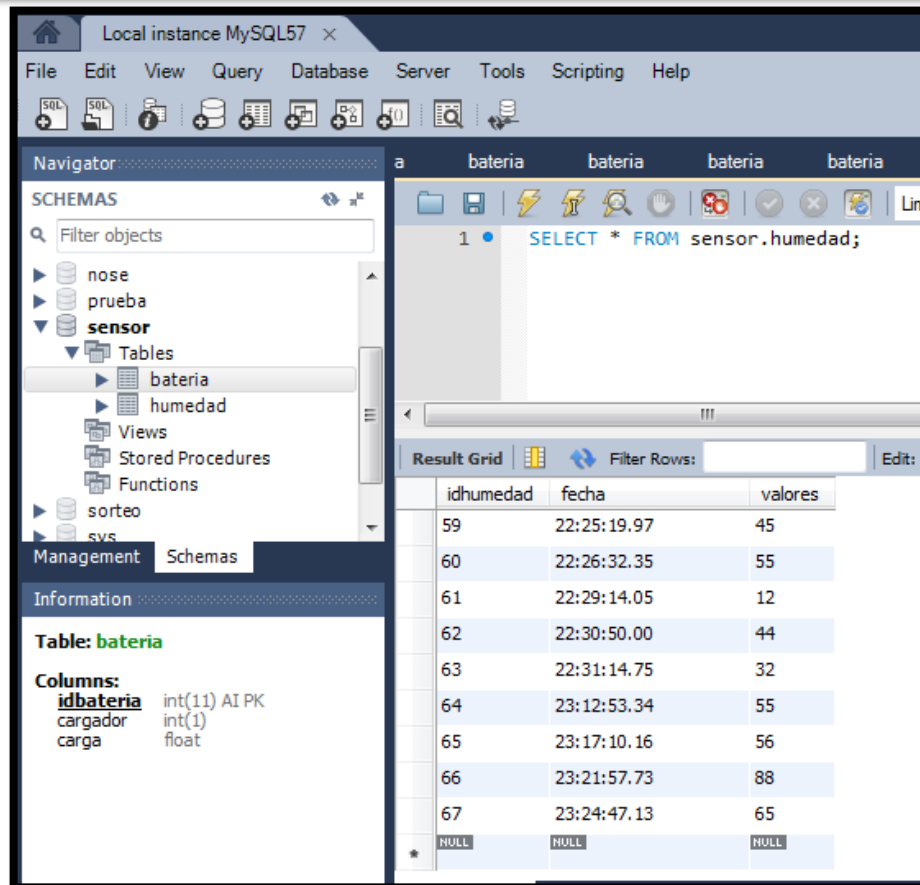


Imagen 5: Detalles del archivo sensor_humedad.sql.

BASE DE DATOS EN MYSQL

Resultados de la implementación de la base de datos en MySQL



Imagen 8: Tablas de batería y humedad

CÓDIGO IMPLEMENTADO EN VSCODE

ENTREGA 2:

The screenshot shows a VS Code editor with a Python script named 'main.py' and its terminal output. The script connects to a MySQL database and displays a menu for humidity and battery level recording.

```

1 """Programa para gestionar la BD e iniciar objetos de la clase codigo"""
2
3
4 from database import Database
5 from codigo import Sensor
6
7 def main():
8
9     def principal():
10         """metodo inicio programa
11         Args: eleccion(int): Menú
12         """
13         db=Database("localhost", 3306,"root","root","sensor")
14         sensor1=Sensor(db)
15         print("BIENVENIDO")
16         sensor1.opcionesMenuPrincipal()
17
18
19 if __name__ == "__main__":
20     main()
21
22

