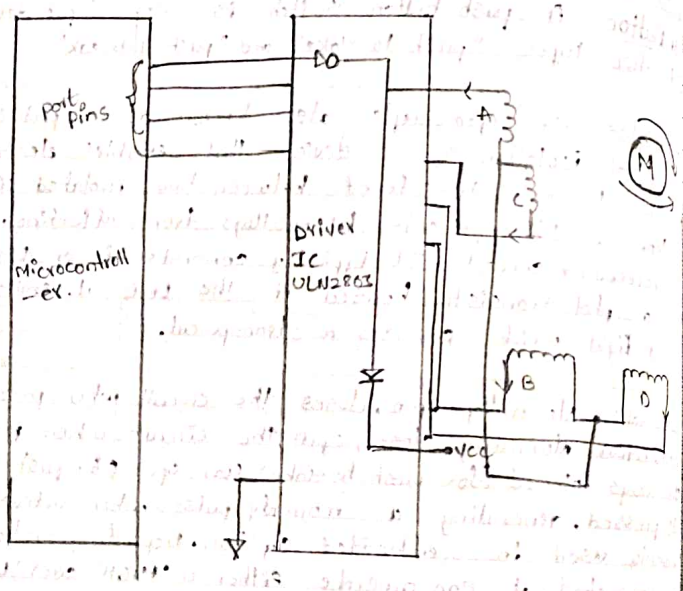



Interfacing of a Stepper motor

Expt. No.: 8. Page No. 5

Expt. Name: Study about Industrial Revolution and cyber physical System opportunities & Challenges. Date: 31-06-24

## Cyber physical System (Cps):-

It integrates computer based algorithms with physical mechanism intertwining software and hardware components across different scales the exhibit varied behaviours and interact contextually cps merges cybernetics, mechatronics, and process science, emphasizing a deep link between computational and physical elements

Example include Smart grids, autonomous vechiles, medical monito-ring, Industrial Control, robotics, and avionics. Cps akin to IOT, features enhance's coordination between physical and computational systems like aerospace, healthcare.

## Industry 4.0

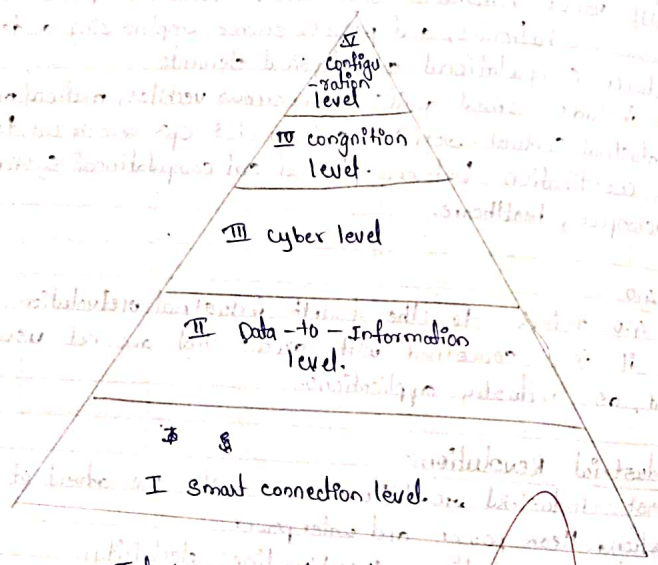
Industry 4.0 refers to the fourth industrial revolution although it is concerned with areas that are not usually classified as industry applications.

## Fourth industrial Revolution:-

1. The first industrial revolution came with the advent of mechanisation, steam power and water power
2. Second: Mass production, assembly lines, electricity,
3. Third: Electronics, IT, automation.
4. Fourth: cyber physical System, IOT, AI, Smart manufacturing.

## Industry 4.0 Technologies:-

Industry 4.0 Technologies include IOT, IIOT, cps, Smart manufacturi-ng, Smart factories, cloud computing and AI. These advancements enables automated manufacturing systems with wireless connectivity and sensors, facilitating realtime monitoring and autonomous




Industry 4.0 level Revelution.

Expt. No.: Page No. 6

Expt. Name: Date

decision making. The integration of 5G networks enhances communication speed digital twin technologies create virtual replicas for testing and decentralised decision making linked via IoT to enable collaboration between cyber physical systems and human operations overall industry 4.0 promotes digital transformation, enhancing productivity through inter connected & autonomous manufacturing processes.

## Example of the Industry 4.0 Revolution.

Industry 4.0 innovations include offline programming and adaptive control for arc welding, interacting product design simulation, & production on shop floors. Examples in automatic and smart factories world-wide highlight it's implementation challenges include differing design practices across engineering disciplines like software and mechanical engineering. Co-simulation advancements like the functional mock-up interface from project such as MODELISAR, facilitate collaborative System

## Cyber physical system opportunities.

The science of cps has the capability to impact technology in a wide variety of industries and organization & cps allows us to imagine, create, develop refine & smart system in fields that result in the betterment of industries communities and individuals.

The ability to design and build successfully cyber physical systems will address many national priorities in ways that traditional computer science cannot increase as diverse as aerospace, automotive, energy, disaster, response, healthcare.

Expt. No.: _____ Page. No.: 7

Expt. Name: _____ Date: _____

are just a few ways in which the benefiting of cps technology have impacted different industries.

## Smart city management:

Tech target defines a Smart city as a municipality that uses information and communication technologies to increase operational efficiency share information with the public and improve both the quality of government services and citizen welfare.

## Infrastructure:

In 2021. The united states infrastructure received a e-rating from the american society of civil enginers' Report card for americans infrastructure.
cps engineers must master acting edge technologies in order to upgrade existing city infrastructure system.

## Automotive:-

IOT and cps technology have made specific advantages in Smart car technologies that can make vechile transportation states: "blind-spot monitoring lane-departure warning and forward collision warning" are just these features that if implemented in all cars in united states.

**Result:-** Studied about Industrial revolution & cyber physical system opportunities.

## 3. Create a program that blinks the LED on the development board using TinkerCAD.

## Objective:

To blink an LED using Arduino UNO by controlling GPIO pin output levels through a simple Arduino sketch.

## Components Required:

| S. No. | Component | Quantity |
|---|---|---|
| 1 | Arduino UNO | 1 |
| 2 | USB Cable (Type-B) | 1 |
| 3 | LED (any color) | 1 |
| 4 | 220Ω Resistor | 1 |
| 5 | Breadboard | 1 |
| 6 | Jumper Wires | 2 |
| 7 | Push Button | 1 |

## Principle:

An LED is controlled by supplying HIGH (5V) or LOW (0V) logic to its connected pin. When a HIGH signal is sent from the Arduino to the pin connected to the LED (through a resistor), the LED lights up. Delaying the signal toggling creates a blinking effect.

## Circuit Connections (Method 1 – LED ON when GPIO is HIGH):

| Component | Connection Details |
|---|---|
| LED Anode (Long leg) | Connected to Arduino Pin D8 via 220Ω resistor |
| LED Cathode (Short leg) | Connected to GND on Arduino |
| USB Cable | Connect Arduino to PC for power and programming |

## 3.1 Blinks the LED on the development board

## What is an LED?

- **LED** stands for **Light Emitting Diode**.
- Emits light when current flows in forward direction.
- Made from semiconductors like gallium arsenide or gallium phosphide.
- Has **two terminals**:
    - **Anode (+):** Long leg – connected to positive voltage.
    - **Cathode (−):** Short leg – connected to GND.

- Requires a **current-limiting resistor** to avoid damage.
- Typical operating current: **5–20 mA**.

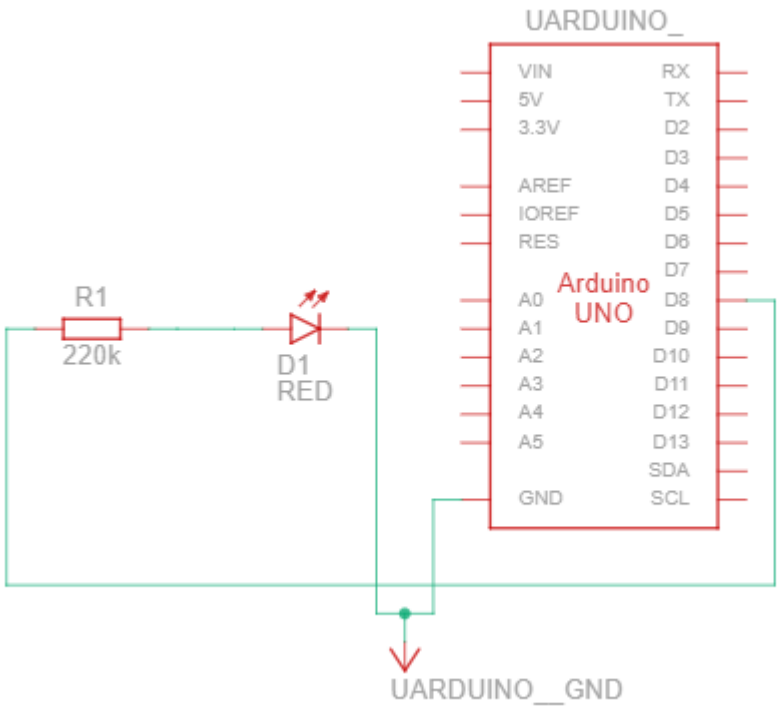
## What is a Resistor?

- Limits the flow of electric current.
- Symbol: **R**
- Unit: **Ohm (Ω)**

$$R = \frac{V}{I} = \frac{5V - 2V}{20mA} = \frac{3\ V}{0.02\ A} = 150\Omega$$

- In this experiment, a **150Ω-220Ω resistor** is used to protect the LED.

