SRM Institute of Science and Technology
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204
Department of Computer Applications
Circular – 2023-24
BCA DS 2nd Sem

Internet of Things (UDS24M01J)

Lab Manual

Lab 1: Explain working of Raspberry Pi

Title: Introduction to Raspberry Pi

Aim: To understand the architecture, features, and basic operations of the Raspberry Pi.

Procedure:

1. Set up the Raspberry Pi with the necessary peripherals (monitor, keyboard, mouse, power supply).
2. Install the operating system (Raspbian) on the SD card.
3. Boot up the Raspberry Pi and explore the desktop environment.
4. Learn basic Linux commands using the terminal.
5. Understand the GPIO pins and their functions.

Source Code: (N/A - This lab is primarily explanatory)

Input: N/A

Expected Output: A working Raspberry Pi setup with the ability to navigate the OS and use basic Linux commands.

Lab 2: Controlling LED with Raspberry Pi

Title: LED Control with Raspberry Pi

Aim: To control an LED using the Raspberry Pi's GPIO pins.

Procedure:

1. Connect an LED to a GPIO pin on the Raspberry Pi using a suitable resistor.
2. Write a Python script to turn the LED on and off.
3. Run the script and observe the LED behavior.

Source Code:

```
import RPi.GPIO as GPIO
import time

# Set the GPIO pin number
led_pin = 18

# Set the GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

try:
    while True:
        GPIO.output(led_pin, GPIO.HIGH)  # Turn LED on
        time.sleep(1)
        GPIO.output(led_pin, GPIO.LOW)   # Turn LED off
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()  # Clean up GPIO settings on program exit
```

Input: N/A

Expected Output: The LED should blink on and off with a 1-second delay.

Lab 3: Interfacing Light Sensor with Raspberry Pi

Title: Light Sensor Interface with Raspberry Pi

Aim: To interface a light sensor (e.g., LDR) with the Raspberry Pi and read light intensity values.

Procedure:

1. Connect the light sensor to the Raspberry Pi using an appropriate circuit (e.g., voltage divider).
2. Write a Python script to read the analog voltage from the sensor using an ADC (if necessary) or a digital light sensor.
3. Convert the voltage reading to a light intensity value.
4. Print the light intensity value.

Source Code:

```
import RPi.GPIO as GPIO
import time
import spidev # Only needed if using an ADC

# Configuration for LDR
LDR_PIN = 24
#spi = spidev.SpiDev() #Only needed if using an ADC
#spi.open(0, 0)      # Only needed if using an ADC

def read_ldr():
    #Simplified LDR reading (Digital)
    GPIO.setup(LDR_PIN, GPIO.IN)
    if GPIO.input(LDR_PIN) == GPIO.LOW:
        return 0  # Dark
    else:
        return 1 # Light

    # # ADC Reading (If you have an analog LDR and ADC)
    # adc_channel = 0
    # data = spi.xfer2([1, (8 + adc_channel) << 4, 0])
    # light_level = ((data[1] & 3) << 8) + data[2]
    # return light_level

# Setup GPIO
GPIO.setmode(GPIO.BCM)
#GPIO.setup(LDR_PIN, GPIO.IN) # Removed, done in function
try:
    while True:
        light_value = read_ldr()
        print("Light Intensity:", light_value)
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()
    #spi.close() #only needed if using ADC
```

Input: Varying light conditions.

Expected Output: The program should print the light intensity value, which changes as the light conditions vary.

Lab 4: Demonstrate a smart object API gateway service reference implementation in IoT toolkit

Title: Smart Object API Gateway

Aim: To demonstrate a smart object API gateway service.

Procedure:

1. Set up an IoT toolkit (e.g., Node-RED, ThingsBoard) on a suitable platform.
2. Configure a simulated or real smart object (e.g., a sensor) to send data.
3. Use the IoT toolkit to create an API gateway that receives data from the smart object.
4. Implement API endpoints to access the data from the smart object.
5. Test the API endpoints to ensure they are working correctly.

