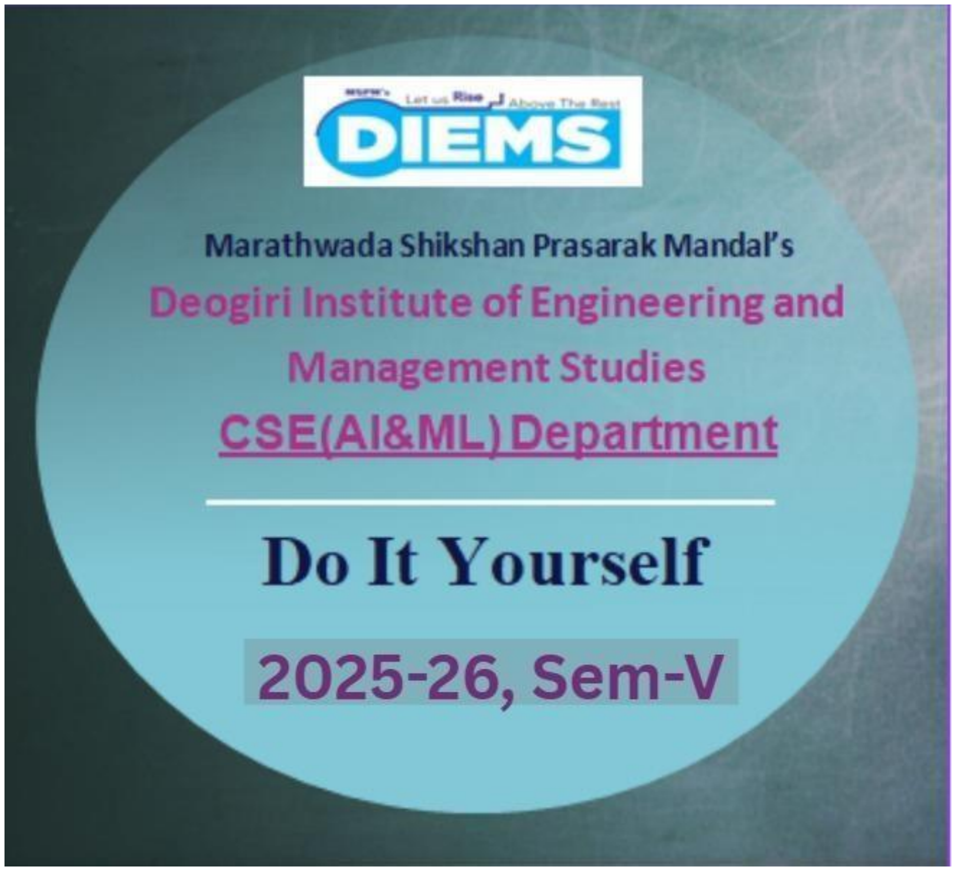

MSPM's Let us Rise Above The Rest

DIEMS

Marathwada Shikshan Prasarak Mandal's

Deogiri Institute of Engineering and Management Studies

CSE(AI&ML) Department

Do It Yourself

2025-26, Sem-V

A Report on DIY Activities

Submitted by

1. HARKAL VISHAL SAMPAT(AI3013)
2. KADAM SURAJ SANJAY(AI3007)
3. JADHAV VAISHNAVI KARBHARI(AI3005)
4. KAMBLE ROHAN KAILAS(AI3015)

Index

Sr.No.	**Name of Activity**	**Page No**
1	IOT Ecosystem exploration	1
2	Arduino Basics	3
3	Sensor Based Automation Using Arduino	6
4	Live Temperature And Humidity Monitering With Arduino	9
5	Exploration Of ESP32 Microcontroller	12
6	Connecting ESP32 To Wifi Network	15
7	IOT Cloud Data Storage Using Blynk App	18
8	Controlling Inbuilt LED Using Blynk App And ESP32	22
9	Displaying City Weather Using Open Weather API	26
10	Mini project	30

Practical 1: IoT Ecosystem Exploration

Objective: To study a specific IoT application and visually represent its ecosystem.

Introduction

The Internet of Things (IoT) refers to a network of physical devices—such as sensors, microcontrollers, actuators, appliances, and machines—that communicate and exchange data through the internet. IoT enables automation, remote monitoring, data analytics, and smart decision-making in real time.

An IoT system generally consists of four major layers:

1. Sensing Layer (Perception Layer)

This layer includes sensors and actuators used to collect real-world data such as:

- Temperature
- Humidity
- Motion
- Light intensity
- Soil moisture
- Gas/smoke levels

Common sensors: DHT11/22, PIR, LDR, MQ-2, ultrasonic sensor.