## Circuit Diagram:




## Code:

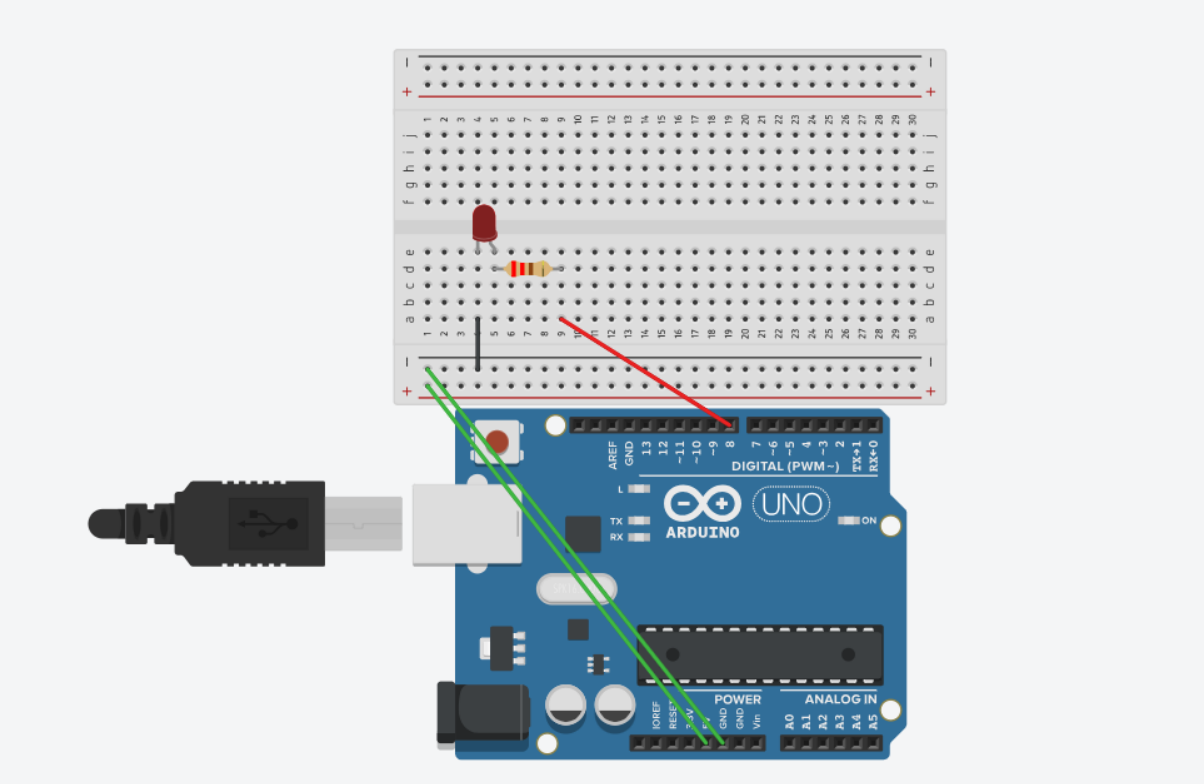
int ledPin=8; //definition digital 8 pins as pin to control the LED

void setup()

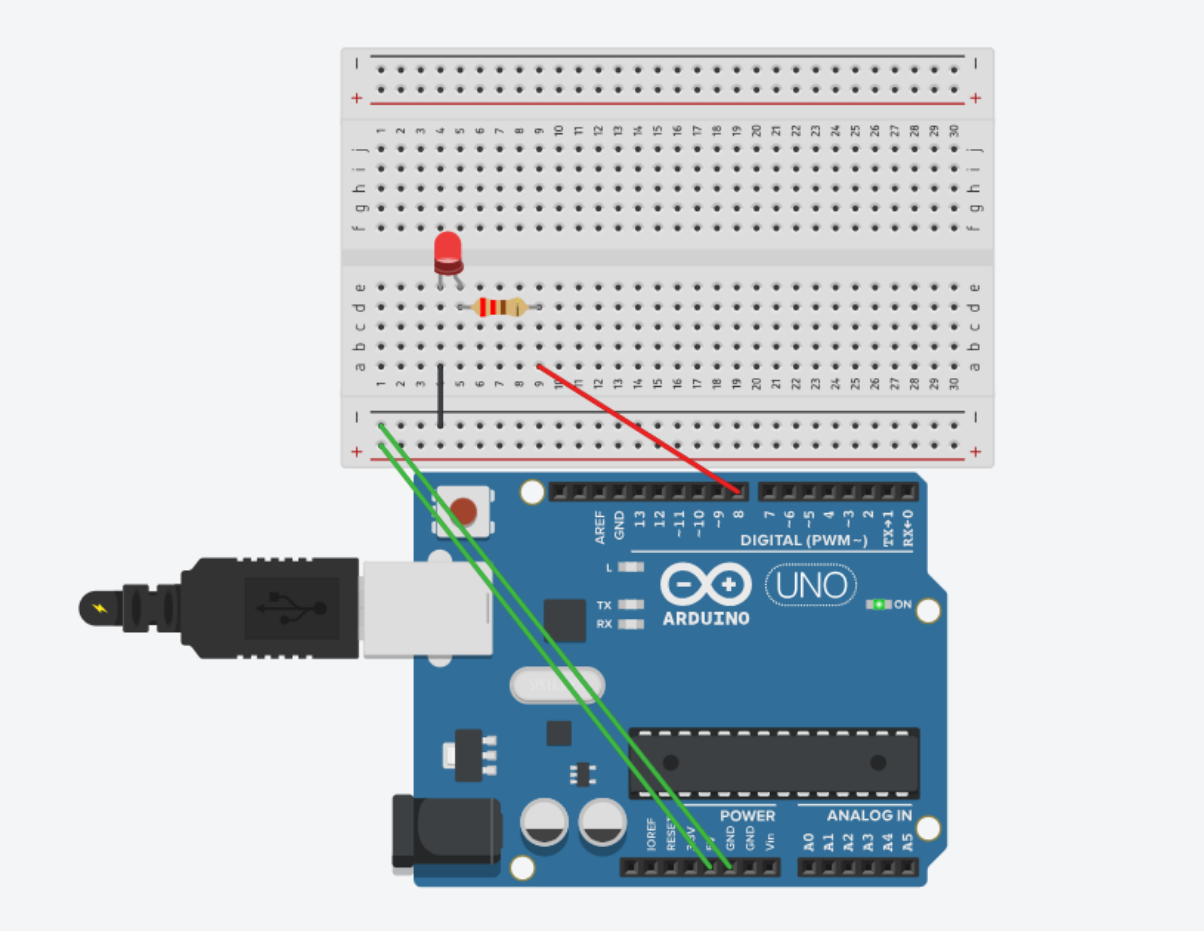
```
{
   pinMode(ledPin,OUTPUT);    //Set the digital 8 port mode, OUTPUT: Output mode
}
void loop()
{
   digitalWrite(ledPin,HIGH); //HIGH is set to about 5V PIN8
   delay(1000);               //Set the delay time, 1000 = 1S
   digitalWrite(ledPin,LOW);  //LOW is set to about 5V PIN8
   delay(1000);               //Set the delay time, 1000 = 1S
}
```

**Circuit Working:**

- When **D8 = LOW**, 0V is applied → LED turns **OFF**

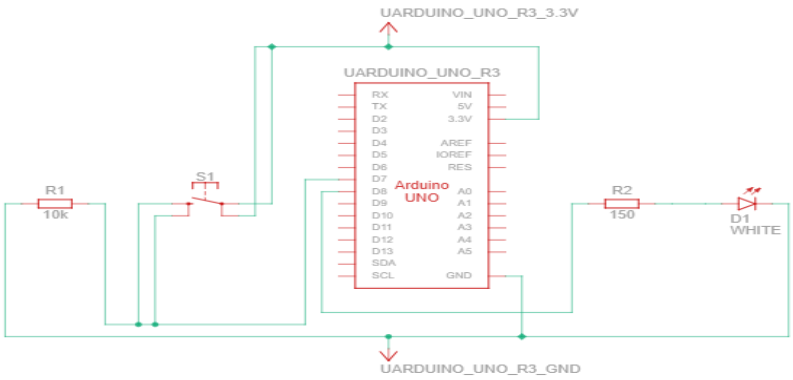



- When **D8 = HIGH**, 5V is applied → LED turns **ON**




### 3.2 LED Blink through push button

## Circuit Diagram:

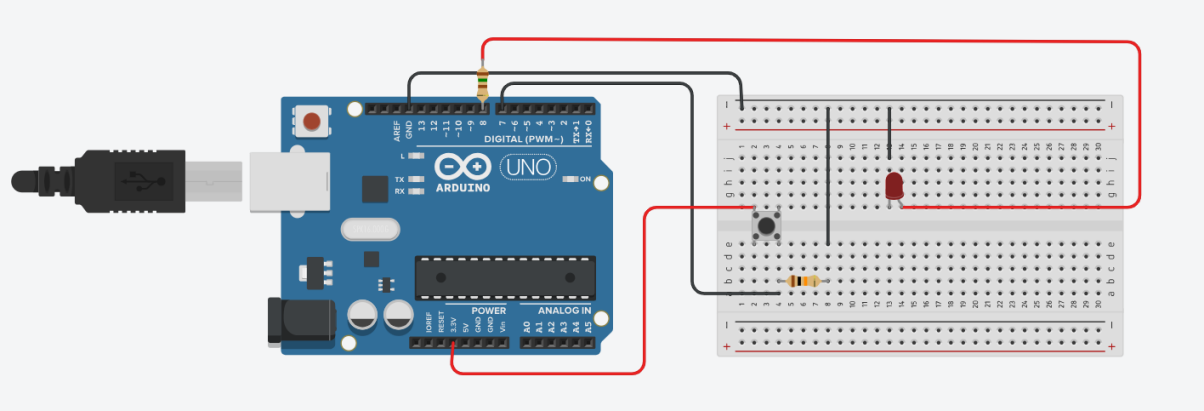



## Code:

unsigned const LED = 8;

