SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

ACADEMIC YEAR: 2025-2026

ODD SEMESTER

LAB MANUAL

(REGULATION - 2024)

MC4364 – INTERNET OF THINGS LABORATORY

THIRD SEMESTER

MCA

Prepared By

N.JOTHI , A.P (O.G) / MCA

P.PANDI DEEPA,A.P (O.G) / MCA

MC4364	INTERNET OF THINGS LABORATORY	L	T	P	C
		0	0	4	2

COURSE OBJECTIVES:

- To design applications to interact with sensors.
- To design and develop IoT application Arduino/Raspberry pi for real world scenario.
- To enable communication between IoT and cloud platforms.
- To develop applications using Django Framework.

LIST OF EXPERIMENTS:

1. To study various IoT protocols – 6LowPAN, IPv4/IPv6, Wifi, Bluetooth, MQTT.
2. IoT Application Development Using sensors and actuators (temperature sensor, light sensor, infrared sensor)
3. To study Raspberry Pi development board and to implement LED blinking applications.
4. To develop an application to send and receive data with Arduino using HTTP request
5. To develop an application that measures the room temperature and posts the temperature value on the cloud platform.
6. To develop an application that measures the moisture of soil and post the sensed data over Google Firebase cloud platform.
7. To develop an application for measuring the distance using ultrasonic sensor and post distance value on Google Cloud IoT platform
8. Develop a simple application based on sensors.
9. Develop IoT applications using Django Framework and Firebase/ Bluemix platform.
10. Develop a commercial IoT application.

TOTAL: 60 PERIODS

HARDWARE/SOFTWARE REQUIREMENTS:

1. The universal microcontroller development board
2. 8051 Daughter Board
3. Raspberry Pi 3B+ Original
4. Arduino Daughter Board
5. Humidity + IR Sensor Interface
6. Ultrasonic Sensors
7. Open source software's Django Framework
8. Open cloud architectures like Bluemix, Development platforms like Firebase

COURSE OUTCOMES (COs):

On completion of the laboratory course, the student should be able

to CO1: To understand the various IoT protocols.

CO2: Test and experiment different sensors for application development.

CO3: To develop applications using Arduino/Raspberry Pi/ Equivalent boards.

CO4: To develop applications that would read the sensor data and post it in Cloud. CO5: Develop IOT applications with different platforms and frameworks.

CO's- PO's MAPPING

Course Outcomes	PROGRAM OUTCOMES					
	1	**2**	**3**	**4**	**5**	**6**
CO1	2	1	2	2	2	2
CO2	2	1	2	2	2	2
CO3	2	1	2	2	2	2
CO4	2	1	2	2	2	2
CO5	2	1	2	2	2	2
AVG	2	1	2	2	2	2

1.PROGRAMME EDUCATIONAL COURSE OBJECTIVES (PEOs):

PEO 1: To equip students with a strong foundation to design and develop solutions in mathematical, scientific, data analytics and business applications.

PEO 2: To develop the ability to plan, analyze, design, code, test, implement and maintain the software product for real time systems.

PEO 3: To practice effectively as individuals and as team members in multidisciplinary projects involving technical, managerial, economic and social constraints.

PEO 4: To progress their career productively in software industry, academia, research, entrepreneurial pursuit, government, consulting firms and other Information Technology enabled services.

2.PROGRAMME COURSE OUTCOMES (POs):

After going through the two years of study, our master's in computer applications graduates will exhibit the ability to:

PO#	PROGRAMME COURSE OUTCOMES
1.	An ability to independently carry out research/investigation and development work to solve practical problems.
2.	An ability to write and present a substantial technical report/document.
3.	An ability to demonstrate the design and development of computer applications for any domains.
4.	An ability to create, select, adapt and apply appropriate innovative techniques, resources, and modern computing tools to complex computing activities with an understanding of the limitations.
5.	An ability to recognize the need and to engage in independent learning for continual development as a computing professional.
6.	An ability to function effectively as an individual and as a member/leader of a team in various technical environments.

3.PEO / PO Mapping:

PROGRAMME EDUCATIONAL COURSE OBJECTIVES	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
I	3	1	3	3	3	3
II	2	1	3	3	3	3
III	3	3	2	3	2	2
IV	2	1	2	1	1	2

(3 – High, 2 – Medium, 1 – Low)

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	30
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	10
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	25
2.	Record	10
3.	Model Test	15
Total		**50**

TABLE OF CONTENTS	
EX.NO.	**NAME OF THE EXPERIMENT**
1	To study various IoT protocols – 6LowPAN, IPv4/IPv6, Wifi, Bluetooth, MQTT.
2	IoT Application Development Using sensors and actuators (temperature sensor, light sensor, infrared sensor)
3	To study Raspberry Pi development board and to implement LED blinking applications.
4	To develop an application to send and receive data with Arduino using HTTP request
5	To develop an application that measures the room temperature and posts the temperature value on the cloud platform.
6	To develop an application that measures the moisture of soil and post the sensed data over Google Firebase cloud platform.
7	To develop an application for measuring the distance using ultrasonic sensor and post distance value on Google Cloud IoT platform
8	Develop a simple application based on sensors.
9	Develop IoT applications using Django Framework and Firebase/ Bluemix platform.
10	Develop a commercial IoT application.