Source Code: (This lab involves configuration rather than direct code. The configuration will depend on the IoT toolkit used. Example Node-RED flow)

Input: Data from a simulated or real smart object.

Expected Output: A functional API gateway that allows access to smart object data.

Lab 5: Write and explain working of an HTTP-to-CoAP semantic mapping proxy in IoT toolkit

Title: HTTP-to-CoAP Proxy

Aim: To implement and explain the working of an HTTP-to-CoAP semantic mapping proxy.

Procedure:

1. Set up an IoT toolkit that supports CoAP (e.g., Node-RED with CoAP nodes).
2. Configure a CoAP server (either simulated or real).
3. Set up a proxy within the IoT toolkit to translate HTTP requests to CoAP requests.
4. Implement semantic mapping to translate data formats and structures between HTTP and CoAP.
5. Test the proxy by sending HTTP requests and verifying that they are correctly translated to CoAP requests and that the CoAP responses are correctly translated back to HTTP.

Source Code: (This lab involves configuration. Example Node-RED flow)

Input: HTTP requests.

Expected Output: Correctly translated CoAP requests and HTTP responses.

Lab 6: Describe gateway as a service deployment in IoT toolkit

Title: Gateway as a Service

Aim: To describe the deployment of a gateway as a service in an IoT toolkit.

Procedure:

1. Choose an IoT toolkit that supports gateway-as-a-service concepts.
2. Configure a gateway within the toolkit. This might involve setting up communication protocols, data processing rules, and security settings.
3. Deploy the gateway as a service. This could involve running it as a separate process, container, or cloud service.
4. Connect simulated or real IoT devices to the gateway.
5. Monitor the gateway's operation, including data flow, resource usage, and any errors.

Source Code: (Configuration-based)

Input: Data from IoT devices.

Expected Output: A functional IoT gateway deployed as a service, correctly routing and processing data.

Lab 7: Explain application framework and embedded software agents for IoT toolkit

Title: IoT Application Framework and Embedded Agents

Aim: To explain the application framework and the role of embedded software agents in an IoT toolkit.

Procedure:

1. Select an IoT toolkit that provides an application framework and supports embedded software agents.
2. Explore the application framework. Understand its components, such as data management, communication protocols, and application logic execution.
3. Learn about embedded software agents. Understand their role in data collection, processing, and communication on IoT devices.
4. Develop a simple IoT application using the framework and deploy an embedded agent to a simulated device.
5. Observe how the application framework and the agent interact.

Source Code: (Example agent code in Python for a MicroPython-based device)

```
# Example MicroPython code for an embedded agent
import time
from umqtt.simple import MQTTClient #library needs to be installed on device

# MQTT Broker details
MQTT_SERVER = "your_mqtt_broker_address"
MQTT_CLIENT_ID = "my_agent_id"
MQTT_TOPIC = "sensor/data"

# Sensor reading function (simulated)
def get_sensor_data():
    return {"temperature": 25 + (random.randint(-5, 5)), "humidity": 60 +
(random.randint(-10, 10))}

# Function to connect to MQTT broker
def connect_mqtt():
    client = MQTTClient(MQTT_CLIENT_ID, MQTT_SERVER)
    client.connect()
    print("Connected to MQTT Broker")
    return client

# Function to publish sensor data
def publish_data(client, data):
    try:
        client.publish(MQTT_TOPIC, ujson.dumps(data))
        print("Published:", data)
    except Exception as e:
        print("Error publishing data:", e)

# Main function
def main():
    try:
        client = connect_mqtt()
        while True:
            sensor_data = get_sensor_data()
            publish_data(client, sensor_data)
            time.sleep(5)  # Send data every 5 seconds
    except KeyboardInterrupt:
        print("Stopping agent")
```

```
    finally:
        client.disconnect()

if __name__ == "__main__":
    main()
```

Input: Sensor data.

Expected Output: The application should function correctly, with the embedded agent collecting and transmitting data.