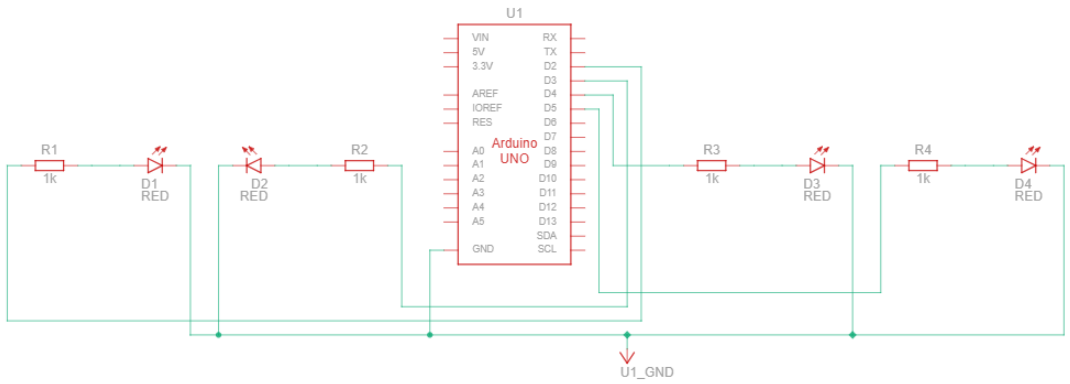
```
unsigned const BUTTON = 7;
unsigned int bState = 0;

void setup()
{
  pinMode(BUTTON, INPUT);
  pinMode(LED, OUTPUT);
}

void loop()
{
  if(digitalRead(BUTTON) == 1) {
          digitalWrite(LED, HIGH);
    bState += 1;
    if(bState % 2 == 0){
      digitalWrite(LED, LOW);
      bState = 0;
    }
          delay(100);
  }

}
```

## Circuit Working:



## 3.3 LED with digital counter

## Circuit Diagram:




## Code:

```
int pin2=2;

int pin3=3;

int pin4=4;

int pin5=5;

int stime=500;


void setup()

{
```

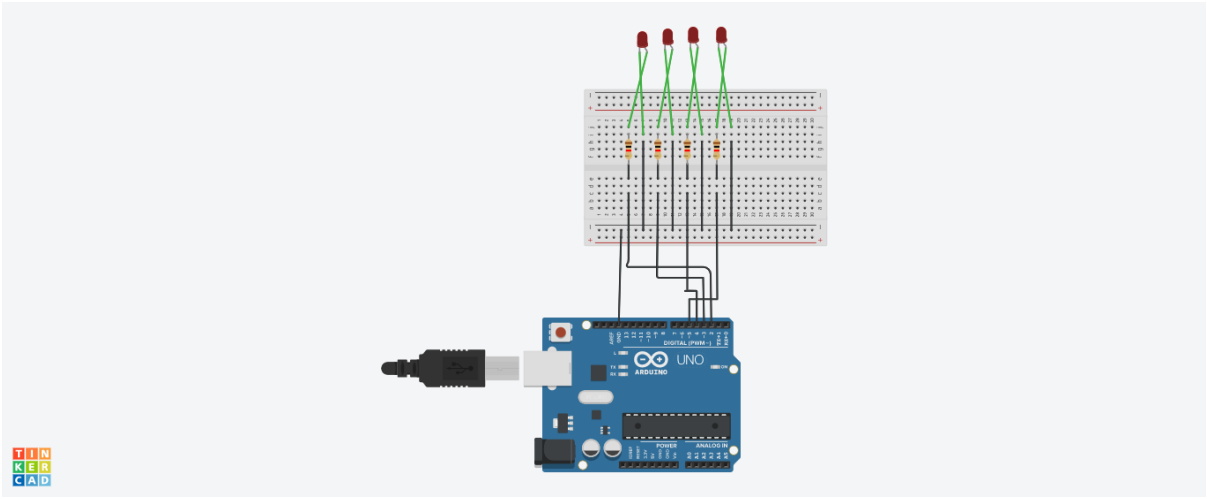
```
  pinMode(pin2,OUTPUT);
  pinMode(pin3,OUTPUT);
  pinMode(pin4,OUTPUT);
  pinMode(pin5,OUTPUT);
}

void loop()
{


  digitalWrite(pin2,LOW);
  digitalWrite(pin3,LOW);
  digitalWrite(pin4,LOW);
  digitalWrite(pin5,LOW);
  delay(stime);

  digitalWrite(pin2,LOW);
  digitalWrite(pin3,LOW);
  digitalWrite(pin4,LOW);
  digitalWrite(pin5,HIGH);
  delay(stime);

  digitalWrite(pin2,LOW);
  digitalWrite(pin3,LOW);
  digitalWrite(pin4,HIGH);
  digitalWrite(pin5,LOW);
  delay(stime);

  digitalWrite(pin2,LOW);
```

```
digitalWrite(pin3,LOW);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,HIGH);
delay(stime);

digitalWrite(pin2,LOW);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,LOW);
digitalWrite(pin5,LOW);
delay(stime);

digitalWrite(pin2,LOW);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,LOW);
digitalWrite(pin5,HIGH);
delay(stime);


digitalWrite(pin2,LOW);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,LOW);
delay(stime);

digitalWrite(pin2,LOW);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,HIGH);
delay(stime);
```

```
digitalWrite(pin2,HIGH);
digitalWrite(pin3,LOW);
digitalWrite(pin4,LOW);
digitalWrite(pin5,LOW);
delay(stime);
digitalWrite(pin2,HIGH);
digitalWrite(pin3,LOW);
digitalWrite(pin4,LOW);
digitalWrite(pin5,HIGH);
delay(stime);

digitalWrite(pin2,HIGH);
digitalWrite(pin3,LOW);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,LOW);
delay(stime);

digitalWrite(pin2,HIGH);
digitalWrite(pin3,LOW);
digitalWrite(pin4,HIGH);
digitalWrite(pin5,HIGH);
delay(stime);

digitalWrite(pin2,HIGH);
digitalWrite(pin3,HIGH);
digitalWrite(pin4,LOW);
digitalWrite(pin5,LOW);
delay(stime);
```

```
  digitalWrite(pin2,HIGH);
  digitalWrite(pin3,HIGH);
  digitalWrite(pin4,LOW);
  digitalWrite(pin5,HIGH);
  delay(stime);

  digitalWrite(pin2,HIGH);
  digitalWrite(pin3,HIGH);
  digitalWrite(pin4,HIGH);
  digitalWrite(pin5,LOW);
  delay(stime);

  digitalWrite(pin2,HIGH);
  digitalWrite(pin3,HIGH);
  digitalWrite(pin4,HIGH);
  digitalWrite(pin5,HIGH);
  delay(stime);
}
```

## Circuit Working:



## Results:

The LED was successfully controlled in three setups using Arduino UNO, blinking at 1-second intervals, toggling with a push button, and operating with a digital counter. These results confirm accurate GPIO pin control and basic Arduino programming using timing, input detection, and sequential logic.

## 4. Pick one-one from the available sensors and actuators and find or create code that will display the sensed data on the PC.

## Aim:

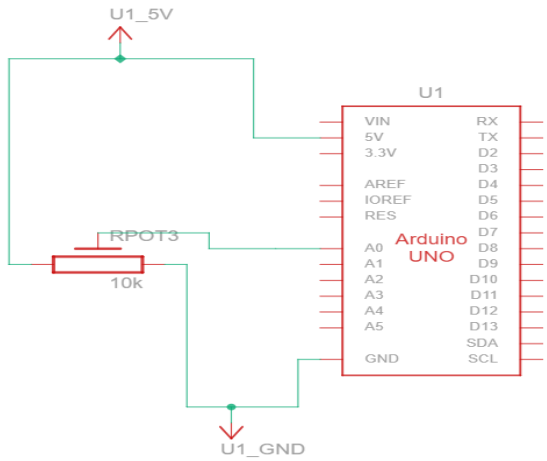
To interface a temperature sensor and an LED using Arduino, and display the sensed temperature data on the PC through the Serial Monitor.

## Components Required:

- **Arduino UNO board**
- **LM35 Temperature Sensor** (Sensor)
- **LED** (Actuator)
- **220Ω Resistor** (for LED)
- **Breadboard**
- **Jumper wires**
- **USB cable** (for connecting Arduino to PC)

## 4.1 Reading Analog Input from Potentiometer using Arduino

## Circuit Diagram:

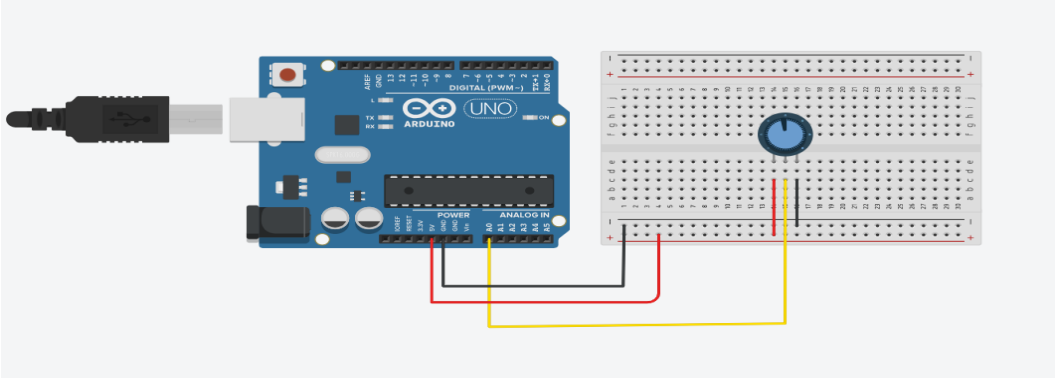



## Code:

```
int pot = A0;


void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  int potvalue = analogRead(pot);
  Serial.print("potvalue: ");  // Print label
  Serial.println(potvalue);    // Print actual value
  delay(10);
}
```