EXP NO:1

DATE: **To study various IoT protocols – 6LowPAN, IPv4/IPv6, Wifi, Bluetooth, MQTT**

AIM:

To study and understand the working of various IoT communication protocols including 6LoWPAN, IPv4/IPv6, WiFi, Bluetooth, and MQTT, and to demonstrate practical communication using MQTT and Bluetooth protocols.

Software Tools Required

- MQTT Broker (Public: HiveMQ, Mosquitto MQTT Broker)
- MQTT Client tools (Mosquitto Client tools, MQTT.fx, or Python paho-mqtt library)
- Bluetooth tools (Bluetoothctl on Linux, PuTTY/Serial Terminal for Windows)
- Arduino IDE (for Arduino based Bluetooth/MQTT demos)
- Python (with paho-mqtt and pybluez libraries for MQTT and Bluetooth demos)
- Operating System: Linux (Ubuntu/Raspberry Pi OS), Windows or macOS

PROCEDURE:

Part 1: Theory and Protocol Overview

1. Study the features, advantages, and applications of each protocol:
    - 6LoWPAN: IPv6 adaptation for low-power wireless networks.
    - IPv4/IPv6: Network layer IP addressing protocols.
    - WiFi: Wireless LAN communication with high data rates.
    - Bluetooth: Short-range device communication.
    - MQTT: Lightweight publish-subscribe messaging protocol.
2. Prepare a comparison table based on range, data rate, power consumption, and use cases.

Part 2: MQTT Communication Setup

1. Install Mosquitto MQTT broker (if local) or use public broker like `broker.hivemq.com`.
2. On the first device, subscribe to a topic:

```
bash
mosquitto_sub -h broker.hivemq.com -t "test/topic"
```

3. On the second device, publish a message to the same topic:

```
bash

mosquitto_pub -h broker.hivemq.com -t "test/topic" -m "Hello from MQTT"
```

4. Observe the message reception on the subscriber device.

Part 3: Bluetooth Communication Setup

1. Enable Bluetooth on two devices (e.g., Raspberry Pi, laptops).
2. Use `bluetoothctl` to scan and pair devices:

```
bash

bluetoothctl
scan on
pair <device_mac_address>
trust <device_mac_address>
connect <device_mac_address>
```

3. Establish Serial Port Profile (SPP) connection using `rfcomm`:

```
bash

sudo rfcomm connect /dev/rfcomm0 <device_mac_address> 1
```

4. Use a terminal application to send/receive messages via Bluetooth serial.

PROGRAM:

MQTT Publisher :

```
import paho.mqtt.client as mqtt

broker = "broker.hivemq.com"
topic = "test/topic"

client = mqtt.Client()
client.connect(broker, 1883, 60)
client.publish(topic, "Hello from MQTT Publisher!")
client.disconnect()
print("Message Published")
```

MQTT Subscriber :

```
import paho.mqtt.client as mqtt

broker = "broker.hivemq.com"
topic = "test/topic"

def on_message(client, userdata, message):
    print(f"Received message: {str(message.payload.decode('utf-8'))}")
```

```
client = mqtt.Client()
client.connect(broker)
client.subscribe(topic)
client.on_message = on_message
client.loop_forever()
```

Bluetooth Communication (Linux Terminal Commands)
bash

```
# On device A - scan and pair
bluetoothctl
scan on
pair <MAC_address_of_device_B>
trust <MAC_address_of_device_B>
connect <MAC_address_of_device_B>

# On device A - connect serial
sudo rfcomm connect /dev/rfcomm0 <MAC_address_of_device_B> 1

# Open terminal on /dev/rfcomm0 for communication
screen /dev/rfcomm0 9600
```

OUTPUT:

Received message: Hello from MQTT Publisher!

RESULT:

Thus the characteristics, features, and use cases of various IoT protocols including 6LoWPAN, IPv4/IPv6, WiFi, Bluetooth, and MQTT was studied successfully

EXP NO: 2

IoT Application Development Using sensors and actuators (temperature sensor, light sensor, infrared sensor)

DATE:

AIM:

To develop an IoT application that reads data from temperature, light, and infrared sensors and controls actuators based on the sensor inputs using Arduino/Raspberry Pi.