2. Network Layer

This layer transfers sensor data to a cloud/server

- Wi-Fi (ESP32, ESP8266)
- Bluetooth (ESP32, HC-05)
- Ethernet
- Mobile hotspots   Protocols used:
- HTTP/HTTPS

3. Processing Layer (Edge Computing / Cloud Layer) This layer processes the data and makes decisions.

- Microcontrollers (Arduino UNO, ESP32, Raspberry Pi Pico)
- Microcomputers (Raspberry Pi 4B)
- Cloud platforms (Blynk, Thingspeak, Firebase, AWS IoT, Azure IoT)

Functions:

- Data processing

- Machine learning inference
- Storage
- Analytics
- Dashboard visualization

4. Application Layer

This is what the end-user sees, such as:

- Mobile apps
- Web dashboards
- Smart home systems Health monitoring systems Smart agriculture apps.

Components Required (General IoT Setup**)**

Depending on your chosen application, typical components include:

Hardware

- Arduino UNO / ESP32 / Raspberry Pi Pico / Raspberry Pi 4 **Software**
- Arduino IDE / Thonny / Python
- Blynk/Thingspeak/Firebase dashboard

Conclusion

In this practical, we studied how IoT systems operate by exploring a complete IoT ecosystem. We understood how sensors collect data, how microcontrollers like Arduino, ESP32, Raspberry Pi Pico and processors like Raspberry Pi 4B handle communication, and how cloud platforms store and visualize data. This practical helps in understanding the foundation of IoT applications used in real-life fields such as smart cities, automation, agriculture, and healthcare.

Practical 2: Arduino Basics

Objective: To blink an LED, read sensor data, and control an actuator using Arduino.

Introduction

Arduino is an open-source microcontroller platform widely used in IoT, automation, robotics, and embedded systems. It allows users to interface sensors, LEDs, motors, buzzers, and other electronic components easily using simple C/C++ programming.

In this practical, three basic tasks are performed:

1. Blinking an LED (Digital Output)

This demonstrates how a microcontroller controls digital pins to turn devices ON or OFF.
It teaches:Pin mode configuration,Writing HIGH/LOW signals,Basic timing using delay()

2. Reading Sensor Data (Analog Input)

Arduino can read varying voltages (0–5V) using its ADC (Analog-to-Digital Converter).
Example sensors:LDR (light sensor),Potentiometer,Temperature sensor (LM35)

This helps understand:How sensors give analog signals ,How Arduino converts them to values (0–1023) ,How serial monitoring is used

3. Controlling an Actuator (Digital Output)

Actuators convert electrical signals into physical action.
Common actuators:Buzzer ,Relay ,Motor (through driver)

This part teaches: How external devices are controlled ,Use of conditional statements ,Decision-making based on sensor readings

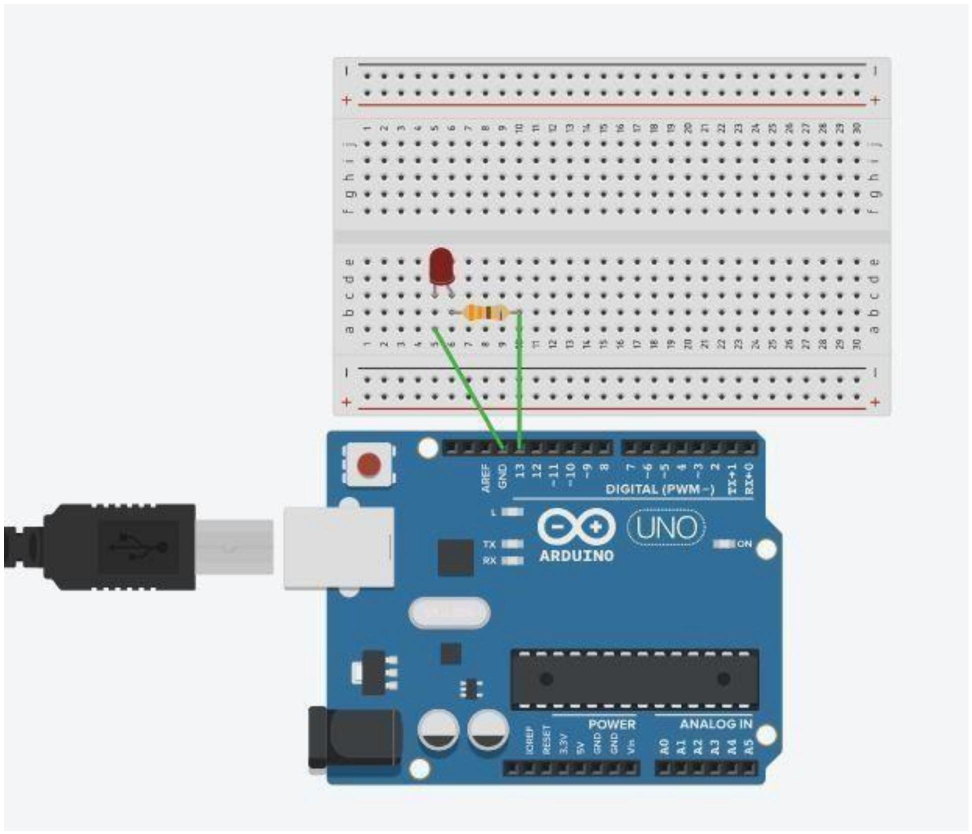
This practical provides the foundation for creating real IoT systems like smart lighting, security systems, alarms, and automation projects.