## Circuit Working:




## Results:

Potvalue: 716

Potvalue: 716

Potvalue: 716

Potvalue: 716

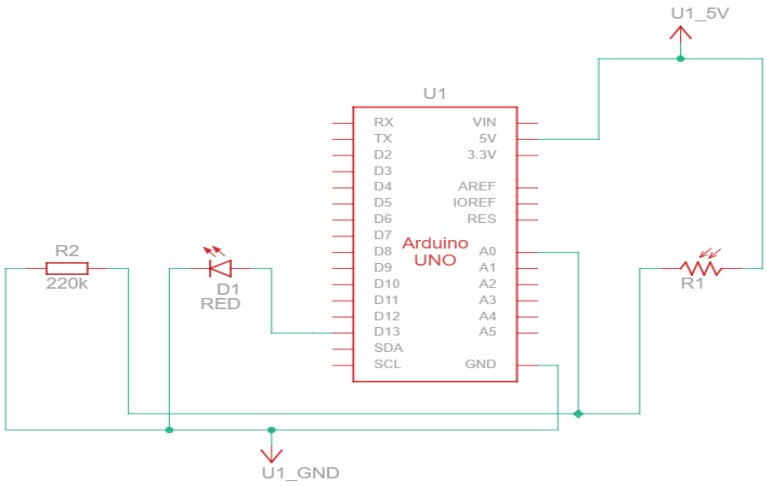
Potvalue: 696

Potvalue: 696

Potvalue: 696

## 4.2 Reading LDR sensor values using Arduino

## Circuit Diagram:

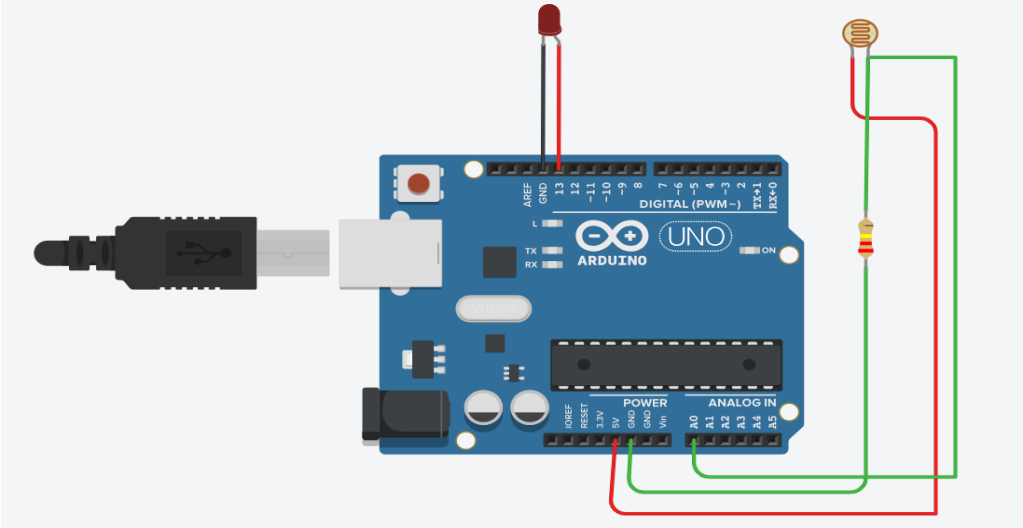



## Code:

```
const int LEDpin = 13;

const int LDRpin = A0;

void setup()

{

  Serial.begin(9600);

  pinMode(LEDpin, OUTPUT);

  pinMode(LDRpin, INPUT);


}

void loop()

{

  int LDRstatus = analogRead(LDRpin);

  if (LDRstatus <= 500)

  {

    digitalWrite (LEDpin, HIGH);

    Serial.print("Current Light Intensity Value is");

    Serial.println(LDRstatus);
```

```
  }
 else
 {
   digitalWrite(LEDpin, LOW);
   Serial.print("Current light Intensity Value is ");
   Serial.println(LDRstatus);
 }
}
```

## Circuit Working:




## Output:

Current Light Intensity Value is 563

Current Light Intensity Value is 563

Current Light Intensity Value is 563

## Results :

The Arduino UNO successfully read varying analog inputs from both an LDR sensor and a potentiometer.

## 5. Create a program that displays data from the sensor in regular intervals in a compact format.

### Aim:

The goal of this experiment is to create a program that displays data from a sensor in regular intervals, showing the data in a compact format.

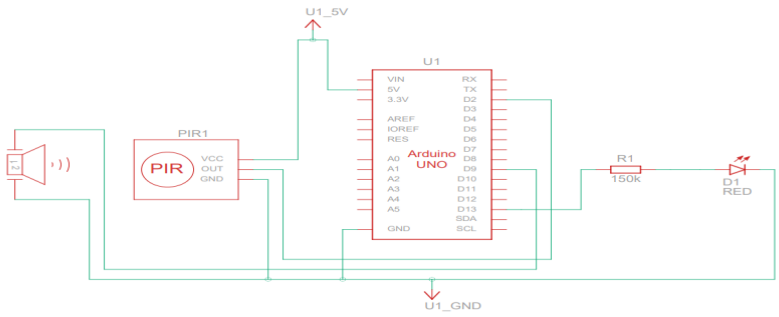
### Components Required:

- PIR Sensor
- LED
- Resistor
- Piezo buzzer

### Steps:

- Log into TinkerCAD and create a new circuit.
- Use the required components (PIR sensor, LED, Piezo) to build the circuit.
- Implement the necessary code to display sensor data at regular intervals.
- Use TinkerCAD’s "start simulation" feature to verify the working of the program.

### Circuit Diagram:

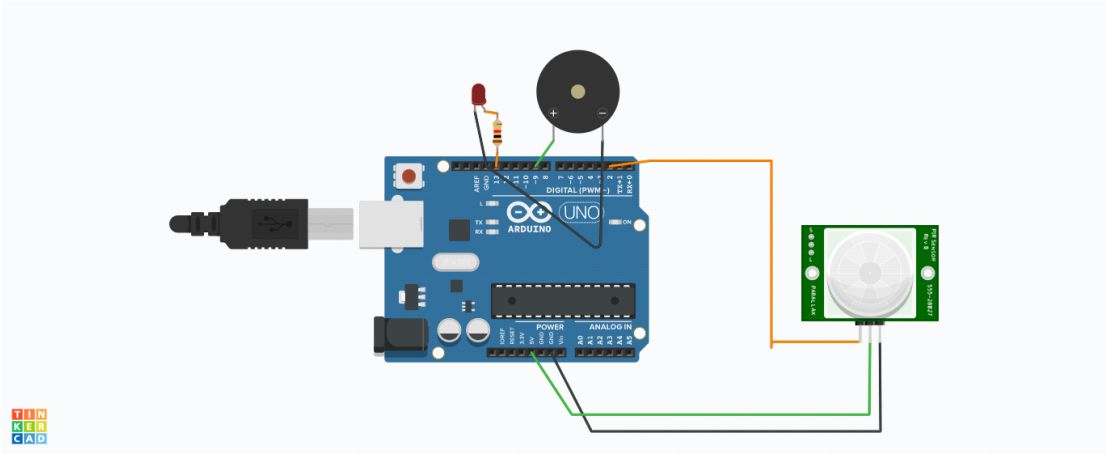



### Code:

```
void setup()
{
```

```
  pinMode(2, INPUT);
  pinMode(13, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  if (digitalRead(2) >= HIGH) {
    digitalWrite(13, HIGH);
    tone(9, 523, 1000); // play tone 60 (C5 = 523 Hz)
  } else {
    digitalWrite(13, LOW);
    noTone(9);
  }
  delay(1); // Wait for 1 millisecond(s)
}
```

## Circuit Working:



## Results:

The simulation successfully displays data from the sensor at regular intervals, confirming that the data is being read and displayed correctly.

## 5. To design a Basic weather station using embedded components using TinkerCAD.

## Objective:

To design and implement a **basic weather station** using TinkerCAD. The system will:

- **Monitor environmental parameters** like temperature and light intensity.
- **Alert users** using LEDs and a piezo buzzer when predefined thresholds are crossed.
- **Display sensor values** on an LCD.

## Components Required:

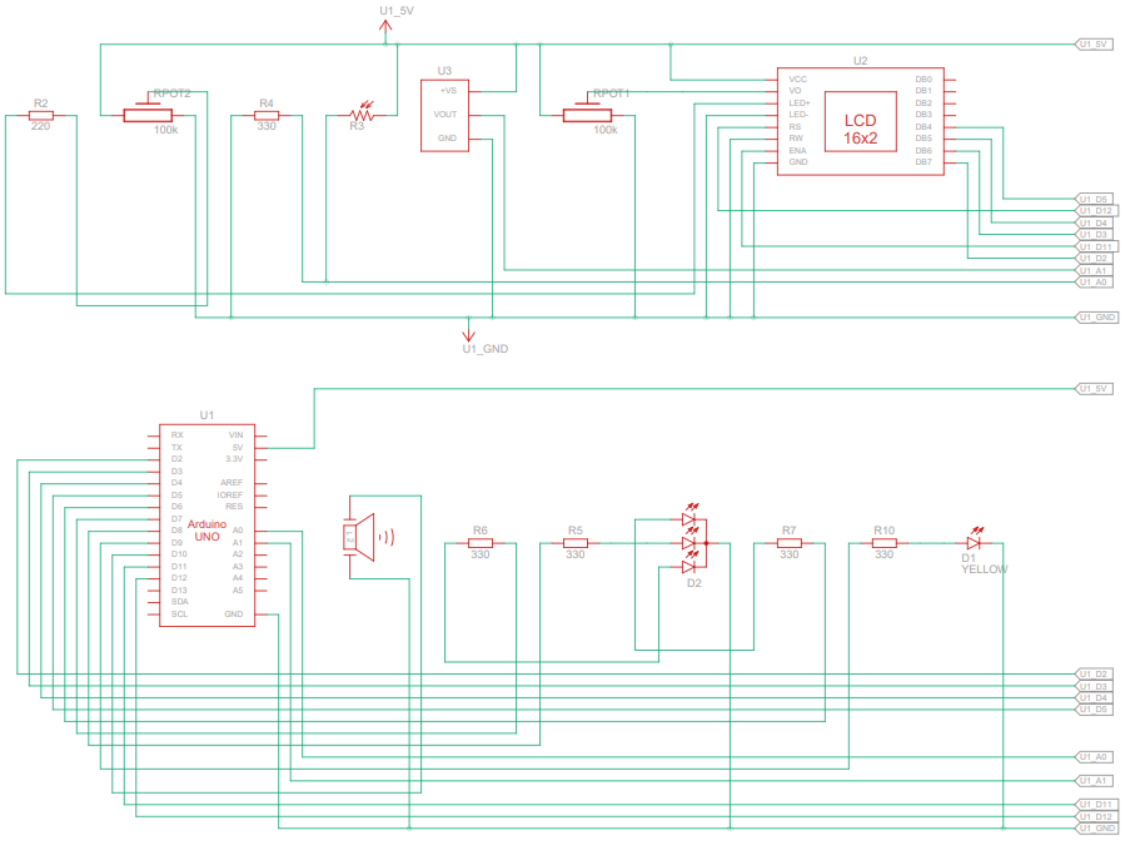
| S.No. | Component Name | Quantity | Purpose |
|---|---|---|---|
| 1 | Arduino UNO R3 | 1 | Main controller to read sensor values & control output |
| 2 | Breadboard | 2–3 | For assembling components and making connections |
| 3 | USB Cable (Type-B) | 1 | Power and serial communication with PC |
| 4 | LDR (Light Dependent Resistor) | 1 | Senses ambient light levels |
| 5 | Resistor (10kΩ for LDR) | 1 | Forms voltage divider with LDR |
| 6 | Temperature Sensor (e.g., TMP36 or LM35) | 1 | Senses ambient temperature |
| 7 | Potentiometer (10kΩ) | 2 | One for LCD contrast; one optional for threshold input |
| 8 | Piezo Buzzer | 1 | Provides sound alert on condition trigger |
| 9 | LED (any color) | 2 | Visual alert or status indicator |
| 10 | Resistors (220Ω) | 2 | Current limiting resistors for LEDs |
| 11 | 16x2 LCD Display (with or without I2C) | 1 | Displays sensor data (temperature, light) |
| 12 | Jumper Wires (Male-to-Male) | 30+ | For connecting components on breadboard |

| S.No. | Component Name | Quantity | Purpose |
|---|---|---|---|
| 13 | Resistors (as needed) | 2–4 | For signal conditioning and safety |