Software Tools Required

- Arduino IDE (for Arduino programming) or Python IDE (for Raspberry Pi)
- Sensor libraries (e.g., DHT library for temperature sensor)
- Serial Monitor or Terminal for output monitoring
- Optional: MQTT client for data publishing (if connecting to cloud)

Hardware Required

- Arduino Board or Raspberry Pi
- Temperature Sensor (e.g., DHT11/DHT22)
- Light Sensor (LDR)
- Infrared Sensor (IR receiver or IR proximity sensor)
- Actuators: LED, Buzzer, Relay module (depending on use case)
- Connecting wires, breadboard, resistors

PROCEDURE

1. **Hardware Setup:**
    - Connect the temperature sensor to Arduino/Raspberry Pi GPIO pins.
    - Connect the LDR (light sensor) with a voltage divider circuit to an analog input pin.
    - Connect the infrared sensor to a digital input pin.
    - Connect actuators (LEDs/buzzer) to digital output pins.
2. **Write Code:**
    - Initialize sensors and actuators in the program.
    - Read sensor values periodically.
    - Based on sensor values, control the actuators (e.g., turn on LED if light intensity is low).
    - Print sensor readings and actuator states to serial monitor.
3. **Upload and Run:**
    - Upload the program to Arduino or run Python script on Raspberry Pi.
    - Observe sensor readings and actuator responses on serial monitor and hardware.

PROGRAM:

```
#include <DHT.h>

#define DHTPIN 2          // Pin connected to DHT sensor
#define DHTTYPE DHT11     // DHT 11 sensor
#define LDR_PIN A0        // Light sensor connected to analog pin A0
#define IR_PIN 3          // Infrared sensor connected to digital pin 3
#define LED_PIN 13        // Built-in LED pin
#define BUZZER_PIN 8      // Buzzer connected to pin 8

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
  pinMode(LED_PIN, OUTPUT);
 pinMode(BUZZER_PIN, OUTPUT);
  pinMode(IR_PIN, INPUT);
}

void loop() {
  // Read temperature
  float temp = dht.readTemperature();

  // Read light sensor
  int lightValue = analogRead(LDR_PIN);

  // Read infrared sensor
  int irValue = digitalRead(IR_PIN);

  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.print(" °C, Light: ");
  Serial.print(lightValue);
  Serial.print(", IR Sensor: ");
  Serial.println(irValue);

 // Control actuators based on sensor values
 if(lightValue < 300) {
    digitalWrite(LED_PIN, HIGH);   // Turn ON LED if dark
  } else {
    digitalWrite(LED_PIN, LOW);
  }

  if(irValue == HIGH) {
    digitalWrite(BUZZER_PIN, HIGH); // Turn ON buzzer if IR detected
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }

  delay(2000); // Wait for 2 seconds
   }
```

OUTPUT:

- Serial Monitor displays temperature in °C, light sensor analog value, and IR sensor digital value.
- LED turns ON when light intensity is low (dark environment).
- Buzzer sounds when the infrared sensor detects an object (IR sensor output HIGH).

```
yaml

Temperature: 26.00 °C, Light: 150, IR Sensor: 1
Temperature: 26.50 °C, Light: 450, IR Sensor: 0
Temperature: 27.00 °C, Light: 100, IR Sensor: 1
```

RESULT:

Thus, the experiment helped in understanding the basic interfacing of sensors and actuators in IoT applications.

EXP NO:3 **To study Raspberry Pi development board and to implement LED blinking applications**

DATE:

Aim

To study the Raspberry Pi development board architecture and GPIO pin configuration, and to implement an LED blinking application using the Raspberry Pi GPIO pins.

Software Tools Required

- Raspberry Pi OS (Raspbian) installed on Raspberry Pi
- Python 3 (pre-installed on Raspberry Pi)
- GPIO Python library (`RPi.GPIO` or `gpiozero`)
- Terminal or SSH client to access Raspberry Pi command line

Hardware Required

- Raspberry Pi 3B+ board (or equivalent)
- LED
- 220Ω resistor
- Breadboard and connecting wires

Procedure

1. **Setup Raspberry Pi:**
   - Power up the Raspberry Pi with Raspberry Pi OS installed.
   - Connect to Raspberry Pi via monitor and keyboard or SSH remotely.
2. **Hardware Connection:**
   - Connect the LED anode (long leg) to GPIO pin 17 (physical pin 11) through a 220Ω resistor.
   - Connect the LED cathode (short leg) to the Raspberry Pi ground (GND, physical pin 6).
3. **Write Python Program:**
   - Use Python GPIO library to control the GPIO pin connected to the LED.
   - The program will blink the LED ON and OFF with a delay.
4. **Run the Program:**
   - Save the Python script and run it in the terminal.
   - Observe the LED blinking.

PROGRAM:

```
import RPi.GPIO as GPIO
import time

# Use BCM pin numbering
GPIO.setmode(GPIO.BCM)

LED_PIN = 17
GPIO.setup(LED_PIN, GPIO.OUT)

try:
       while True:
                GPIO.output(LED_PIN, GPIO.HIGH)  # LED ON
                time.sleep(1)                  # Wait 1 second
               GPIO.output(LED_PIN, GPIO.LOW)   # LED OFF
               time.sleep(1)                  # Wait 1 second
except KeyboardInterrupt:
       print("Program stopped")

finally:
      GPIO.cleanup()  # Reset GPIO settings
```

OUTPUT:

- The LED connected to Raspberry Pi GPIO pin 17 blinks ON and OFF repeatedly in one-second intervals.
- The LED lights up for 1 second, then turns off for 1 second, continuously until the program is stopped.
- When the program is interrupted (Ctrl+C), the LED stops blinking and the GPIO pins are reset safely.