```

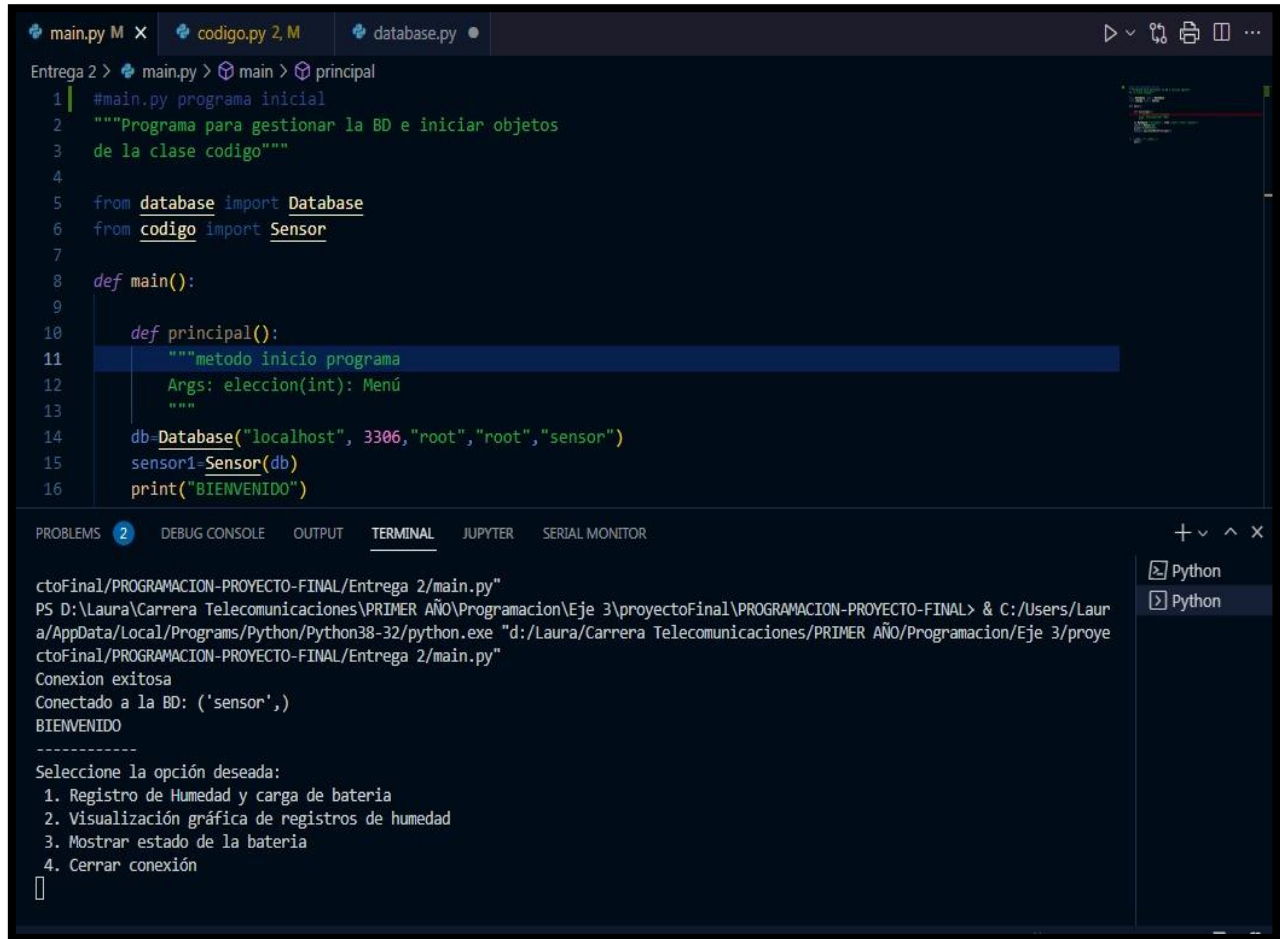
The terminal output shows the execution of the script:

```

nicolas@DESKTOP-HH88IVT MINGW64 /f:/nicolas/Documents/Pao/ISPC/SEGUNDO_CUATRIMESTRE/PROYECTO-PROGRAMACIÓN
$ f:/nicolas/Documents/Pao/ISPC/SEGUNDO_CUATRIMESTRE/PROYECTO-PROGRAMACIÓN/.venv/Scripts/python.exe "f:/nicolas/Documents/Pao/ISPC/SEGUNDO_CUATRIMESTRE/PROYECTO-PROGRAMACIÓN/PROYECTO-PROGRAMACIÓN-PROYECTO-FINAL/Entrega 2/main.py"
00427ac0Conexión exitosa
Conectado a la BD: ('sensor',)
BIENVENIDO
-----
Seleccione la opción deseada:
1. Registro de Humedad
2. Registro de nivel de batería
3. Visualización de registros de humedad
4. Cerrar conexión

```

ÚLTIMA MODIFICACIÓN



The image shows a Visual Studio Code editor window with three tabs: `main.py`, `codigo.py`, and `database.py`. The `main.py` tab is active, displaying the following Python code:

```
1 | #main.py programa inicial
2 | """Programa para gestionar la BD e iniciar objetos
3 | de la clase codigo"""
4 |
5 | from database import Database
6 | from codigo import Sensor
7 |
8 | def main():
9 |
10 |     def principal():
11 |         """metodo inicio programa
12 |         Args: eleccion(int): Menú
13 |         """
14 |         db=Database("localhost", 3306,"root","root","sensor")
15 |         sensor1=Sensor(db)
16 |         print("BIENVENIDO")
```

Below the editor, the `TERMINAL` panel is open, showing the execution output:

```
ctoFinal\PROGRAMACION-PROYECTO-FINAL\Entrega 2/main.py"
PS D:\Laura\Carrera Telecomunicaciones\PRIMER AÑO\Programacion\Eje 3\proyectoFinal\PROGRAMACION-PROYECTO-FINAL> & C:/Users/Laura/AppData/Local/Programs/Python/Python38-32/python.exe "d:/Laura/Carrera Telecomunicaciones/PRIMER AÑO/Programacion/Eje 3/proyectoFinal/PROGRAMACION-PROYECTO-FINAL/Entrega 2/main.py"
Conexion exitosa
Conectado a la BD: ('sensor',)
BIENVENIDO
-----
Seleccione la opción deseada:
1. Registro de Humedad y carga de batería
2. Visualización gráfica de registros de humedad
3. Mostrar estado de la batería
4. Cerrar conexión
█
```

Imagen 9: Código ejecutado en Vscode

GRÁFICOS DE RELACIÓN

La imagen que se presenta a continuación, muestra los valores obtenidos referidos a la humedad. Para ello se utilizó Matplotlib.