## Principle:

The basic weather station operates on the principle of sensor-based environmental monitoring using the Arduino microcontroller. It utilizes analog sensors like an LDR (Light Dependent Resistor) to detect ambient light levels and a temperature sensor (such as TMP36 or LM35) to measure the surrounding temperature. These sensors convert physical quantities light intensity and heat, into analog electrical signals, which are read by the Arduino through its analog input pins. The Arduino processes this data and displays the real-time values on a 16x2 LCD screen. When the sensor readings exceed predefined threshold values, the Arduino triggers visual (LED) and audio (piezo buzzer) alerts to notify the user. This setup demonstrates how embedded systems can integrate sensor inputs and output devices to build a compact, real-time environmental monitoring solution.

## Circuit Diagram:



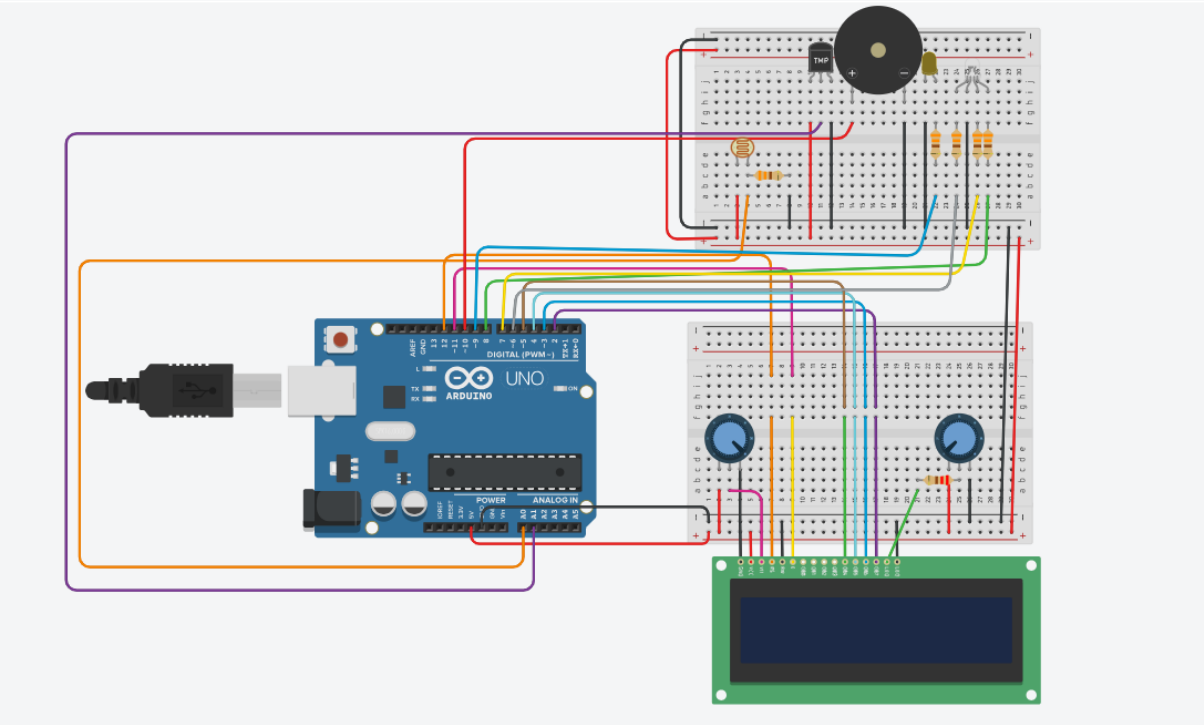
## Code:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int wLED = 9;
int lightLevel = 0;
int LDR = A0;
int TMP = A1;
int rLED = 6;
int bLED = 7;
int gLED = 8;
int sensorPin = A1;

void setup() {
  Serial.begin(9600);
  lcd.begin (16, 2);
  pinMode(wLED, OUTPUT);
  pinMode(rLED, OUTPUT);
  pinMode(bLED, OUTPUT);
  pinMode(gLED, OUTPUT);
}

void loop() {

  int reading = analogRead(sensorPin);

  // converting that reading to voltage, for 3.3v arduino use 3.3
  float voltage = reading * 5.0;
```

```
  voltage /= 1024.0;

  // print out the voltage
  Serial.print(voltage); Serial.println(" volts");

  // now print out the temperature
  float temperatureC = (voltage - 0.5) * 100 ;  //converting from 10 mv per degree wit 500 mV
offset
                                                //to degrees ((voltage - 500mV) times 100)
  Serial.print(temperatureC); Serial.println(" degrees C");

  // now convert to Fahrenheit
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
  Serial.print(temperatureF); Serial.println(" degrees F");

  lightLevel = analogRead(LDR);
  Serial.println(lightLevel);

  if (lightLevel < 10) {
    digitalWrite(wLED, HIGH);
  } else {
    digitalWrite(wLED, LOW);
  }

    lcd.clear();
  lcd.setCursor(0, 0);

  if (temperatureC < 2) {
    digitalWrite(bLED, HIGH);
```

```
    digitalWrite(rLED, LOW);
    lcd.print("Cold Weather");
    lcd.setCursor(0, 1);
    lcd.print(temperatureC);
    tone(10, 260);
  } else if (temperatureC >= 2 && temperatureC < 45){
    digitalWrite(bLED, LOW);
    digitalWrite(rLED, LOW);
    lcd.setCursor(0, 0);
    lcd.print("Normal Weather");
    noTone(10);
  } else {
    digitalWrite(rLED, HIGH);
    digitalWrite(bLED, LOW);
    lcd.setCursor(0, 0);
    lcd.print("Hot Weather");
    lcd.setCursor(0, 1);
    lcd.print(temperatureC);
    tone(10, 260);
  }

  delay (500);

}
```

## Circuit Working:




## Results:

The weather station successfully monitored temperature and light intensity in real-time using embedded sensors and displayed the data on a 16x2 LCD. When the sensor readings exceeded predefined thresholds, it effectively triggered visual (LED) and audio (buzzer) alerts, demonstrating reliable environmental monitoring and alert functionality.

## 7. Login to TinkerCAD and create a project then create a virtual device. Add the corresponding sensor and actuator to the virtual device.

### Aim:

The aim of this experiment is to create a virtual IoT device in Tinkercad, integrate a sensor and actuator, and simulate their interaction within a cloud-based platform.

### Components Required:

| S.No | Components | Description |
|---|---|---|
| 1 | Arduino UNO | Microcontroller to control the circuit. |
| 2 | Servo Motor | Actuator to perform mechanical movements. |
| 3 | Temperature Sensor | Sensor to measure temperature (e.g., TMP36). |
| 4 | Potentiometer | Input device to simulate sensor data. |
| 5 | Breadboard | Platform for building the circuit. |
| 6 | Jumper Wires | Wires for connecting components. |
| 7 | Resistor | Used for limiting current and protecting components. |

### Steps:

1. Add the following components to the workspace:

- Arduino UNO
- Servo Motor
- Temperature Sensor (e.g., TMP36 or any available temperature sensor)
- Potentiometer (optional, to simulate sensor input)
- Breadboard
- Jumper Wires and Resistors (as needed)

2. Wire the components:

- Connect the servo motor's control pin to pin 9 on the Arduino.
- Connect the servo motor's power and ground pins to the breadboard's power and ground rails.
- Connect the temperature sensor's VCC and GND to the power and ground rails.
- Connect the temperature sensor's output pin to analog input pin A1 on the Arduino.
- Connect one side of the potentiometer to power and the other side to ground.
- Connect the middle pin of the potentiometer to analog input pin A0 on the Arduino.

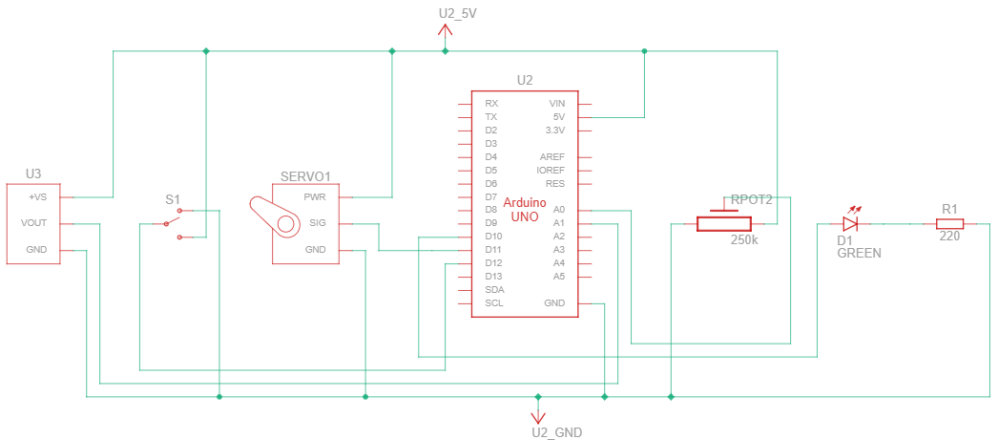
3. Go to the Code section in Tinkercad and select Text and write the code.

## Code:

```
#include <Servo.h>

int reading=0;
int duty;
int angle;

Servo servo_11;

void setup()
{
  pinMode(12, INPUT);
  pinMode(A0, INPUT);
  pinMode(10, OUTPUT);
  servo_11.attach(11);
  pinMode(A1, INPUT);
}

void loop()
{
  if (digitalRead(12) == 0)
  { reading = analogRead(A0);
    duty= map(reading,0,1023,0,225);
    analogWrite(10, duty);
    angle= map(reading,0,1023,0,180);
    servo_11.write(angle);
  }
```

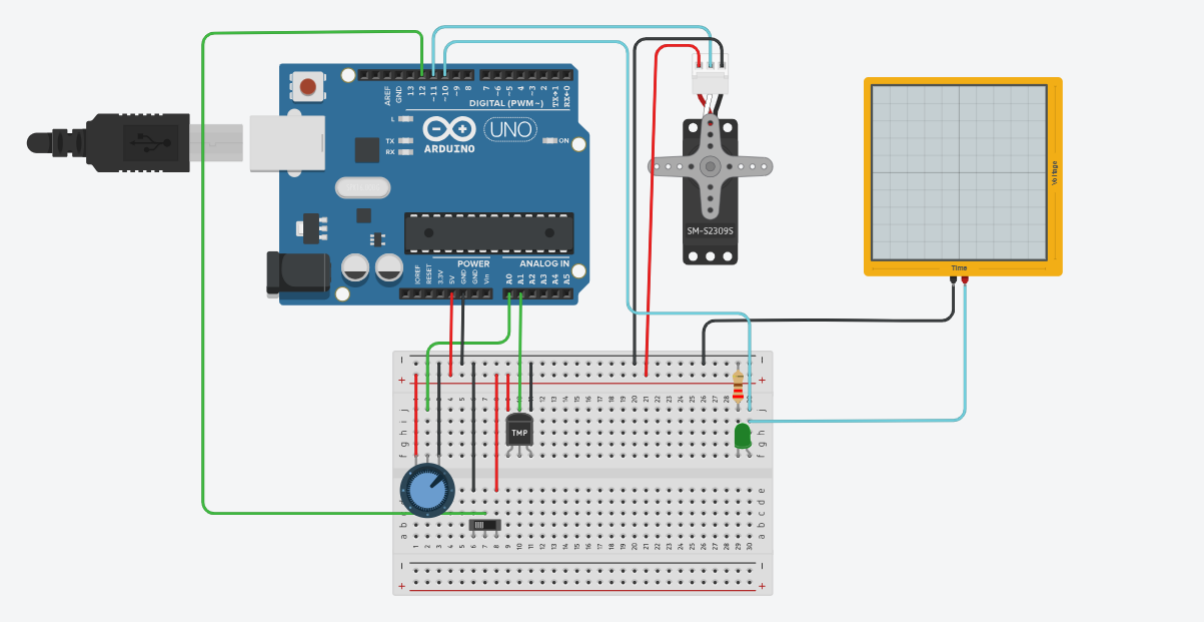
```
  else
  { reading = analogRead(A1);
    duty= map(reading,20,359,0,255);
    analogWrite(10, duty);
    angle= map(reading,20,359,0,180);
    servo_11.write(angle);
  }
  delay(100);
}
```

## Circuit Diagram:




## Circuit Working:

This setup demonstrates the creation and management of virtual IoT devices using Tinkercad. It simulates real-world applications such as adjusting a servo motor based on sensor data. The platform allows you to visualize how data from a potentiometer (as a sensor) is read and used to control the actuator (servo motor), providing insight into how IoT systems function by processing inputs and controlling outputs within a simulated environment.




## Result:

The virtual device was successfully created, with the sensor and actuator integrated. The system simulates real-world IoT functionality, allowing the interaction between sensor input (e.g., potentiometer or temperature sensor) and actuator output (e.g., servo motor).

## 8. Study the MQTT protocol. Examine the components of the protocol.

## Aim:

The aim of this experiment is to implement the MQTT protocol for communication in IoT systems, focusing on connecting devices to a central broker for efficient message exchange in a publish/subscribe model, to understand its lightweight and reliable communication in IoT applications.

## Introduction:

The MQTT (Message Queuing Telemetry Transport) protocol is a lightweight messaging protocol designed for efficient communication in Internet of Things (IoT) systems. It operates on a publish/subscribe model, where devices (publishers) send messages to specific topics, and other devices (subscribers) receive those messages by subscribing to the relevant topics. This protocol is ideal for environments where bandwidth is limited, and devices need to send small amounts of data with low power consumption. MQTT ensures reliable message delivery, supports different levels of Quality of Service (QoS), and allows for easy integration in IoT applications. The protocol's simplicity and efficiency make it a popular choice for real-time communication in IoT networks, providing an effective solution for connecting devices and exchanging data in various industrial and consumer applications.

## Characteristics of MQTT:

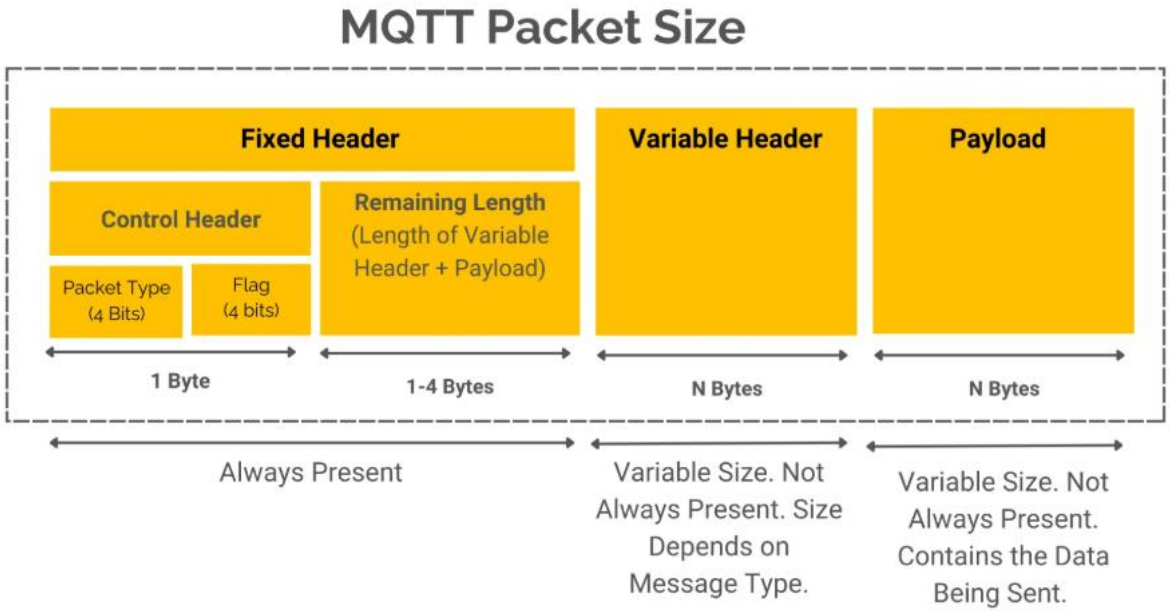
1. **Lightweight and Efficient**: MQTT has minimal overhead, making it ideal for constrained devices and low-bandwidth environments. It is designed to operate efficiently even with small data packets.
2. **Publish/Subscribe Model**: MQTT operates on a publish/subscribe communication model, where publishers send messages to topics, and subscribers receive messages from topics they are subscribed to. This decouples the sender and receiver, improving scalability.
3. **Quality of Service (QoS) Levels**: MQTT supports three QoS levels (0, 1, and 2) that determine the message delivery guarantee. This allows for flexibility in how message delivery is managed based on application needs.
4. **Retained Messages**: MQTT allows messages to be retained by the broker. When a new client subscribes to a topic, it receives the last retained message, ensuring that important information is always available.
5. **Last Will and Testament (LWT)**: MQTT provides the ability to define a Last Will and Testament message, which is sent by the broker if a client unexpectedly disconnects. This helps notify other clients about the disconnection, enabling better system reliability and monitoring.

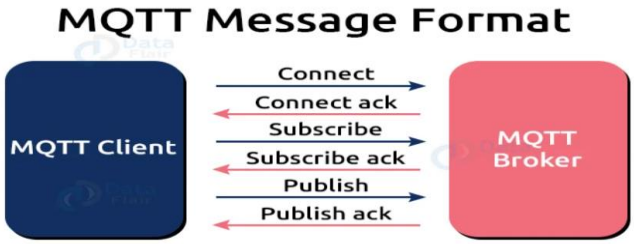
## Components of MQTT Protocol:

1. **Publisher**:
    a. The device or application that sends messages (data) to a topic. It is responsible for publishing data to the MQTT broker.
2. **Subscriber**:
    a. The device or application that subscribes to a specific topic to receive messages. It listens for updates published on that topic.