Components Required:

Hardware: Arduino UNO ,LED (1 pc), 220Ω Resistor,Breadboard,Jumper wires,USB cable

Software: Arduino IDE

Schematic Diagram (Description):



Code:

Below is a combined code that performs all 3 tasks:

```
// Practical 2: Arduino Basics int ledPin = 13;

// LED pin int sensorPin = A0;   // Sensor

connected to A0 int buzzerPin = 8;     // Buzzer

pin int sensorValue = 0; void setup() {

pinMode(ledPin, OUTPUT);

pinMode(buzzerPin, OUTPUT);
```

```
  Serial.begin(9600);   // Start serial monitor
} void loop() { // 1. Blink an
LED   digitalWrite(ledPin,
HIGH);   delay(500);
digitalWrite(ledPin, LOW);
delay(500);
 // 2. Read Sensor Data   sensorValue =
analogRead(sensorPin);
Serial.print("Sensor Value: ");
Serial.println(sensorValue);
// 3. Control Actuator Based on Sensor Value   if
(sensorValue > 500) {      digitalWrite(buzzerPin,
HIGH);  // Turn buzzer ON
  } else {
    digitalWrite(buzzerPin, LOW);   // Turn buzzer OFF
  }
delay(200);}
```

Conclusion:

In this practical, we successfully learned the basic operations of Arduino such as blinking an LED using digital output, reading real-time data from an analog sensor, and controlling an actuator (buzzer) based on sensor values. These foundational skills are essential for building smart IoT systems, automation projects, and advanced embedded applications. This practical helps students understand how hardware and software work together in an IoT ecosystem.

Practical No. 3: Sensor-Based Automation Using Arduino

Objective:

To interface a DHT11 sensor, LED, and DC motor with Arduino to automatically control the devices based on temperature and humidity levels**.**

Introduction (Theory Section)

The DHT11 sensor is a widely used digital sensor capable of measuring temperature and humidity. Using Arduino, we can monitor environmental conditions in real-time and make automatic decisions.

In many real-world applications—such as smart homes, greenhouses, server rooms, and agriculture—devices must respond automatically to environmental changes.
Examples:

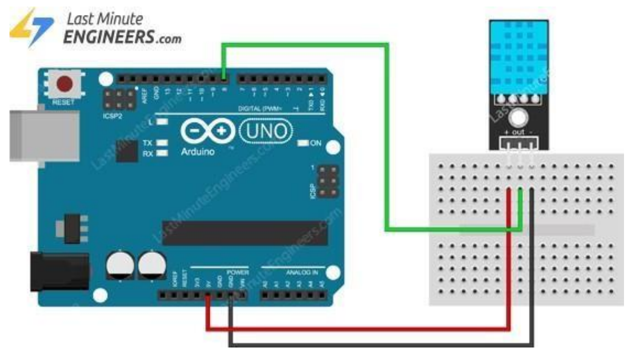
Turning ON a fan when temperature is high

Components Required:

Hardware:Arduino UNO,DHT11 Temperature & Humidity Sensor,LED (1 pc),220Ω resistor, DC Motor,MotorDriver (L293D or L298N),External 5–9V battery, motor ,Jumper wires ,Breadboard

Software: Arduino IDE **,**DHT Library (Adafruit or SimpleDHT)

Schematic Diagram (Description)



Code:

Here is the complete working code:

```
#include <DHT.h>

#define DHTPIN 2           // DHT11 data pin

#define DHTTYPE DHT11 DHT

dht(DHTPIN, DHTTYPE);

int ledPin = 13;           // LED pin int

motorPin1 = 9;         // Motor driver pin 1 int

motorPin2 = 10;        // Motor driver pin 2

// Threshold values float tempThreshold = 30.0;

// Temperature in °C float humidityThreshold =

70; // Humidity % void setup() {

Serial.begin(9600);   dht.begin();

pinMode(ledPin, OUTPUT);

pinMode(motorPin1, OUTPUT);

pinMode(motorPin2, OUTPUT);

} void loop()

{

  // Reading humidity and temperature

float h = dht.readHumidity();   float t

= dht.readTemperature();

  // Display values on Serial Monitor

  Serial.print("Temperature: ");

  Serial.print(t);

  Serial.print(" °C  | Humidity: ");

  Serial.print(h);

  Serial.println(" %");

// LED Control based on humidity

  if (h > humidityThreshold) {

digitalWrite(ledPin,  HIGH);      // LED ON
```

```
  } else {
    digitalWrite(ledPin, LOW);       // LED OFF
  }
 // Motor Control based on temperature   if (t >
tempThreshold) {     digitalWrite(motorPin1,
HIGH);   // Motor ON      digitalWrite(motorPin2,
LOW);
  } else {
    digitalWrite(motorPin1, LOW);    // Motor OFF
digitalWrite(motorPin2, LOW);
  }
  delay(1000);
}
```

Conclusion:

In this practical, we successfully interfaced the DHT11 temperature & humidity sensor, an LED, and a DC motor with Arduino. The system automatically monitored temperature and humidity levels and activated the LED and motor based on predefined threshold values.

Practical No. 4: Live Temperature and Humidity Monitoring with Arduino UNO

Objective:

To measure real-time temperature and humidity using the DHT11 sensor and store the collected data in a CSV file.

Introduction (Theory Section)

The DHT11 sensor is a widely used digital sensor capable of measuring temperature and humidity. Using Arduino, we can monitor environmental conditions in real-time and make automatic decisions.

In many real-world applications—such as smart homes, greenhouses, server rooms, and agriculture—devices must respond automatically to environmental changes.
Examples:

- Turning ON a fan when temperature is high ▯ Indicating humidity changes with LEDs
- Activating motors for ventilation or cooling

In this practical, Arduino continuously reads temperature and humidity from the DHT11 sensor. Based on threshold values:

- An LED is turned ON/OFF as an indicator
- A DC motor (acting as a fan) is automatically controlled

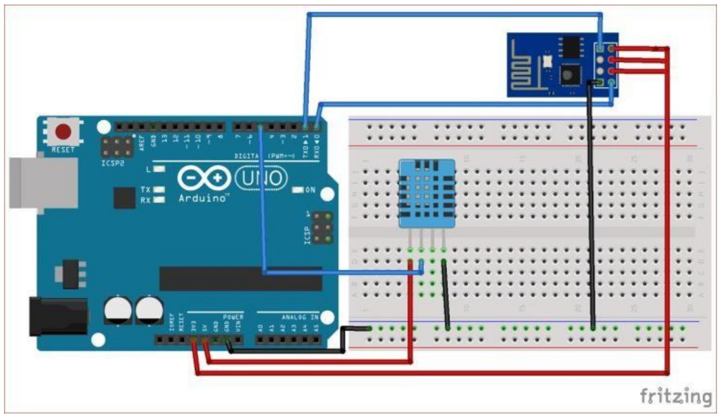
This experiment demonstrates the fundamentals of IoT automation, sensor interfacing, digital output control, and embedded decision-making.

Components Required:

Hardware: Arduino UNODHT11 Temperature & Humidity SensorLED (1 pc)220Ω resistorJumper wiresBreadboard

Software: Arduino IDE

Schematic Diagram (Description)



Code:

Here is the complete working code:

```
#include <DHT.h>

#define DHTPIN 2          // DHT11 data pin

#define DHTTYPE DHT11 DHT

dht(DHTPIN, DHTTYPE);

int ledPin = 13;          // LED pin int

motorPin1 = 9;        // Motor driver pin 1 int

motorPin2 = 10;        // Motor driver pin 2

// Threshold values float tempThreshold = 30.0;

// Temperature in °C float humidityThreshold =

70; // Humidity % void setup() {

Serial.begin(9600);   dht.begin();

pinMode(ledPin, OUTPUT);

pinMode(motorPin1, OUTPUT);

pinMode(motorPin2, OUTPUT);

}

void loop() { // Reading humidity and temperature

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

dht.readTemperature();

 // Display values on Serial Monitor
```

```
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" °C  | Humidity: ");
  Serial.print(h);
  Serial.println(" %");
// LED Control based on humidity   if (h >
humidityThreshold) {     digitalWrite(ledPin,
HIGH);      // LED ON
  } else {
    digitalWrite(ledPin,  LOW);       // LED OFF
  }
  // Motor Control based on temperature   if (t >
tempThreshold) {     digitalWrite(motorPin1,
HIGH);   // Motor ON     digitalWrite(motorPin2,
LOW);
  } else {
    digitalWrite(motorPin1, LOW);    // Motor OFF
digitalWrite(motorPin2, LOW);
  } delay(1000);
}
```