.

ENTREGA 2:

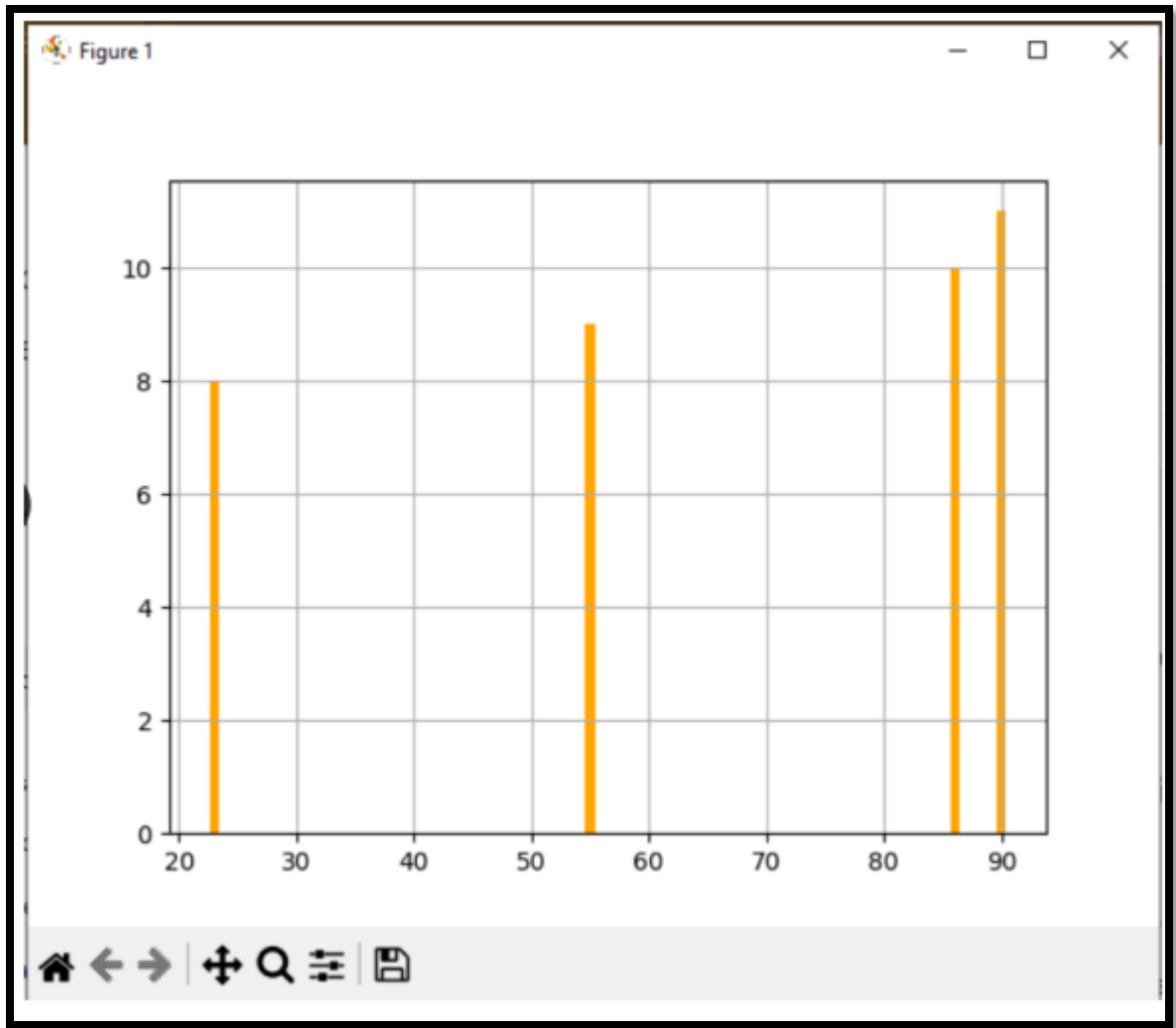


Imagen 10: valores de humedades representadas en Matplotlib

ENTREGA 3:

En esta entrega se realizaron los cambios pertinentes dentro del código para poder representar de una forma más clara la relación que existe entre la humedad y el tiempo.