3. **Broker**:
    a. The central server that manages communication between publishers and subscribers. It handles message distribution, topic subscriptions, and ensures message delivery.
4. **Topic**:
    a. A string that represents the subject of the message. Publishers send messages to specific topics, and subscribers receive messages based on their subscriptions to those topics.
5. **Message**:
    a. The actual data sent by the publisher to the broker. It can contain various types of information, such as sensor readings, commands, or status updates.
6. **Quality of Service (QoS)**:
    a. A level of message delivery assurance. MQTT supports three QoS levels:
        i. **QoS 0**: At most once delivery (no acknowledgment).
        ii. **QoS 1**: At least once delivery (message acknowledgment).
        iii. **QoS 2**: Exactly once delivery (ensures no duplication).
7. **Client ID**:
    a. A unique identifier for each client (publisher or subscriber) that connects to the broker.
8. **Last Will and Testament (LWT)**:
    a. A message that is sent by the broker if a client unexpectedly disconnects. It allows subscribers to be notified of a client's disconnection.

## MQTT Message Format:

The MQTT message format is structured into several key components to ensure efficient and reliable communication between IoT devices. The Fixed Header contains essential information, such as the message type and flags that define the behavior of the message. It is mandatory for every MQTT message. The Variable Header varies depending on the message type and contains additional data such as the message identifier, topic name, or other protocol-specific parameters. The Payload is the actual content or data being transmitted, which can be any form of information, like sensor readings or command instructions. Lastly, the Optional section includes additional data that may not be required for every message, such as retained message flags or the last will and Testament (LWT) message. This structured format ensures that MQTT can deliver lightweight, low-overhead messaging while maintaining flexibility in different IoT scenarios.



## Results:

The MQTT protocol study successfully illustrates its lightweight and efficient structure, consisting of the fixed header, variable header, payload, and optional sections. This design ensures reliable, low-overhead messaging, making MQTT an ideal choice for real-time communication in IoT environments.

## 9. Create a connection from an MQTT capable device/software with an MQTT broker then send and receive data using it.

**Aim:**

To connect an MQTT client (using MQTT Explorer) to an MQTT broker (testclient-cloud.mqtt.cool), then send and receive data.

**Components Needed:**

- **MQTT Explorer** (a GUI tool to connect to MQTT brokers, send/receive messages, and monitor MQTT topics).
- **Internet Connection**.
- **Browser** (for connecting to the MQTT broker via testclient-cloud.mqtt.cool).

**Procedure:**

**Step 1: Set up MQTT Broker**

We will be using testclient-cloud.mqtt.cool as a public MQTT broker.

1. Open the browser and go to: https://testclient-cloud.mqtt.cool/
2. You will see a web interface to connect to the MQTT broker. No registration or login is needed since it's a public broker.

**Step 2: Connect MQTT Explorer to the Broker**

1. Download and Install MQTT Explorer (if you don't have it already):
    - Download it from MQTT Explorer's official website.
    - Install the tool on your system.
2. Open MQTT Explorer and click on File → New Connection.
3. Enter the Connection Details:
    - Broker Address: testclient-cloud.mqtt.cool
    - Port: 1883 (for non-SSL connections).
    - Client ID: (You can leave this as default or enter a custom name).
    - Username and Password: These are not required for this public broker.
    - Leave other settings as they are (or adjust if needed).
4. Click Connect to establish the connection to the MQTT broker.

**Step 3: Subscribe to a Topic in MQTT Explorer**

1. Once connected, you can see the Broker connection and the topics available in MQTT Explorer.
2. To subscribe to a topic:
    - Click on the Subscribe button at the top.
    - Enter a topic name (e.g., test/topic).
    - Click OK to subscribe.