Conclusion:

In this practical, we successfully interfaced the DHT11 temperature & humidity sensor, an LED, and a DC motor with Arduino. The system automatically monitored temperature and humidity levels and activated the LED and motor based on predefined threshold values. This experiment demonstrates how sensors and actuators work together in an embedded system and forms the basis for real-world IoT applications such as smart cooling systems, greenhouse automation, and environmental monitoring.

Practical No. 5: Exploration of ESP32 Microcontroller

Objective:

To study the architecture, pin configuration, applications, and Arduino IDE setup of the ESP32 microcontroller and to blink the inbuilt LED.

Introduction

The ESP32 is a powerful, low-cost, Wi-Fi + Bluetooth-enabled microcontroller developed by *Espressif Systems*. It is widely used in IoT applications due to its high performance, dual-core processor, rich peripherals, and wireless communication features.

Key Features of ESP32

- Dual-core Tensilica LX6 CPU (up to 240 MHz)
- Built-in Wi-Fi (802.11 b/g/n)
- Bluetooth 4.2 & BLE
- 520 KB SRAM + external flash memory
- Multiple ADC, DAC, PWM, I2C, SPI, UART interfaces
- Low power consumption modes
- Capacitive touch pins
- Built-in hall sensor, temperature sensor
- Secure encryption support for IoT applications

Applications of ESP32

- IoT devices
- Smart home systems
- Weather monitoring systems
- Smart agriculture solutions
- Wearable devices
- Robotics and automation
- Wireless data logging
- Bluetooth beacons

- Arduino IDE Setup for ESP32

To program ESP32 using the Arduino IDE:

1. Open File → Preferences
2. Add this URL in *Additional Board Manager URLs:* https://dl.espressif.com/dl/package_esp32_index.json
3. Open Tools → Board → Boards Manager
4. Search ESP32 → Install
5. Select Tools → Board: ESP32 Dev Module
6. Select Port → Upload code **Components Required:**

- ESP32 Development Board
- USB Cable (Micro-USB or Type-C depending on board)
- Laptop with Arduino IDE

Schematic Diagram (Description):

Code to Blink Inbuilt LED (ESP32)

```
// Practical No. 5: ESP32 LED Blink


int ledPin = 2;   // Inbuilt LED on GPIO2 (on most ESP32 boards)


void setup() {   pinMode(ledPin, OUTPUT);   // Configure
pin as output
}


void loop() {   digitalWrite(ledPin,
HIGH);  // LED ON   delay(1000);
// 1 second delay   digitalWrite(ledPin,
LOW);   // LED OFF   delay(1000);
// 1 second delay
}
```

If your ESP32 board uses a different LED pin, I can modify it.

Conclusion:

In this practical, we explored the ESP32 microcontroller, its architecture, pin configuration, and applications. We also learned how to set up the Arduino IDE to program the ESP32 and successfully uploaded a simple LED blinking program. This experiment helps understand the basics of programming ESP32 and prepares students for advanced IoT applications involving sensors, cloud connectivity, and wireless communication.

Practical No. 6: Connecting ESP32 to Wi-Fi Network

Objective:

To understand the process of connecting the ESP32 microcontroller to a Wi-Fi network and verify the connection using the Arduino IDE Serial Monitor.

Introduction

The ESP32 is a Wi-Fi and Bluetooth-enabled microcontroller widely used in IoT applications. One of its core features is the built-in Wi-Fi module, which allows devices to connect to:

- Local networks
- IoT cloud platforms

When ESP32 connects to a Wi-Fi network, it receives:

- An IP address

Using the Serial Monitor, we can verify:

- Connection attempts
- Connection success/failure
- Assigned IP address

This practical builds the foundation for cloud communication, data logging, home automation, and real-time IoT applications.

Components Required:

Hardware:

- ESP32 Development Board

Software: Arduino IDE **,**ESP32 Board Package

Schematic Diagram:

Since this practical only involves Wi-Fi connectivity, no wiring or external components are required.