Program stopped

RESULT:

Thus, Raspberry Pi GPIO pins were successfully configured and controlled using Python

EXP NO:4

To develop an application to send and receive data with Arduino using HTTP request

DATE:

Aim

To develop an application where an Arduino sends sensor data to a server and receives commands or data via HTTP requests, demonstrating client-server communication over HTTP.

Software Tools Required

- Arduino IDE
- ESP8266 or ESP32 WiFi module/library (e.g., ESP8266WiFi.h or WiFi.h)
- Local or public HTTP server (e.g., using Postman Echo server or a simple Python Flask server)
- Serial Monitor for debugging
- Optional: Python/Node.js for server setup

Hardware Required

- Arduino board (e.g., Arduino Uno)
- ESP8266 WiFi module or Arduino with built-in WiFi (ESP32)
- Sensor (optional, e.g., temperature sensor like DHT11)
- Connecting wires and breadboard

Procedure

1. **Hardware Setup:**
   - Connect ESP8266 WiFi module to Arduino (if using Arduino Uno).
   - Connect sensor to Arduino if sensor data is required.
2. **Configure WiFi:**
   - In the program, add WiFi SSID and password to connect to the local WiFi network.
3. **HTTP Request Programming:**
   - Write Arduino code to:
     - Connect to WiFi.
     - Send HTTP GET or POST request to a server with sensor data or sample data.
     - Receive HTTP response from the server.
     - Parse and print response data to Serial Monitor.
4. **Setup Server (Optional):**
   - Run a simple HTTP server (e.g., Flask) to receive data and respond.
   - Or use public HTTP request testing servers like `httpbin.org` or Postman Echo.
5. **Upload and Run:**
   - Upload code to Arduino.
   - Open Serial Monitor to observe connection, requests sent, and responses received.

PROGRAM:

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

void setup() {
      Serial.begin(115200);
       WiFi.begin(ssid, password);
        Serial.print("Connecting to WiFi");

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

          Serial.println("\nWiFi connected");
          Serial.println("IP address: ");
          Serial.println(WiFi.localIP());
            }

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

        // Example HTTP GET request to httpbin.org/get
        http.begin("http://httpbin.org/get?data=HelloArduino");
        int httpCode = http.GET();

       if (httpCode > 0) {
              String payload = http.getString();
                Serial.println("HTTP Response code: " + String(httpCode));
                Serial.println("Response: " + payload);
        } else {
          Serial.println("Error on HTTP request");
        }

       http.end();
      } else {
      Serial.println("WiFi Disconnected");
        }

       delay(10000); // Wait 10 seconds before next request

}
```

OUTPUT:

- Arduino connects successfully to the WiFi network.
- HTTP GET request is sent to the server every 10 seconds.
- Server response is received and printed on the Serial Monitor.
- Sample output on Serial Monitor:

Connecting to WiFi...

.....

WiFi connected

IP address:

192.168.1.10

HTTP Response code: 200

Response: {

"args": {

"data": "HelloArduino"

},

...

}

RESULT:

Thus, the WiFi connection on Arduino using ESP8266 module was completed successfully.

EXP NO:5

To develop an application that measures the room temperature and posts the temperature value on the cloudplatform.

DATE:

Aim

To develop an IoT application that measures room temperature using a sensor and posts the temperature data to a cloud platform (e.g., Firebase, Thingspeak) for remote monitoring.

Software Tools Required

- Arduino IDE or Raspberry Pi Python environment
- Sensor libraries (e.g., DHT sensor library for temperature)
- Cloud platform account (e.g., Thingspeak, Firebase)
- HTTP client libraries (e.g., ESP8266HTTPClient for Arduino or `requests` for Python)
- Serial Monitor / Terminal for debugging

Hardware Required

- Arduino board with WiFi module (ESP8266/ESP32) or Raspberry Pi
- Temperature sensor (DHT11 or DHT22)
- Connecting wires, breadboard
- Power supply

Procedure

1. **Hardware Setup:**
    - Connect the temperature sensor (DHT11/DHT22) to the microcontroller GPIO pin.
    - Connect the WiFi module if required.
2. **Cloud Platform Setup:**
    - Create an account on the cloud platform (e.g., Thingspeak or Firebase).
    - Create a new channel or database to receive temperature data.
    - Note down the API key or URL endpoint for data posting.
3. **Programming:**
    - Initialize WiFi connection in the program.
    - Read temperature data from the sensor periodically.
    - Post temperature data to the cloud platform using HTTP POST/GET requests with the API key.
    - Print status messages to the Serial Monitor.
4. **Upload and Run:**
    - Upload the code to the Arduino or run the Python script on Raspberry Pi.
    - Monitor the Serial output for success messages.
    - Check the cloud platform dashboard to verify the received data.

PROGRAM:

```
#include <ESP8266WiFi.h>
#include <DHT.h>
#include <ESP8266HTTPClient.h>

#define DHTPIN 2        // GPIO pin where DHT sensor is connected
#define DHTTYPE DHT11

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

const char* server = "http://api.thingspeak.com/update";
const String apiKey = "YOUR_THINGSPEAK_API_KEY";

DHT dht(DHTPIN, DHTTYPE);

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

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

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

  Serial.println("\nConnected to WiFi");
}

void loop() {
 float temperature = dht.readTemperature();

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

Serial.print("Temperature: ");
 Serial.print(temperature);
Serial.println(" °C");  if (WiFi.status() == WL_CONNECTED) {
HTTPClient http;

  String postData = String(server) + "?api_key=" + apiKey + "&field1=" + String(temperature);
  http.begin(postData);
int httpCode = http.GET();

 if (httpCode > 0) {
  Serial.println("Data posted successfully");
  } else {
   Serial.println("Error in posting data");
```

```
}
http.end();
 } else {
  Serial.println("WiFi Disconnected");
 }

delay(20000); // Wait for 20 seconds before next reading
}
```

OUTPUT:

- Serial Monitor shows WiFi connection status.
- Displays temperature readings every 20 seconds.
- Displays success message when data is posted to the cloud.
- On the cloud platform dashboard, temperature values are updated periodically and displayed graphically.

Connecting to WiFi...
.....
Connected to WiFi
Temperature: 26.50 °C
Data posted successfully
Temperature: 26.60 °C
Data posted successfully

RESULT:

Thus the WiFi connection and communicated with the cloud platform was established successfully.

EXP NO:6

To develop an application that measures the moisture of soil and post the sensed data over Google Firebase cloud platform.

DATE:

Aim

To develop an IoT application that measures soil moisture using a soil moisture sensor and posts the sensed data to the Google Firebase cloud platform for remote monitoring.

Software Tools Required

- Arduino IDE (for Arduino) or Python environment (for Raspberry Pi)
- Firebase Realtime Database or Firestore setup
- Firebase Arduino Client Library (`FirebaseESP8266.h` or `FirebaseArduino.h`) for Arduino
- WiFi library (e.g., ESP8266WiFi.h for ESP8266)
- Serial Monitor for debugging

Hardware Required

- Arduino board with WiFi capability (e.g., ESP8266, ESP32) or Raspberry Pi
- Soil moisture sensor module
- Connecting wires and breadboard
- Power supply

Procedure

1. **Hardware Setup:**
    - Connect the soil moisture sensor output pin to an analog input pin of the microcontroller.
    - Connect power and ground pins appropriately.
2. **Firebase Setup:**
    - Create a Firebase project on Firebase Console.
    - Create a Realtime Database or Firestore database.
    - Note down the database URL and generate a database secret or authentication token (depending on Firebase rules).
    - Configure database rules to allow read/write during testing.
3. **Programming:**
    - Initialize WiFi connection in the program.
    - Initialize Firebase connection with the database URL and authentication credentials.
    - Periodically read analog value from soil moisture sensor.
    - Post the sensor value to Firebase database using the appropriate library functions.
    - Print sensor readings and Firebase update status to Serial Monitor.
4. **Upload and Run:**
    - Upload the program to the Arduino or run the Python script on Raspberry Pi.
    - Monitor Serial output for sensor data and Firebase update confirmation.
    - Verify data on Firebase console in real-time.

PROGRAM:

```
#include <ESP8266WiFi.h>
#include <FirebaseESP8266.h>

// Replace these with your network credentials
#define WIFI_SSID "YOUR_SSID"
#define WIFI_PASSWORD "YOUR_PASSWORD"

// Replace these with your Firebase project credentials
#define FIREBASE_HOST "your-project-id.firebaseio.com"
#define FIREBASE_AUTH "your-database-secret-or-auth-token"

// Soil moisture sensor analog pin
#define SOIL_MOISTURE_PIN A0

FirebaseData firebaseData;

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

 // Connect to WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
 delay(500);
  Serial.print(".");
  }
 Serial.println("\nConnected to WiFi");

 // Initialize Firebase
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  Firebase.reconnectWiFi(true);
}
```

```
void loop() {
  int soilMoistureValue = analogRead(SOIL_MOISTURE_PIN);
  Serial.print("Soil Moisture Value: ");
 Serial.println(soilMoistureValue);

  // Post value to Firebase under the node "soilMoisture"
  if (Firebase.setInt(firebaseData, "/soilMoisture", soilMoistureValue)) {
  Serial.println("Firebase updated successfully");
} else {
   Serial.print("Firebase update failed: ");
 Serial.println(firebaseData.errorReason());
}

 delay(20000);  // Post every 20 seconds
}
```

OUTPUT:

- Serial Monitor shows WiFi connection status.
- Displays soil moisture sensor analog readings every 20 seconds.
- Confirms successful data posting to Firebase or shows error messages.
- Soil moisture data updates in real-time on the Firebase console under `/soilMoisture`.