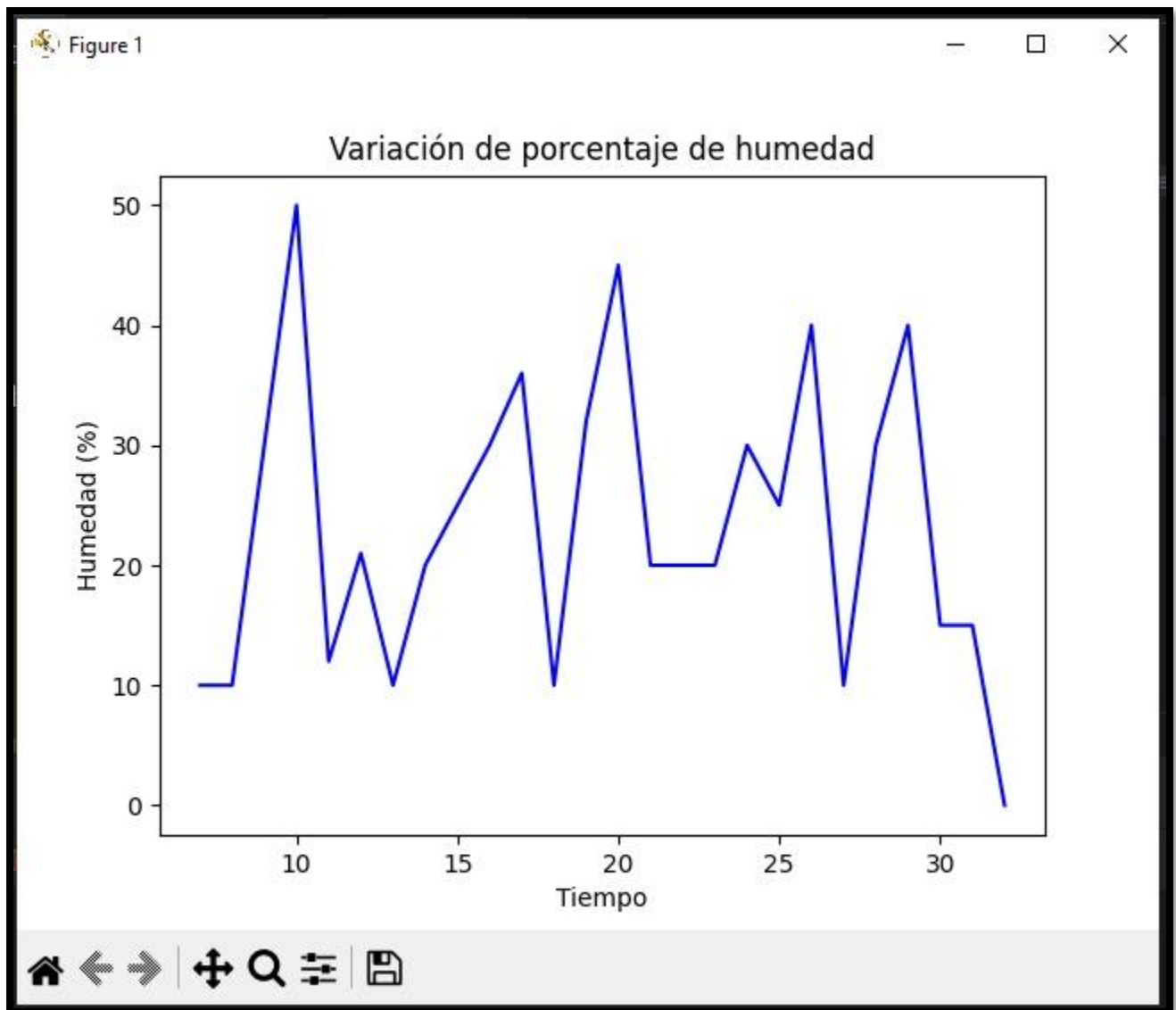


Imagen 11: valores de humedades representadas en Matplotlib

Conclusión

El desarrollo de este nodo IoT para "Dispositivos Inteligentes SRL" ha permitido implementar un sistema de monitoreo de humedad eficiente, ideal para espacios reducidos. Este dispositivo inteligente permite registrar y almacenar datos de porcentaje de humedad en tiempo real posibilitando un control preciso y adaptado a entornos específicos:

1. Simulación del Sensor en Proteus: La simulación del sensor de humedad en Proteus permitió verificar que los datos capturados son precisos y consistentes, garantizando que el sistema responde correctamente a variaciones de humedad. Esto establece una base sólida para futuros pasos de implementación en entornos reales, brindando seguridad sobre el funcionamiento y rendimiento del sensor bajo condiciones controladas.
2. Conexión y Almacenamiento en la Base de Datos: La transmisión continua y confiable de datos desde el sensor hacia la base de datos valida que la estructura de almacenamiento es eficiente. La base de datos organiza los registros de humedad en un formato que permite un acceso rápido y seguro, lo cual es esencial para mantener la integridad y disponibilidad de los datos.
3. Interfaz en Visual Studio Code: La interfaz en Visual Studio Code, diseñada como un menú de navegación, proporciona a los usuarios un acceso intuitivo a los datos, favoreciendo la toma de decisiones. Además, esta plataforma permite flexibilidad en futuras mejoras, como agregar nuevas funciones o modificar configuraciones, asegurando la escalabilidad del proyecto.
4. Conectividad Integral del Sistema: La integración exitosa entre el sensor, la base

de datos y la interfaz de usuario confirma la solidez de la arquitectura del sistema. Cada componente cumple su función dentro del flujo de datos: el sensor recoge información en tiempo real, la base de datos almacena y organiza, y la interfaz en Visual Studio Code permite visualizar y gestionar el sistema de forma eficiente. La conectividad estable entre estos componentes sugiere que el sistema es capaz de escalar a un entorno físico con mínimas modificaciones.

5. Mejoras implementadas: la utilización de leds de alertas automáticas ante valores críticos de humedad y porcentaje de batería, y la incorporación de gráficos visuales en la interfaz permite una interpretación de datos más eficiente y rápida.

En conclusión, esta solución no solo responde a las necesidades de monitoreo de la empresa, sino que también contribuye al avance en el uso de tecnologías IoT para la gestión y optimización de recursos en espacios controlados, integrando simulación, almacenamiento y visualización en una interfaz de desarrollo accesible. Se demuestra así su potencial para evolucionar hacia aplicaciones de monitoreo en tiempo real con mínimas intervenciones. La eficiencia de cada componente y su integración son factores claves que garantizan la escalabilidad y usabilidad del sistema en entornos reales.

Webgrafía

- Copyright © 2023 - Prometec . (s.f.). *Prometec* . Obtenido de Prometec :
<https://www.prometec.net/sensores-dht11/>
- geekfactory. (28 de Sep de 2017). *geekfactory*.
Obtenido de <https://www.geekfactory.mx/tutoriales-arduino/alimentar-el-arduino-la-guia-definitiva/>
- Hernández, L. d. (s.f.). *programarfacil*. Obtenido de
programarfacil:<https://programarfacil.com/domotica/bateria-para-esp32-esp8266/>
- juanjobe. (11 de May de 2021). *iasalud*.
Obtenido de [iasalud:https://iasalud.es/esp32-y-dth11-temperatura-y-humedad/](https://iasalud.es/esp32-y-dth11-temperatura-y-humedad/)
- *manlybatterycompany*.(s.f.).Obtenido de
<https://manlybattery.com/es/gu%C3%ADa-de-bater%C3%ADa-lifepo4-de-12v-todo-lo-que-necesitas/#:~:text=Esta%20disposici%C3%B3n%20es%20clave%20para,nomina%20de%203%2C2%20V.>
- SenodeX. (8 de may de 2022). *youtube*. Obtenido
de <https://www.youtube.com/watch?v=4oGh68PtBz0&t=3s>

ANEXOS

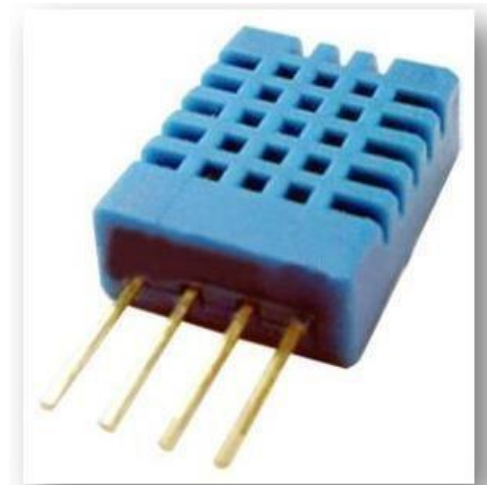
ANEXO ALFA:

Datasheet DHT 11 Humidity & Temperature Sensor

DHT 11 Humidity & Temperature Sensor

1. Introduction

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

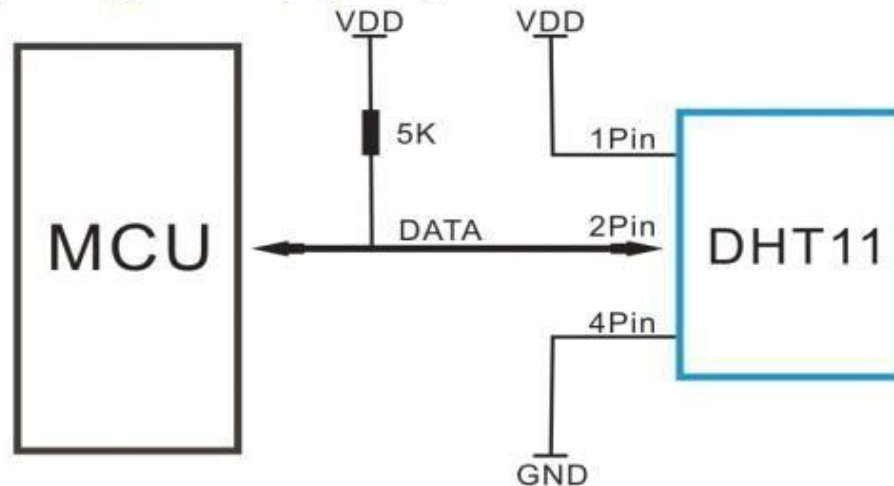
2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 °C	±5 %RH	±2°C	1	4 Pin Single Row

Detailed Specifications:

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			± 1%RH	
Accuracy	25 °C		± 4%RH	
	0-50 °C			± 5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0 °C	30%RH		90%RH
	25 °C	20%RH		90%RH
	50 °C	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25 °C , 1m/s Air	6 S	10 S	15 S
Hysteresis			± 1%RH	
Long-Term Stability	Typical		± 1%RH/year	
Temperature				
Resolution		1 °C	1 °C	1 °C
		8 Bit	8 Bit	8 Bit
Repeatability			± 1 °C	
Accuracy		± 1 °C		± 2 °C
Measurement Range		0 °C		50 °C
Response Time (Seconds)	1/e(63%)	6 S		30 S

3. Typical Application (Figure 1)**Figure 1 Typical Application**

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

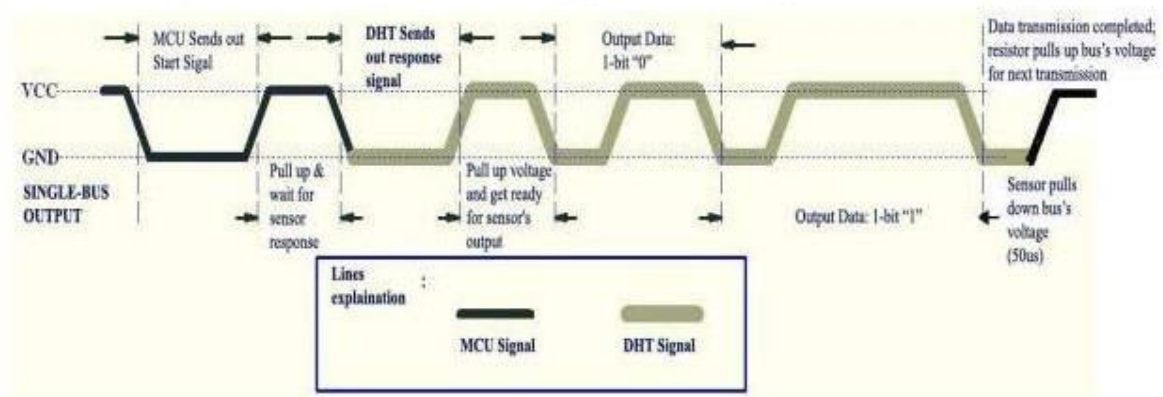


Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

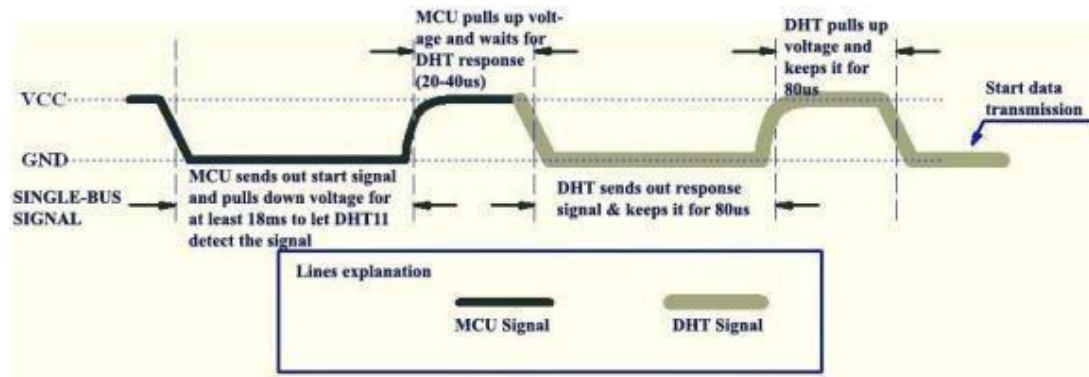


Figure 3 MCU Sends out Start Signal & DHT Responses

5.3 DHT Responses to MCU (Figure 3, above)

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).