ESP32 → Laptop (via USB) → Arduino IDE → Wi-Fi Router

Code (ESP32 Wi-Fi Connection Program)

```
#include <WiFi.h> const char* ssid = "Your_WiFi_Name";        // Replace with your
Wi-Fi name const char* password = "Your_WiFi_Password"; // Replace with your Wi-
Fi password void setup() {   Serial.begin(115200);   delay(1000);
Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);   // Attempt until
connected   while (WiFi.status() !=
WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
// Successful connection
  Serial.println("");
  Serial.println("Connected Successfully!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
} void loop()
{   //
Nothing
required
here for this

practical

}
```

What Serial Monitor Displays

```
Connecting to WiFi...

......

Connected Successfully!

IP Address: 192.168.x.x
```

Conclusion:

In this practical, we successfully connected the ESP32 microcontroller to a Wi-Fi network using the Arduino IDE. We observed connection progress and verified the assigned IP address using the Serial Monitor. This practical is essential for developing IoT applications where ESP32 communicates with cloud platforms, servers, mobile apps, and smart home ecosystems. It establishes the foundation for upcoming experiments involving MQTT, HTTP, Blynk, Firebase, and API integration.

Practical No. 7: IoT Cloud Data Storage using Blynk App

Objective:

To understand the use of the Blynk App for IoT applications and to create a project that

Introduction

The Blynk IoT platform is a cloud-based service that enables remote monitoring, real-time data visualization, and control of IoT devices through mobile apps and web dashboards. It provides features like:Virtual pins,Cloud storage,Real-time data graphs,Device control widgets (buttons, sliders, switches),Notifications,Dashboards and templates

Using Blynk, IoT devices like ESP32 or ESP8266 can securely send data (temperature, humidity, sensor readings) to the cloud and receive control commands (turn ON/OFF LEDs, motors, relays).

In this practical, we send temperature & humidity values from the ESP32/ESP8266 to the Blynk Cloud and visualize it using the Blynk mobile app. This demonstrates IoT cloud integration, virtual pin communication, and remote monitoring.

Components Required:

Hardware:ESP32 / ESP8266 NodeMCU,DHT11 / DHT22 Sensor,Jumper wires,Breadboard,USB cable

Software: Arduino IDE ,Blynk IoT Mobile App

Schematic Diagram (Description):

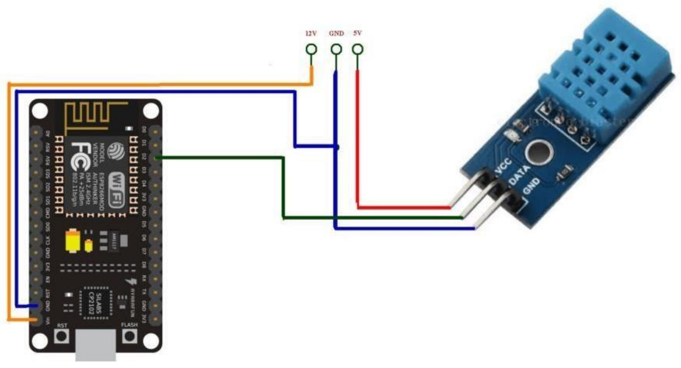
Connections for DHT11 with ESP32

DHT11 VCC → 3.3V

DHT11 GND → GND

DHT11 Data → GPIO 4 (or any digital pin)

ESP32 → USB Cable → Laptop → Arduino IDE → Blynk Cloud



Code (ESP32 with Blynk Cloud & DHT11)

Replace

- "Your_SSID" with your Wi-Fi name
- "Your_Password" with your password
- "Your_Blynk_AuthToken" with your device token

```
#define BLYNK_TEMPLATE_ID "Your_Template_ID"

#define BLYNK_DEVICE_NAME "Blynk IoT Device"

#define BLYNK_AUTH_TOKEN "Your_Blynk_AuthToken"

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <DHT.h>


char ssid[] = "Your_SSID";         char

pass[] = "Your_Password";


#define DHTPIN 4

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);


BlynkTimer timer;
```

```
// Function to send sensor data void
sendSensorData() {   float humidity =
dht.readHumidity();   float temperature =
dht.readTemperature();

  Blynk.virtualWrite(V0, temperature);
  Blynk.virtualWrite(V1, humidity);

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print("  | Humidity: ");
  Serial.println(humidity);
}

void setup() {
Serial.begin(115200);  dht.begin();
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  timer.setInterval(2000L, sendSensorData);
}

void loop() {
Blynk.run();  timer.run();
}
```

What You Will See in Blynk App

- Real-time temperature updates on V0
- Real-time humidity updates on V1

- Live graph data
- Cloud-stored data
- Dashboard values changing automatically

Conclusion:

In this practical, we learned how to use the Blynk IoT Cloud to store and visualize sensor data from an ESP32/ESP8266 device. The DHT11 sensor readings were successfully uploaded to the cloud using virtual pins and monitored using the Blynk mobile dashboard. This experiment demonstrates the core concepts of cloud connectivity, virtual pin communication, IoT data logging, and remote monitoring—essential skills for building modern IoT applications.