```
Connecting to WiFi...
.....
Connected to WiFi
Soil Moisture Value: 540
Firebase updated successfully
Soil Moisture Value: 520
Firebase updated successfully
```

RESULT:

Thus the Soil moisture data was periodically sent and updated on the Firebase cloud platform.

EXP NO:7 **To develop an application for measuring the distance using ultrasonic sensor and post distance value on Google Cloud IoT platform**

DATE:

Aim

To develop an IoT application that measures distance using an ultrasonic sensor and posts the measured distance value to the Google Cloud IoT platform for remote monitoring.

Software Tools Required

- Arduino IDE or Python environment (for Raspberry Pi)
- Google Cloud IoT Core SDK or MQTT client libraries (e.g., PubSubClient for Arduino)
- Google Cloud Platform account with IoT Core enabled
- Cloud SDK tools for setup (gcloud CLI)
- Serial Monitor or terminal for debugging

Hardware Required

- Arduino board with WiFi or Ethernet capability (ESP8266/ESP32) or Raspberry Pi
- Ultrasonic sensor module (HC-SR04)
- Breadboard, jumper wires
- Power supply

Procedure

1. **Hardware Setup:**
    - Connect the HC-SR04 ultrasonic sensor pins to the microcontroller as follows:
        - VCC to 5V
        - GND to Ground
        - Trig to a digital output pin (e.g., GPIO 5)
        - Echo to a digital input pin (e.g., GPIO 4)
2. **Google Cloud Setup:**
    - Create a Google Cloud project and enable IoT Core API.
    - Create a registry and device in IoT Core.
    - Generate device credentials (JWT tokens or RSA keys).
    - Note down MQTT bridge endpoints and device details.
3. **Programming:**
    - Write code to measure distance using the ultrasonic sensor:
        - Trigger the sensor.
        - Measure pulse duration.
        - Calculate distance in centimeters or inches.

- Use MQTT client to connect to Google Cloud IoT MQTT bridge.
- Authenticate using JWT tokens.
- Publish the distance data to a specific MQTT topic.
- Handle connection and reconnection.

4. **Upload and Run:**
   - Upload the code to the device or run the script on Raspberry Pi.
   - Monitor Serial output for sensor readings and connection status.
   - Verify published messages in Google Cloud Console via IoT Core or Pub/Sub subscription.

PROGRAM:

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>

// Ultrasonic sensor pins
#define TRIG_PIN 5
#define ECHO_PIN 4

// WiFi credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Google Cloud IoT Core details
const char* project_id = "your-project-id";
const char* location = "your-region"; // e.g., us-central1
const char* registry_id = "your-registry-id";
const char* device_id = "your-device-id";

const char* mqtt_server = "mqtt.googleapis.com";
const int mqtt_port = 8883;

WiFiClientSecure net;
PubSubClient client(net);

// Function prototypes
long readUltrasonicDistance();
String createJwt();

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

 pinMode(TRIG_PIN, OUTPUT);
 pinMode(ECHO_PIN, INPUT);

 WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi");
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
```

```
 }
  Serial.println("\nWiFi connected");

// Configure secure connection
 net.setCACert(your_root_ca_cert); // Add your root CA certificate here

  client.setServer(mqtt_server, mqtt_port);

  // Connect to MQTT with JWT authentication
  connectToCloudIoT();
}

void connectToCloudIoT() {
  while (!client.connected()) {
   String jwt = createJwt();
   Serial.println("Connecting to MQTT...");
   if (client.connect(device_id,
 "unused", jwt.c_str())) { // username unused, password is JWT
 Serial.println("Connected to MQTT");
 } else {
  Serial.print("Failed MQTT connection, rc=");
   Serial.println(client.state());
   delay(5000);
   }
 }
}

void loop() {
 if (!client.connected()) {
   connectToCloudIoT();
  }
  client.loop();

  long distance = readUltrasonicDistance();
Serial.print("Distance: ");
 Serial.print(distance);
Serial.println(" cm");

 // Publish distance to Google Cloud IoT topic
 String topic = "/devices/" + String(device_id) + "/events";
String payload = "{\"distance\":" + String(distance) + "}";

 if (client.publish(topic.c_str(), payload.c_str())) {
  Serial.println("Distance data published");
 } else {
Serial.println("Failed to publish");
 }




delay(20000); // Publish every 20 seconds
}
```

```
long readUltrasonicDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  long distanceCm = duration * 0.034 / 2;
  return distanceCm;
}

// Dummy JWT creation function (use JWT libraries or generate externally)
String createJwt() {
// Implement JWT token creation using your private key and project info
 return "YOUR_JWT_TOKEN";
}
```

OUTPUT

- Serial Monitor displays WiFi connection status.
- Displays ultrasonic sensor measured distance periodically.
- Shows MQTT connection status and message publish confirmation.
- Distance data appears on Google Cloud IoT platform in real time.