Lab 8: Arduino with ESP8266 explanation

Title: Arduino with ESP8266

Aim: To explain how to use an ESP8266 with an Arduino for IoT projects.

Procedure:

1. Introduce the ESP8266 module and its capabilities (Wi-Fi connectivity).
2. Explain different ways to connect the ESP8266 with an Arduino (e.g., serial communication).
3. Show how to program the ESP8266 using the Arduino IDE (with the ESP8266 add-on).
4. Demonstrate a simple project, such as sending sensor data from an Arduino to a web server using the ESP8266.

Source Code:

```
// Arduino code to send data via ESP8266
#include <SoftwareSerial.h>

SoftwareSerial espSerial(2, 3); // RX, TX for Arduino
#define DEBUG true

String apiKey = "YOUR_API_KEY";  // Replace with your ThingSpeak API key
String ssid = "YOUR_WIFI_SSID";      // Replace with your Wi-Fi SSID
String pass = "YOUR_WIFI_PASSWORD";  // Replace with your Wi-Fi password
String server = "api.thingspeak.com";
int port = 80;

void setup() {
  Serial.begin(9600);
  espSerial.begin(9600);
  delay(1000);
  sendData("AT+RST", 2000, DEBUG);
  sendData("AT+CWMODE=1", 1000, DEBUG);
  connectWiFi();
}

void loop() {
  int sensorValue = analogRead(A0);  // Read from an analog sensor
  String data = String("field1=") + String(sensorValue);
  sendData("AT+CIPSTART=\"TCP\",\"" + server + "\"," + port, 10000, DEBUG);
  sendData("AT+CIPSEND=" + String(data.length() + 54), 1000, DEBUG); //
Adjusted length
  String postCommand = "POST /update?api_key=" + apiKey + "&" + data + "
HTTP/1.1\r\nHost: " + server + "\r\nConnection: close\r\n\r\n";
  sendData(postCommand, 5000, DEBUG);
  sendData("AT+CIPCLOSE", 5000, DEBUG);
  Serial.println("Data sent");
  delay(30000); // Send data every 30 seconds
}

void connectWiFi() {
  sendData("AT+CWJAP=\"" + ssid + "\",\"" + pass + "\"", 10000, DEBUG);
}

String sendData(String command, const int timeout, boolean debug) {
  String response = "";
  espSerial.println(command);
```

```
  long int time = millis();
  while ((time + timeout) > millis()) {
    while (espSerial.available()) {
      char c = espSerial.read();
      response += c;
    }
  }
  if (debug) {
    Serial.print(command);
    Serial.print(" : ");
    Serial.println(response);
  }
  return response;
}
```

Input: Sensor data from Arduino.

Expected Output: Data sent from Arduino to a web server (e.g., ThingSpeak) via ESP8266.

Lab 9: Weather Monitoring System

Title: Weather Monitoring System

Aim: To build a system that monitors weather parameters like temperature, humidity, and pressure.

Procedure:

1. Connect temperature and humidity sensor (e.g., DHT11, DHT22) and a pressure sensor (e.g., BMP180, BMP280) to a microcontroller (e.g., Arduino, Raspberry Pi).
2. Write code to read data from the sensors.
3. (Optional) Store the data locally or send it to a cloud platform (e.g., ThingSpeak, Adafruit IO).
4. Display the weather data on a local display (e.g., LCD) or a web interface.

Source Code: (Arduino)

```
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_BMP280.h> //Library needs to be installed

#define DHTPIN 7
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP280 bmp;

float temp;
float hum;
float pressure;

void setup() {
  Serial.begin(9600);
  dht.begin();
  if (!bmp.begin(0x76)) {
    Serial.println("Could not find a valid BMP280 sensor!");
    while (1);
  }
}

void loop() {
  delay(2000);
  temp = dht.readTemperature();
  hum = dht.readHumidity();
  pressure = bmp.readPressure();

  if (isnan(temp) || isnan(hum)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.print(" *C\t");
  Serial.print("Humidity: ");
  Serial.print(hum);
  Serial.print(" %\t");
  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" Pa");
```

```
}
```

Input: Environmental conditions.