Practical No. 8: Controlling Inbuilt LED using Blynk App and ESP32

Objective:

To learn how to control the inbuilt LED of an ESP32 board remotely using the Blynk IoT app through a virtual switch.

Introduction

The ESP32 microcontroller supports built-in Wi-Fi, making it ideal for IoT applications. The Blynk IoT App allows users to remotely control and monitor IoT devices using virtual widgets such as switches, sliders, and buttons.

In this practical, we create a simple IoT control system where:

- The user toggles a switch on the Blynk Mobile App
- The switch sends a command to the Blynk Cloud
- The ESP32 receives the command and toggles the inbuilt LED (GPIO 2)

This experiment demonstrates:

- Cloud-based device control
- Virtual pin usage (V0, V1, etc.)
- Real-time communication between ESP32 and Blynk
- Basic IoT automation

This is one of the simplest yet most important IoT tasks because it teaches how devices are controlled over the internet.

Name of Activity:

Controlling ESP32 Inbuilt LED using Blynk Virtual Switch

Components Required:

Hardware:

- ESP32 Development Board
- USB Cable

Software:

- Arduino IDE
- Blynk IoT App
- Blynk Device Template
- Wi-Fi network

(No external components or wiring are needed)

Schematic Diagram:

This practical requires no external circuit.

Representation:

Blynk App (Virtual Switch) → Blynk Cloud → ESP32 → Inbuilt LED (GPIO 2)

GPIO2
Gnd
ON
Blynk

Code (ESP32 LED Control using Blynk Virtual Pin)

Replace the following before uploading:

- BLYNK_TEMPLATE_ID
- BLYNK_DEVICE_NAME
- BLYNK_AUTH_TOKEN
- Your Wi-Fi SSID & Password
- Make sure the Blynk switch widget is linked to Virtual Pin V0

```
#define BLYNK_TEMPLATE_ID "Your_Template_ID"
#define BLYNK_DEVICE_NAME "ESP32 LED Control"
#define BLYNK_AUTH_TOKEN "Your_Blynk_AuthToken"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "Your_WiFi_Name";        char
pass[] = "Your_WiFi_Password";

int ledPin = 2;  // Inbuilt LED for ESP32

// Virtual pin handler for V0
BLYNK_WRITE(V0) {
  int ledState = param.asInt();    digitalWrite(ledPin,
ledState);
}

void setup() {
  Serial.begin(115200);


  pinMode(ledPin,                    OUTPUT);
digitalWrite(ledPin, LOW);


  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}


void loop() {
  Blynk.run();
}
```

Conclusion:

In this practical, we successfully controlled the inbuilt LED of the ESP32 board using the Blynk IoT App. A virtual switch was used to send commands through the Blynk Cloud, and the ESP32 responded instantly by turning the LED ON or OFF. This demonstrates the working of IoT remote control, virtual pins, and cloud-based device communication—forming a foundation for more advanced applications such as home automation, smart appliances, and IoT dashboards.

Practical No. 9: Displaying City Weather using OpenWeather API

Objective:

To understand how to use the OpenWeather API to fetch and display real-time weather data such as temperature, humidity, and weather conditions for a specific city.

Introduction

The OpenWeather API is a cloud-based weather information service that provides real-time and forecasted data for any city in the world. It is widely used in IoT applications, mobile apps, dashboards, and automation systems.

The API returns weather data in JSON format, which includes:

- Temperature
- Humidity
- Weather conditions (clear, cloudy, rain, etc.)
- Wind speed
- Atmospheric pressure
- Sunrise/Sunset timing

Microcontrollers like ESP32 can connect to Wi-Fi and access internet APIs using HTTP requests. When the ESP32 sends a request with the city name and API key, the OpenWeather server returns live weather data.