Sample Serial Output:

```
vbnet
Connecting to WiFi...
.........
WiFi connected
Connecting to MQTT...
Connected to MQTT
Distance: 45 cm
Distance data published
Distance: 44 cm
Distance data published
```

RESULT:

Thus measuring the distance using ultrasonic sensor and post distance value on Google Cloud IoT platform was developed successfully.

EXP NO: 8 **Develop a simple application based on sensors**

DATE

Aim

To develop a simple IoT application that reads data from a sensor (e.g., light sensor) and displays the sensor v alues on the Serial Monitor.

Software Tools Required

- Arduino IDE
- Sensor-specific libraries (if required)
- Serial Monitor for output

Hardware Required

- Arduino board (e.g., Arduino Uno)
- Light sensor (LDR - Light Dependent Resistor)
- 10kΩ resistor (for voltage divider)
- Breadboard and connecting wires
- Power supply

Procedure

1. **Hardware Setup:**
   - Connect one terminal of the LDR to 5V.
   - Connect the other terminal of the LDR to an analog input pin (e.g., A0) and also connect a 10kΩ resistor from this node to Ground to form a voltage divider.
   - Connect Arduino to the PC via USB.
2. **Programming:**
   - Open Arduino IDE and write code to read analog values from the sensor pin.
   - Print the sensor value to Serial Monitor continuously.
   - Upload the program to Arduino.
3. **Testing:**
   - Open Serial Monitor and observe the light sensor readings.
   - Cover the sensor or expose it to light and observe the changes in analog values.

PROGRAM:

```
const int sensorPin = A0;  // Analog pin connected to LDR

void setup() {
  Serial.begin(9600);
 Serial.println("Light Sensor Reading:");
}

void loop() {
  int sensorValue = analogRead(sensorPin);
  Serial.print("LDR Value: ");
 Serial.println(sensorValue);
  delay(1000); // Wait for 1 second
}
```

OUTPUT:

- Serial Monitor displays the light sensor readings continuously every second.
- The values vary depending on the amount of light falling on the LDR sensor.
- Lower values correspond to less light (darker), and higher values correspond to more light.

```
Light Sensor Reading:
LDR Value: 450
LDR Value: 460
LDR Value: 700
LDR Value: 800
LDR Value: 300
```

RESULT:

Successfully Understood basic sensor data reading and monitoring techniques for IoT applications.

EXP NO: 9

DATE

Develop IoT applications using Django Framework and Firebase/ Bluemix platform.

Aim

To develop an IoT web application using Django framework that interacts with sensor data stored and retrieved from Firebase or IBM Bluemix cloud platform.

Software Tools Required

- Python 3.x
- Django Framework
- Firebase Admin SDK for Python or IBM Bluemix SDK
- Firebase or IBM Bluemix account and project setup
- Web browser for testing
- Code editor (e.g., VS Code, PyCharm)

Hardware Required

- Any IoT device/sensor setup (optional for simulation)
- Network access to cloud platforms

Procedure

1. Setup Cloud Platform

- Create a project on Firebase or IBM Bluemix.
- For Firebase: Set up Realtime Database or Firestore and generate service account credentials JSON.
- For Bluemix: Set up the required cloud services and APIs.

2. Setup Django Project

- Install Django and Firebase Admin SDK via pip:

```
bash

pip install django firebase-admin
```

- Start a new Django project:

```
bash

django-admin startproject iotproject
cd iotproject
python manage.py startapp sensorapp
```

3. Configure Firebase in Django

- Place Firebase service account JSON in your project directory.
- Initialize Firebase in your Django app:

```
# sensorapp/firebase.py
import firebase_admin
from firebase_admin import credentials, firestore

cred = credentials.Certificate('path/to/serviceAccountKey.json')
firebase_admin.initialize_app(cred)
db = firestore.client()
```

4. Create Views to Fetch and Display Data

```
# sensorapp/views.py
from django.shortcuts import render
from .firebase import db

def sensor_data_view(request):
 docs = db.collection('sensor_data').stream()
 data = []
 for doc in docs:
data.append(doc.to_dict())
return render(request, 'sensorapp/data.html', {'sensor_data': data})
```

5. Define URL Routing

```
# iotproject/urls.py
from django.urls import path
from sensorapp.views import sensor_data_view

urlpatterns = [
path('sensor-data/', sensor_data_view, name='sensor_data'),
]
```

6. Create Template to Display Sensor Data

```
<!-- sensorapp/templates/sensorapp/data.html -->
<!DOCTYPE html>
<html>
<head>
 <title>Sensor Data</title>
</head>
<body>
 <h1>Sensor Data from Firebase</h1>
<ul>
 {% for item in sensor_data %}
     <li>{{ item.sensor_type }} : {{ item.value }} at {{ item.timestamp }}</li>
 {% empty %}
     <li>No sensor data found.</li>
 {% endfor %}
</ul>
</body>
</html>
```

7. Run the Server and Test

- Run Django development server:

  ```
  bash
  python manage.py runserver
  ```

- Open browser and navigate to `http://localhost:8000/sensor-data/` to view the sensor data fetched from Firebase.