Expected Output: Display of temperature, humidity, and pressure readings.

Lab 10: Reading Data from Internet using sensor

Title: Reading Data from Internet using sensor

Aim: To read data from a sensor and send it to a web server, making it accessible over the internet.

Procedure:

1. Connect a sensor (e.g., temperature, humidity) to a microcontroller with internet connectivity (e.g., ESP8266, ESP32, Raspberry Pi).
2. Write code to read data from the sensor.
3. Establish a connection to a Wi-Fi network.
4. Send the sensor data to a web server (e.g., using HTTP POST or MQTT).
5. (Optional) Display the data on a webpage or use a web API to retrieve the data.

Source Code: (ESP32 with DHT11)

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* serverName = "your_server_address";  // e.g.,
"api.thingspeak.com"
String apiKey = "YOUR_API_KEY";           // e.g., your ThingSpeak API key

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  delay(2000);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  WiFiClient client;
  HTTPClient http;

  String postData = "api_key=" + apiKey + "&field1=" + String(temperature) +
"&field2=" + String(humidity);
  http.begin(client, serverName);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
```

```
  int httpResponseCode = http.POST(postData);

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  http.end();
  Serial.println("Data sent");
}
```

Input: Sensor data.

Expected Output: Sensor data sent to a web server and accessible over the internet.

Lab 11: Home Automation

Title: Home Automation

Aim: To design and implement a basic home automation system.

Procedure:

1. Connect devices like relays (to control lights, fans), sensors (temperature, motion), and a microcontroller (e.g., Raspberry Pi, ESP32)
2. Write code to control the devices based on sensor readings or user input.
3. Create a user interface (web or mobile app) to control and monitor the system.
4. Implement features like remote control, scheduling, and automation rules.

Source Code: (Raspberry Pi with Flask (web server) and GPIO)

app.py (Flask web server)

```
from flask import Flask, render_template, request, jsonify
import RPi.GPIO as GPIO
import time

app = Flask(__name__)

# GPIO setup
GPIO.setmode(GPIO.BCM)
LIGHT_PIN = 17  # Example: GPIO 17 controls a light
GPIO.setup(LIGHT_PIN, GPIO.OUT)
GPIO.output(LIGHT_PIN, GPIO.LOW)  # Initialize light as off

# Function to get temperature (simulated for simplicity)
def get_temperature():
    # In a real application, read from a sensor like DHT22
    return 25 + (time.time() % 10)  # Simulate temperature variation

# Routes
@app.route("/")
def home():
    return render_template("index.html")

@app.route("/toggle_light", methods=["POST"])
def toggle_light():
    try:
        GPIO.output(LIGHT_PIN, not GPIO.input(LIGHT_PIN))
        light_state = "on" if GPIO.input(LIGHT_PIN) else "off"
        return jsonify(status="success", light_state=light_state)
    except Exception as e:
        return jsonify(status="error", error=str(e))

@app.route("/get_temp")
def get_temp():
    temperature = get_temperature()
    return jsonify(temperature=temperature)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80, debug=True)
```

index.html (HTML template)

```
<!DOCTYPE html>
<html>
<head>
    <title>Home Automation</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <style>
      body { font-family: sans-serif; }
      .container { width: 400px; margin: auto; padding: 20px; border:
1px solid #ccc; }
      button { padding: 10px; margin-top: 10px; cursor: pointer; }
      #temperature { font-size: 20px; font-weight: bold; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Home Automation System</h1>
        <button id="light_button">Toggle Light</button>
        <p>Light State: <span id="light_state">off</span></p>
        <p>Temperature: <span id="temperature"></span> °C</p>
    </div>
    <script>
    $(document).ready(function() {
        // Function to update light state
        function updateLightState(state) {
            $("#light_state").text(state);
            $("#light_button").text("Toggle Light (" + state + ")");
        }