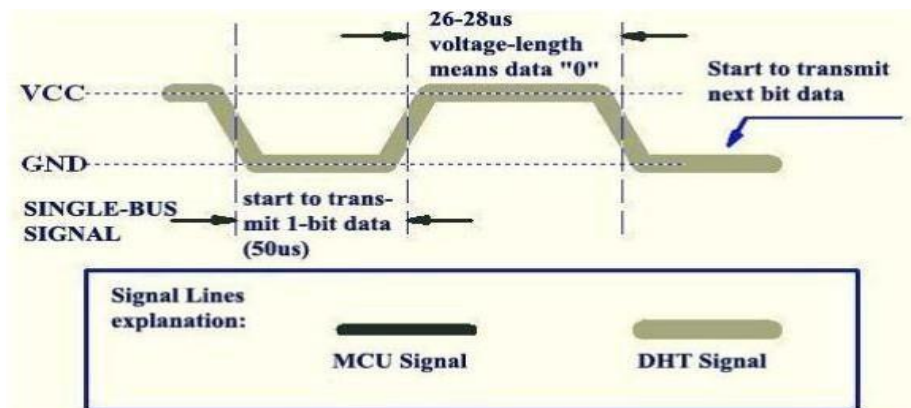


Figure 4 Data "0" Indication

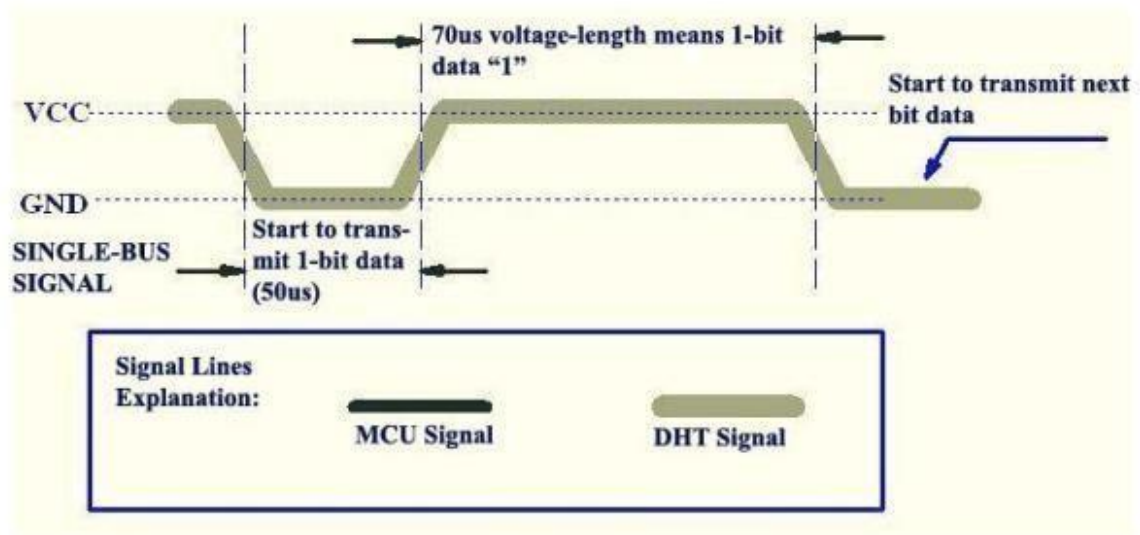


Figure 5 Data "1" Indication

If the response signal from DHT is always at high-voltage-level, it suggests that DHT is not responding properly and please check the connection. When the last bit data is transmitted, DHT11 pulls down the voltage level and keeps it for 50us. Then the Single-Bus voltage will be pulled up by the resistor to set it back to the free status.

6. Electrical Characteristics

VDD=5V, T = 25°C (unless otherwise stated)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

7. Attentions of application

(1) Operating conditions

Applying the DHT11 sensor beyond its working range stated in this datasheet can result in 3%RH signal shift/discrepancy. The DHT11 sensor can recover to the calibrated status gradually when it gets back to the normal operating condition and works within its range. Please refer to (3) of

this section to accelerate its recovery. Please be aware that operating the DHT11 sensor in the non-normal working conditions will accelerate sensor's aging process.

(2) Attention to chemical materials

Vapor from chemical materials may interfere with DHT's sensitive-elements and debase its sensitivity. A high degree of chemical contamination can permanently damage the sensor.

(3) Restoration process when (1) & (2) happen

Step one: Keep the DHT sensor at the condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two:K keep the DHT sensor at the condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Temperature Affect

Relative humidity largely depends on temperature. Although temperature compensation technology is used to ensure accurate measurement of RH, it is still strongly advised to keep the humidity and temperature sensors working under the same temperature. DHT11 should be mounted at the place as far as possible from parts that may generate heat.

(5) Light Affect

Long time exposure to strong sunlight and ultraviolet may debase DHT's performance.

(6) Connection wires

The quality of connection wires will affect the quality and distance of communication and high quality shielding-wire is recommended.