Data Structure in Firebase:


```
{
 "sensor_data": [
  {
   "sensor_type": "temperature",
  "value": 26.5,
   "timestamp": "2025-06-26T10:00:00Z"
  },
  {
   "sensor_type": "humidity",
   "value": 60,
  "timestamp": "2025-06-26T10:05:00Z"
  }
 ]
}
```

OUTPUT:

- Web page displays a list of sensor data fetched from Firebase or Bluemix.
- Real-time or periodic updates can be implemented with advanced features.

Sensor Data from Firebase
- temperature : 26.5 at 2025-06-26T10:00:00Z
- humidity : 60 at 2025-06-26T10:05:00Z

RESULT:

Successfully Learned how to combine backend frameworks with cloud services for real-time IoT data visualization.

EXP NO: 10

DATE

Develop a commercial IoT application.

Aim

To design and develop a commercial-grade IoT application involving sensor data collection, cloud integration, secure communication, data analytics, and user interface for real-time monitoring and control.

Software Tools Required

- IoT device SDK (Arduino, ESP32, Raspberry Pi, etc.)
- Cloud platform (AWS IoT, Azure IoT, Google Cloud IoT, or IBM Bluemix)
- Backend framework (Node.js, Django, Flask, etc.)
- Database (Firebase, MongoDB, SQL, etc.)
- MQTT Broker or HTTP REST APIs
- Frontend framework (React, Angular, Vue.js, or mobile app frameworks)
- Security tools (TLS/SSL certificates, OAuth, JWT)
- Analytics and visualization tools (Grafana, Kibana, Google Data Studio)

Hardware Required

- IoT sensors and actuators (temperature, humidity, motion sensors, relays, etc.)
- Microcontroller boards (ESP32, Raspberry Pi, Arduino)
- Networking modules (WiFi, Ethernet, GSM)

Procedure

1. **Requirement Analysis**
    - Define the commercial use case (e.g., smart agriculture, smart home security, asset tracking).
    - Identify sensors and actuators required.
    - Specify data points to be collected, frequency, and control commands.
2. **Hardware Setup**
    - Connect sensors and actuators to microcontroller board.
    - Ensure power supply and networking modules are functional.
3. **Cloud Platform Configuration**
    - Choose cloud platform and set up IoT Core/Hub.
    - Register devices with unique identities and credentials.
    - Configure secure communication protocols (TLS, MQTT with authentication).
4. **Device Firmware Development**
    - Program device to read sensor data.
    - Implement data transmission using MQTT or REST API securely.
    - Handle command/control messages from cloud to device.

5. **Backend and API Development**
    - Create backend services to receive, process, and store IoT data.
    - Implement authentication, data validation, and business logic.
    - Design REST or WebSocket APIs for frontend communication.
6. **Frontend Application Development**
    - Develop a user-friendly dashboard for real-time monitoring and control.
    - Visualize sensor data using charts and alerts.
    - Allow user inputs to send commands back to devices.
7. **Security Implementation**
    - Use secure tokens (JWT/OAuth) for user and device authentication.
    - Encrypt data transmission with SSL/TLS.
    - Implement role-based access control.
8. **Testing and Validation**
    - Test end-to-end data flow: sensor → device → cloud → frontend.
    - Validate real-time updates, reliability, and latency.
    - Conduct security testing (penetration tests, vulnerability scans).
9. **Deployment and Maintenance**
    - Deploy backend and frontend on cloud infrastructure.
    - Set up monitoring and logging for system health.
    - Plan for firmware and software updates.

- 23 GPIO are digital-only, with three also being ADC-capable
- Can be surface mounted as a module
- 3-pin ARM serial wire debug (SWD) port
- Simple yet highly flexible power supply architecture
- Various options for easily powering the unit from micro-USB, external supplies, or batteries
- High quality, low cost, high availability
- Comprehensive SDK, software examples, and documentation
- Dual-core Cortex M0+ at up to 133MHz
- On-chip PLL allows variable core frequency
- 264kByte multi-bank high-performance SRAM

Example Use Case: Smart Home Energy Monitoring

- Devices measure electricity consumption via current sensors.
- Data sent every minute to cloud platform via MQTT.
- Backend stores data, analyzes usage patterns, and detects anomalies.
- Web/mobile dashboard displays real-time and historical energy usage.
- User receives alerts on unusual consumption and can remotely control appliances.

RESULT:

Thus developed a scalable, secure, and user-friendly commercial IoT application.