        // Function to get and display temperature
        function getTemperature() {
            $.get("/get_temp", function(data) {
                $("#temperature").text(data.temperature);
            });
        }
        getTemperature(); // Initial update
        setInterval(getTemperature, 5000); // Update every 5 seconds

        // Light button click handler
        $("#light_button").click(function() {
            $.post("/toggle_light", function(data) {
                if (data.status === "success") {
                    updateLightState(data.light_state);
                } else {
                    alert("Error: " + data.error);
                }
            });
        });
    });
    </script>
</body>
</html>
```

Input: User actions (button clicks) and sensor data.

Expected Output: A web interface to control and monitor home devices.

Lab 12: Remote Surveillance System

Title: Remote Surveillance System

Aim: To set up a system for remote surveillance using a camera and internet connectivity.

Procedure:

Connect a camera (e.g., USB webcam, Raspberry Pi camera module) to a device with internet connectivity (e.g., Raspberry Pi).

1. Install and configure software to capture video from the camera (e.g., Motion, OpenCV).
2. Set up a way to access the video stream remotely (e.g., using a web server, streaming service).
3. Implement features like motion detection, recording, and alerts.

Source Code: (Raspberry Pi with Motion)

motion.conf (Configuration - Important settings)

```
daemon on
videodevice /dev/video0 # Or the correct video device
framerate 30
width 640
height 480
output_pictures off # Save individual images
stream_maxrate 5 # Frames per second for streaming
stream_localhost off # Allow remote access
```

Input: Camera feed.

Expected Output: A remotely accessible video stream with optional features like motion detection.

Lab 13: Smart Irrigation System

Title: Smart Irrigation System

Aim: To develop an automated irrigation system that optimizes water usage.

Procedure:

1. Connect a soil moisture sensor to a microcontroller (e.g., Arduino, Raspberry Pi).
2. Connect a water pump or valve to the microcontroller using a relay.
3. Write code to read soil moisture levels and control the pump/valve accordingly.
4. (Optional) Integrate a web interface or schedule to control and monitor the system remotely.

Source Code: (Arduino)

```
#include <SoilMoistureSensor.h> // Assuming you have this library
#include <Relay.h> //and this one

// Define sensor and relay pins
#define MOISTURE_PIN A0
#define RELAY_PIN 8

SoilMoistureSensor moistureSensor(MOISTURE_PIN);
Relay waterRelay(RELAY_PIN, LOW); // Relay is active low

// Define moisture thresholds
#define DRY_LEVEL 600  // Example: Adjust based on your sensor
#define WET_LEVEL 300  // Example: Adjust based on your sensor

void setup() {
  Serial.begin(9600);
  waterRelay.init();
  moistureSensor.begin();
}

void loop() {
  int moistureValue = moistureSensor.read();
  Serial.print("Moisture Value: ");
  Serial.println(moistureValue);

  if (moistureValue > DRY_LEVEL) {
    Serial.println("Soil is dry, turning on water pump");
    waterRelay.on();
    delay(5000); // Water for 5 seconds (adjust as needed)
    waterRelay.off();
    Serial.println("Water pump turned off");
  } else if (moistureValue < WET_LEVEL) {
    Serial.println("Soil is wet enough");
  } else {
    Serial.println("Moisture level is optimal");
  }
  delay(60000); // Check every minute
}
```

Input: Soil moisture levels.

Expected Output: Automated control of a water pump or valve based on soil moisture.

Lab 14: Health care system

Title: Health Care System

Aim: To develop a basic health care monitoring system.

Procedure:

1. Connect sensors like heart rate sensor, body temperature sensor to a microcontroller (e.g., Arduino, ESP32).
2. Write code to read data from the sensors.
3. (Optional) Transmit the data wirelessly (e.g., using Bluetooth, Wi-Fi) to a central system.
4. Display the data on a local display or a remote interface.
5. Implement features like alerts for abnormal readings.

Source Code: (ESP32 with Heart Rate Sensor and Temperature)