This practical demonstrates:

- API usage
- JSON data parsing
- Wi-Fi + HTTP request handling Real-time IoT data display

Name of Activity:

Fetching and Displaying Real-Time Weather Data of a City using ESP32 and OpenWeather API

Components Required:

Hardware:

- ESP32 Development Board
- USB cable **Software:**
- Arduino IDE

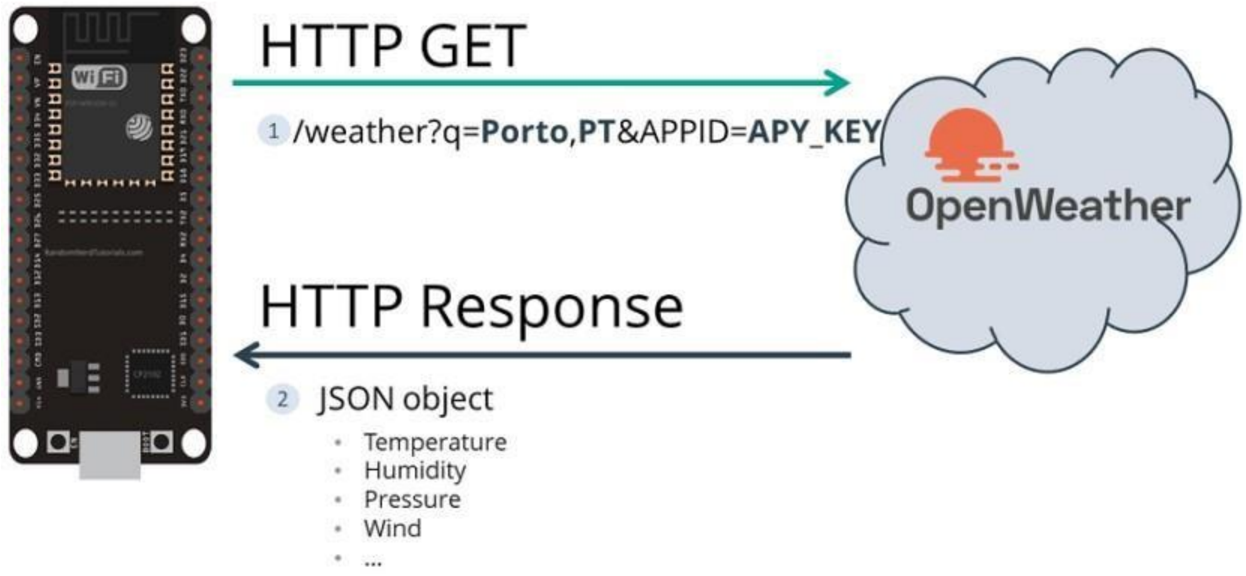
- OpenWeather API Key
- Wi-Fi network
- Arduino JSON library HTTPClient library

Schematic Diagram:

Since this practical uses only Wi-Fi and HTTP, no external components are required.

Representation:

ESP32 → Wi-Fi Router → OpenWeather API Server → JSON Weather Data



Code (ESP32 Fetching Weather from OpenWeather API)

Before uploading, replace:

- "Your_SSID" → your Wi-Fi name
- "Your_PASSWORD" → your Wi-Fi password
- "YOUR_API_KEY" → your OpenWeather API key
- "City_Name" → city you want to fetch weather for

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

const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";

String apiKey = "YOUR_API_KEY";
String city = "City_Name";   // Example: Mumbai
String server = "http://api.openweathermap.org/data/2.5/weather?q=" + city +
"&appid=" + apiKey + "&units=metric";

void                  setup()                  {
Serial.begin(115200);
  delay(1000);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED) {
delay(500);      Serial.print(".");
  }

  Serial.println("\nConnected!");
}

void loop() {
  if      (WiFi.status()      ==      WL_CONNECTED)      {
HTTPClient http;
    http.begin(server);

    int httpResponseCode = http.GET();

    if    (httpResponseCode    ==    200)    {
String payload = http.getString();

      // JSON document
      DynamicJsonDocument doc(2048);
      deserializeJson(doc, payload);

      float    temperature    =    doc["main"]["temp"];
float humidity = doc["main"]["humidity"];
      const char* condition = doc["weather"][0]["description"];

      Serial.println("------------------");
      Serial.println("CITY WEATHER REPORT");
      Serial.print("Temperature: ");
      Serial.print(temperature);
      Serial.println(" °C");

      Serial.print("Humidity: ");
      Serial.print(humidity);
      Serial.println(" %");

      Serial.print("Condition: ");
      Serial.println(condition);
      Serial.println("----------------------------- ");
    }
else {
      Serial.print("Error fetching data. Code: ");
      Serial.println(httpResponseCode);
    }

    http.end();
  }

  delay(60000);  // Refresh every 60 seconds
}
```

Conclusion:

In this practical, we successfully learned how to fetch real-time weather data using the OpenWeather API and display it through the ESP32 Serial Monitor. The ESP32 connected to the internet, sent an HTTP request, received a JSON response, and extracted important weather parameters such as temperature, humidity, and conditions. This experiment demonstrates how IoT devices communicate with web APIs, enabling smart weather monitoring systems, automated climate control, and advanced IoT dashboards.