(7) Other attentions

- * Welding temperature should be bellow 260Celsius and contact should take less than 10 seconds.
- * Avoid using the sensor under dew condition.
- * Do not use this product in safety or emergency stop devices or any other occasion that failure of DHT11 may cause personal injury.
- * Storage: Keep the sensor at temperature 10-40°C, humidity <60%RH.

Disclaimer

This is a translated version of the manufacturer's data sheet. OSEPP is not responsible for the accuracy of the translated information.

ANEXO BRAVO:
CÓDIGO ARDUINO IDE-ARDUINO UNO

```
//CODIGO FINAL
#include <DHT.h>
#include <DHT_U.h>

#include "DHT.h"
// Pines para los LEDs
#define LEDVERDE 3
#define LEDAMARILLO 4
#define LEDROJO 5
// Variables
int analogValor = 0;
float voltaje = 0;
int ledDelay = 800;
// Umbrales
float maximo = 40;
float medio = 27;

#define DHTTYPE DHT11
const int DHTPIN = 2; // Pin donde está conectado el DHT11
DHT dht(DHTPIN, DHTTYPE);
#define LED_AZUL 6 // LED azul para indicar humedad
#define RELE 7 // Pin del relé que controla el LED amarillo
#define SENSOR_BATERIA A0
//Variable donde almacenaremos el valor del potenciómetro
long valor=A1;
int Potenciómetro = 0;
int position;

void setup() {
  Serial.begin(9600);
  Serial.println("Sensor de Temperatura y Humedad DHT11");
  pinMode(LED_AZUL, OUTPUT);
  pinMode(RELE, OUTPUT);
  dht.begin();
  // Los pines de LED en modo salida
  pinMode(LEDVERDE, OUTPUT);
  pinMode(LEDAMARILLO, OUTPUT);
  pinMode(LEDROJO, OUTPUT);
}

void loop() {
```



```

delay(2000);
// Leer humedad
float humedad = dht.readHumidity();
delay(10);

// Leer nivel de batería
int lectura_bateria = analogRead(SENSOR_BATERIA);
float voltaje_bateria = lectura_bateria * (9.0 / 1023.0); // Conversión a voltaje real

//leer el valor del potenciómetro
Potenciómetro = analogRead(A1);
position = map(Potenciómetro, 0, 1023, 0, 100); // convertir a porcentaje

// Dependiendo del valor del potenciómetro mostramos un LED u otro
if (position >= maximo)//40
{
digitalWrite(LEDVERDE, HIGH);
delay(250);
digitalWrite(LEDVERDE, LOW);
}
else if (position < maximo)//50 y 27
{
digitalWrite(LEDAMARILLO, HIGH);
delay(250);
digitalWrite(LEDAMARILLO, LOW);
}

// Apagamos todos los LEDs
digitalWrite(LEDVERDE, LOW);
digitalWrite(LEDAMARILLO, LOW);
digitalWrite(LEDROJO, LOW);
Serial.print(String(humedad));
Serial.print(",");
Serial.print(String(voltaje_bateria));
Serial.print(",");
Serial.println(String(position));

// Comportamiento del LED azul y el relé (LED amarillo) según la humedad
if (humedad > 85) {
// Si la humedad es mayor al 85%
digitalWrite(RELE, HIGH); // Encender el LED amarillo
delay(250); // LED amarillo parpadea cada 250 ms (más rápido)
digitalWrite(RELE, LOW);
delay(250); // Mantener apagado por 250 ms
} else if (humedad >80 && humedad <= 85) {
// Si la humedad está entre 80% y 85%
digitalWrite(LED_AZUL, HIGH); // Mantener el LED azul encendido

```

```
digitalWrite(RELE, HIGH); // Mantener el relé (LED amarillo) encendido
} else if (humedad > 55 && humedad <= 80) {
// Si la humedad está entre 55% y 80%
digitalWrite(LED_AZUL, HIGH); // Encender el LED azul
digitalWrite(RELE, HIGH); // Encender el relé (LED amarillo)
delay(5000); // Mantenerlos encendidos 5 segundos
digitalWrite(LED_AZUL, LOW); // Apagar el LED azul después de 5 segundos
} else if (humedad <=25) {
// Si la humedad es menor al 25%, el LED azul parpadea 3 veces
for (int i = 0; i < 3; i++) {
digitalWrite(LED_AZUL, HIGH);
delay(500); // Mantener encendido 500 ms
digitalWrite(LED_AZUL, LOW);
delay(500); // Apagar 500 ms
}
delay(4000); // Esperar 4 segundos antes de repetir
} else {
// Si la humedad está entre 25% y 55%, ambos LEDs se apagan
digitalWrite(LED_AZUL, LOW); // Apagar el LED azul
digitalWrite(RELE, LOW); // Apagar el relé (LED amarillo)
}
delay(2000); // Pequeña pausa entre lecturas
```