```
#include <Wire.h>
//#include "MAX30105.h" //  Example: for a pulse oximeter.  Library needs to
be installed
//#include "spo2_algorithm.h" //and this one
#include <WiFi.h>
#include <HTTPClient.h>

//MAX30105 particleSensor; // Pulse Oximeter
int heartRate;
float bodyTemperature;
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* serverName = "your_server_address";  // e.g.,
"yourdomain.com/api/data"

// Example temperature sensor pin
#define TEMP_PIN 15

void setup() {
  Serial.begin(115200);
  /*
  // Initialize pulse oximeter (MAX30105)
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30105 was not found. Please check wiring.");
    while (1);
  }
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0A); // Turn Red LED to low to save
power
  particleSensor.setPulseAmplitudeIR(0x0A);  // Turn IR LED to low to save
power
  */
  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

float readBodyTemperature() {
```

```
  // Replace this with actual temperature sensor reading
  // Example:  Use a thermistor or TMP36 sensor
  int sensorValue = analogRead(TEMP_PIN);
  float voltage = sensorValue * (5.0 / 1023.0);
  float temperatureC = (voltage - 0.5) * 100;  // Example for TMP36
  return temperatureC;
}

void loop() {
  /*
  // Read heart rate and SpO2
  long irValue = particleSensor.getRed();
  if (irValue > 50) {
    if (IR_SAMPLE_COUNT < 25) {
      IR_SAMPLES[IR_SAMPLE_COUNT] = irValue;
      IR_SAMPLE_COUNT++;
    }
    if (IR_SAMPLE_COUNT == 25)
    {
      heartRate = calculateHeartRate(IR_SAMPLES, IR_SAMPLE_COUNT);
      IR_SAMPLE_COUNT = 0;
    }
  }
  */
  heartRate = 80 + (random(20)); //Simulate
  bodyTemperature = readBodyTemperature();

  Serial.print("Heart Rate: ");
  Serial.print(heartRate);
  Serial.println(" bpm");
  Serial.print("Body Temperature: ");
  Serial.print(bodyTemperature);
  Serial.println(" °C");

  // Send data to server
  WiFiClient client;
  HTTPClient http;
  String postData = "heartRate=" + String(heartRate) + "&bodyTemperature=" +
String(bodyTemperature);
  http.begin(client, serverName);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  int httpResponseCode = http.POST(postData);
  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  http.end();
  delay(5000); // Send data every 5 seconds
}
```

Input: Sensor data (heart rate, temperature).

Expected Output: Display or transmission of health data, potentially with alerts.

Lab 15: Air Pollution Monitoring System

Title: Air Pollution Monitoring System

Aim: To build a system that monitors air quality by measuring pollutants.

Procedure:

1. Connect air quality sensors (e.g., MQ-135, MQ-7) to a microcontroller (e.g., Arduino, ESP32).
2. Write code to read data from the sensors.
3. (Optional) Transmit the data wirelessly to a server or cloud platform.
4. Display the pollutant levels and provide an air quality index.

Source Code: (ESP32 with MQ-135)

```
#include <WiFi.h>
#include <HTTPClient.h>
//#include "MQ135.h" //needs to be installed
// Define sensor pin
#define MQ135_PIN 34

//MQ135 mq135_sensor(MQ135_PIN);

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* serverName = "your_server_address";  // e.g.,
"yourdomain.com/api/airquality"

void setup() {
  Serial.begin(115200);
  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  // put your setup code here, to run once:
  //mq135_sensor.calibrate();
}

void loop() {
  // Read sensor values
  //float co2 = mq135_sensor.readCO2();
  float co2 = 400 + random(100); //Simulate
  Serial.print("CO2: ");
  Serial.print(co2);
  Serial.println(" ppm");

  // Send data to server
  WiFiClient client;
  HTTPClient http;
  String postData = "co2=" + String(co2);
  http.begin(client, serverName);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  int httpResponseCode = http.POST(postData);

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
```

```
    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  http.end();
  delay(5000); // Send data every 5 seconds
}
```

Input: Air pollutant levels.

Expected Output: Display or transmission of air quality data.