**Step 4: Send Data (Publish Message)**

1. Publish a message to a topic:
    - In MQTT Explorer, go to the Publish section.
    - Enter the topic name (e.g., test/topic).
    - Enter the message (e.g., Hello from MQTT Explorer!).
    - Click Publish to send the message.
2. The subscriber (MQTT Explorer window listening to the topic test/topic) will automatically display the message you sent.

**Step 5: Monitor and Verify Data Exchange**

1. **Verify the Data Exchange**:
    - After publishing the message from MQTT Explorer, the message should appear in the Subscriber section under test/topic.
    - You should see the message "Hello from MQTT Explorer!" appear in the window of MQTT Explorer.

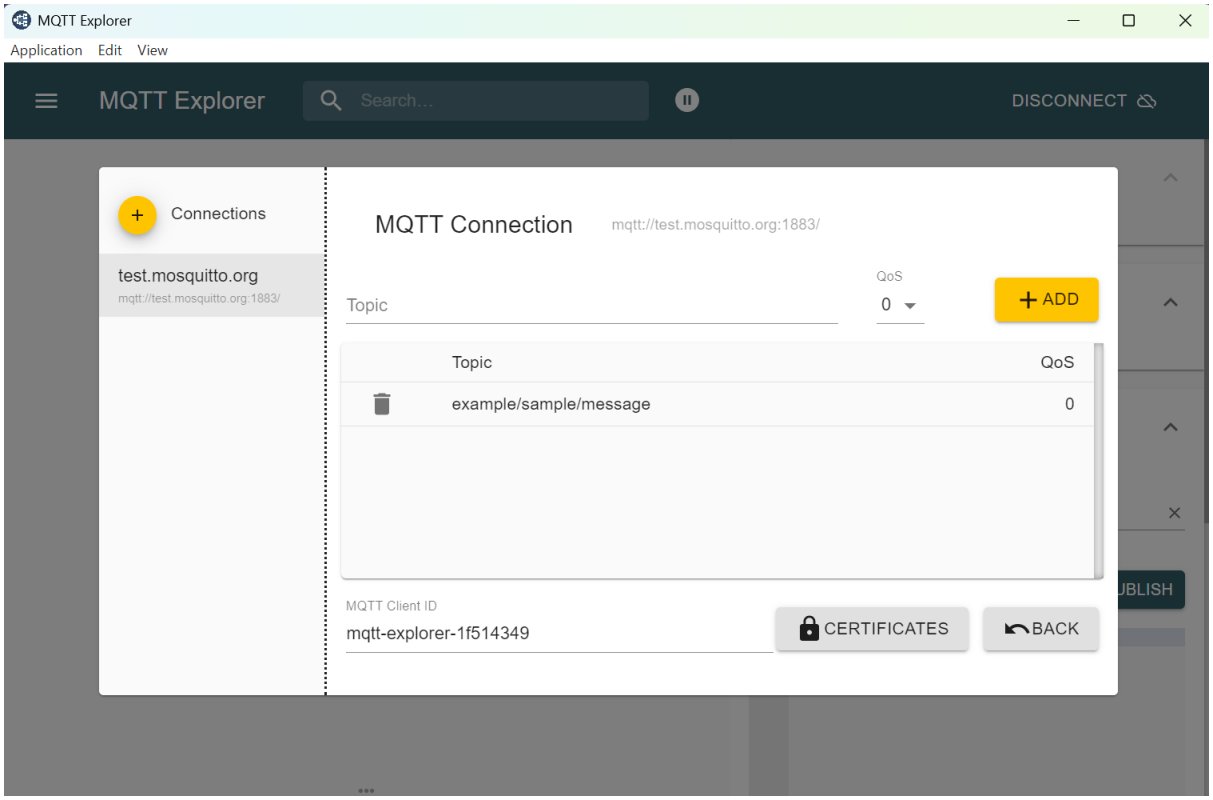
**Step 6: Send and Receive Data from the TestClient**

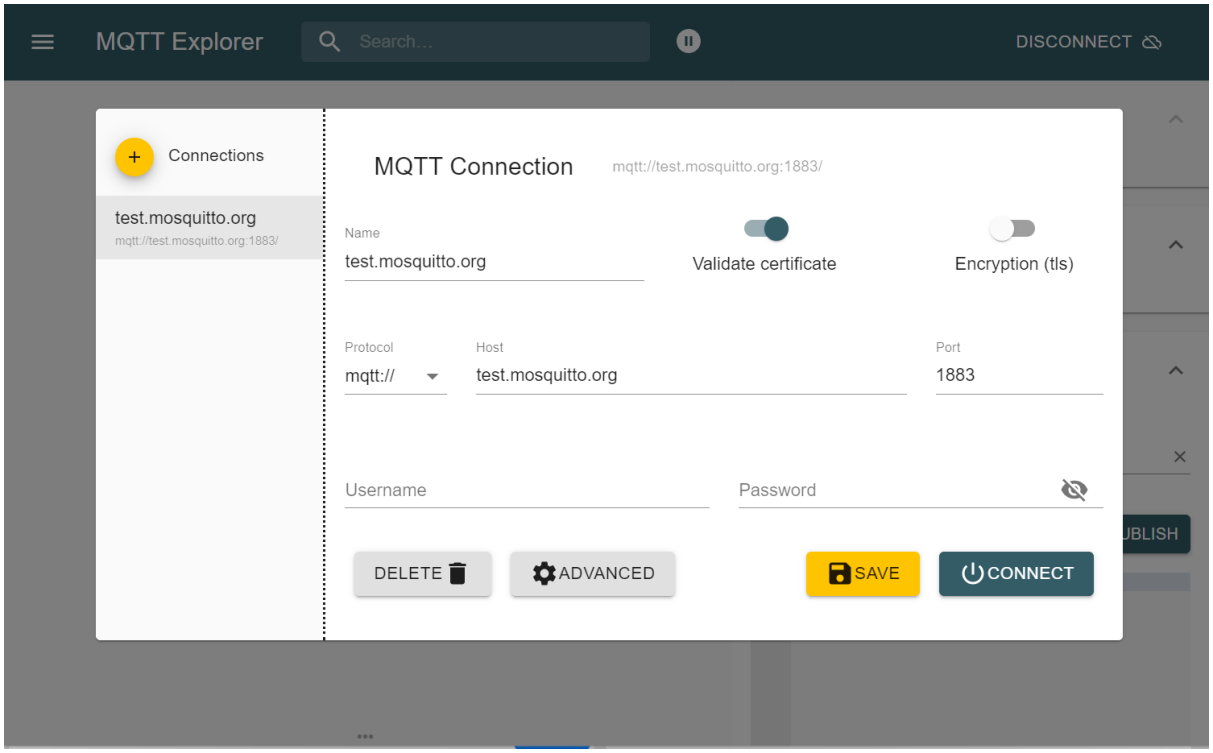
1. **In TestClient (Browser interface)**:
    - Go to https://testclient-cloud.mqtt.cool/.
    - Enter the same broker details (e.g., testclient-cloud.mqtt.cool and port 1883).
    - Connect to the broker.
2. **Subscribe to a Topic**:
    - Enter the topic test/topic in the Subscription section.
    - Press Subscribe.
3. **Publish Message from TestClient**:

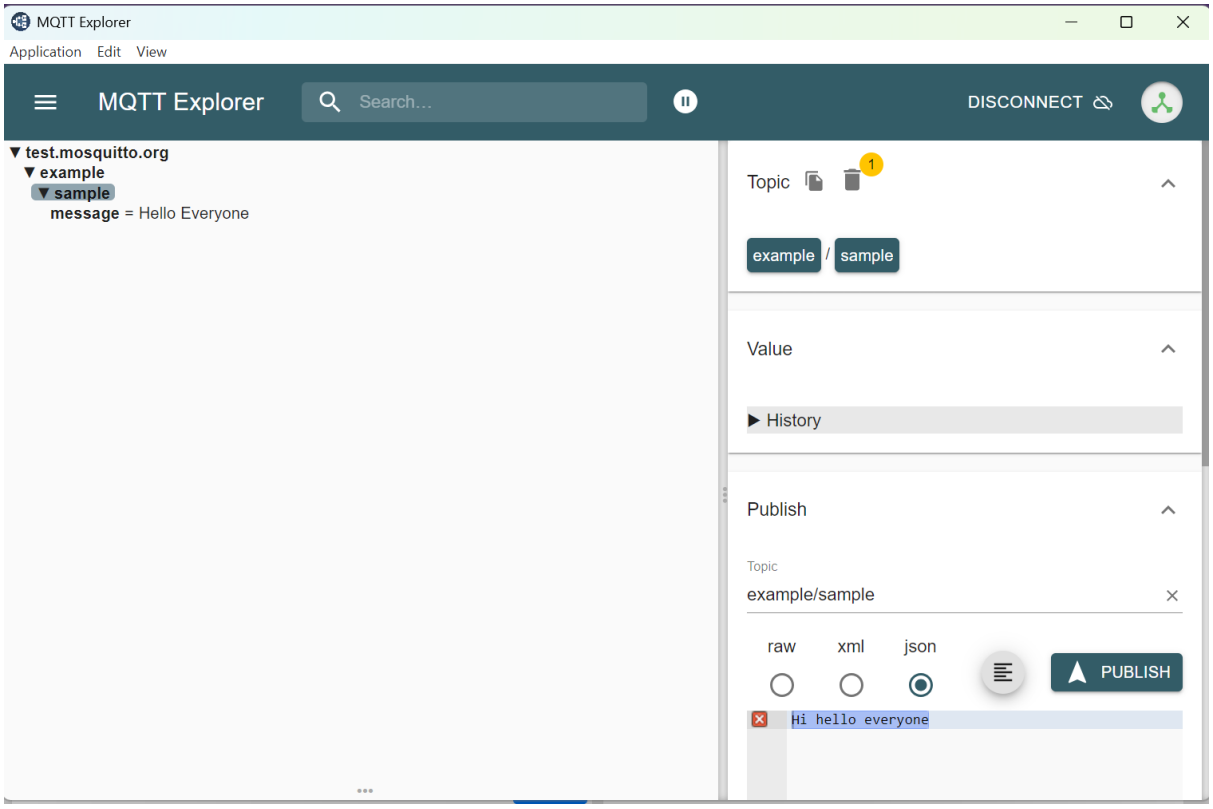
- To send a message, enter the topic test/topic and message "Hello from TestClient!".
- Click Publish.

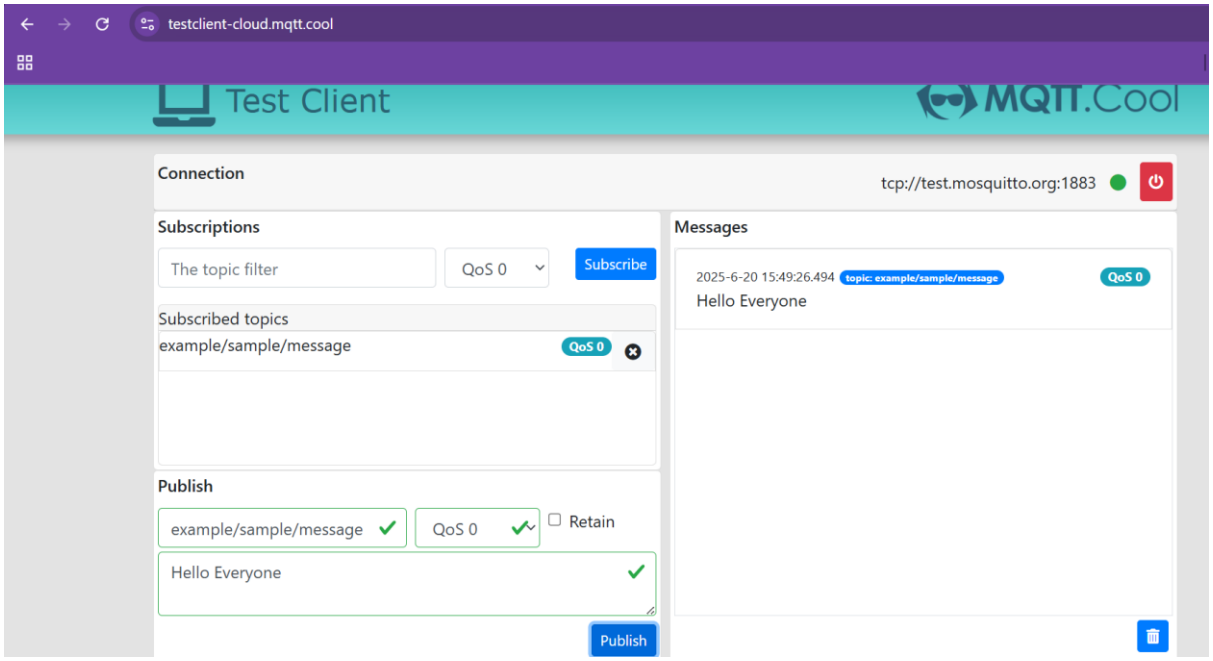
4. **Verify Message in MQTT Explorer**:
   - After publishing from the browser interface (TestClient), you should see the message "Hello from TestClient!" in the MQTT Explorer's Subscriber section under test/topic.

**Output:**




MQTT Explorer
Search...
DISCONNECT
Connections
test.mosquitto.org
mqtt://test.mosquitto.org:1883/
MQTT Connection
mqtt://test.mosquitto.org:1883/
Name
test.mosquitto.org
Validate certificate
Encryption (tls)
Protocol
mqtt://
Host
test.mosquitto.org
Port
1883
Username
Password
DELETE
ADVANCED
SAVE
CONNECT


MQTT Explorer
Application Edit View
MQTT Explorer
Search...
DISCONNECT
test.mosquitto.org
example
sample
message = Hello Everyone
Topic
1
example / sample
Value
History
Publish
Topic
example/sample
raw
xml
json
PUBLISH
Hi hello everyone




**Results:**

The experiment successfully connected **MQTT Explorer** to the **MQTT broker**, enabling successful message exchange between **MQTT Explorer** and **TestClient**. Real-time data monitoring and communication through the broker were verified.

## 10. To configuring the gateways and exchange the data to local database using LABVIEW.

### Aim:

The aim of the experiment is to configure LabVIEW to connect with a local database, query data using SQL commands, and display the filtered results graphically in a waveform graph. The data is filtered using a condition (e.g., WHERE randomvalue < 0.5), allowing for the visualization of trends, outliers, and anomalies. This demonstrates the integration of LabVIEW with databases for data visualization and analysis.

### Components Required:

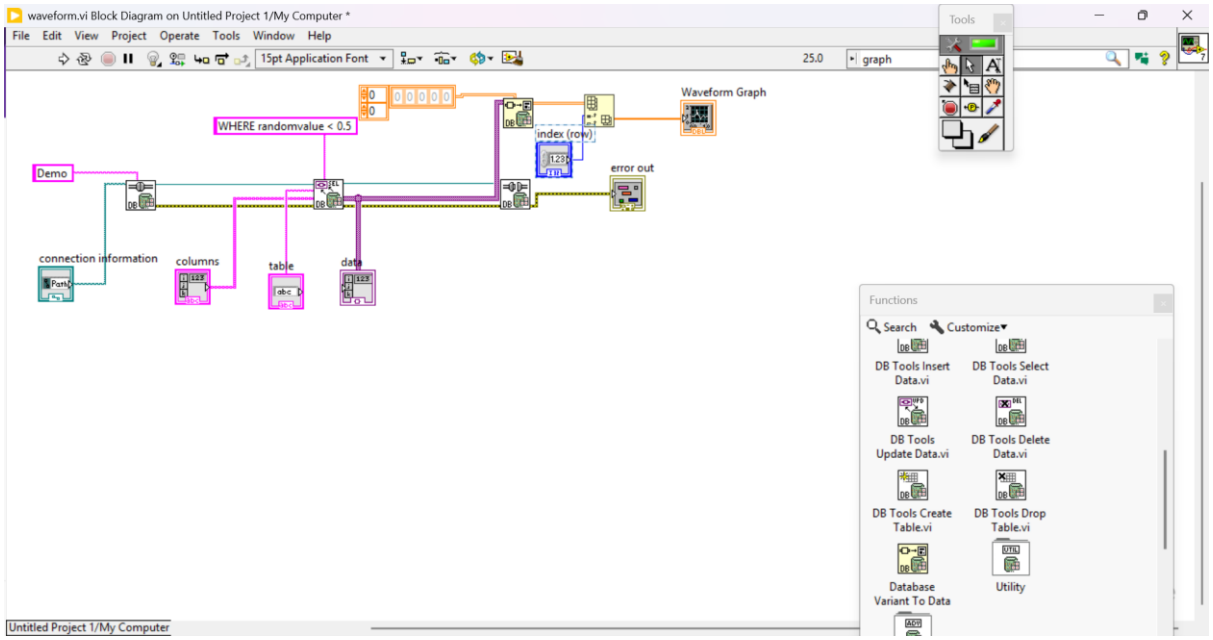
| Component | Description |
|---|---|
| **DB Tools Open Connection.vi** | Opens a connection to the database using the provided credentials and path. |
| **DB Tools Select Data.vi** | Executes a query to retrieve data from the database. |
| **DB Tools Insert Data.vi** | Used for inserting data into the database (though not shown in the diagram). |
| **DB Tools Create Table.vi** | Creates a table in the database (potentially used for initial setup). |
| **DB Tools Update Data.vi** | Updates existing data in the database (not explicitly shown but useful). |
| **DB Tools Delete Data.vi** | Deletes data from the database (useful for data cleanup). |
| **DB Tools Drop Table.vi** | Drops a table from the database (potentially for cleanup purposes). |
| **DB Tools Error Out.vi** | Captures and handles any errors that occur during the database operations. |
| **Waveform Graph** | Displays the data retrieved from the database as a graphical waveform. |

| Component | Description |
|---|---|
| **Random Value Generator** | Generates random values that are used in the database for filtering (randomvalue). |
| **SQL Query String** | Used to form the SQL query string for data retrieval with a specific condition. |
| **Path Control** | Specifies the path or location of the local database file. |

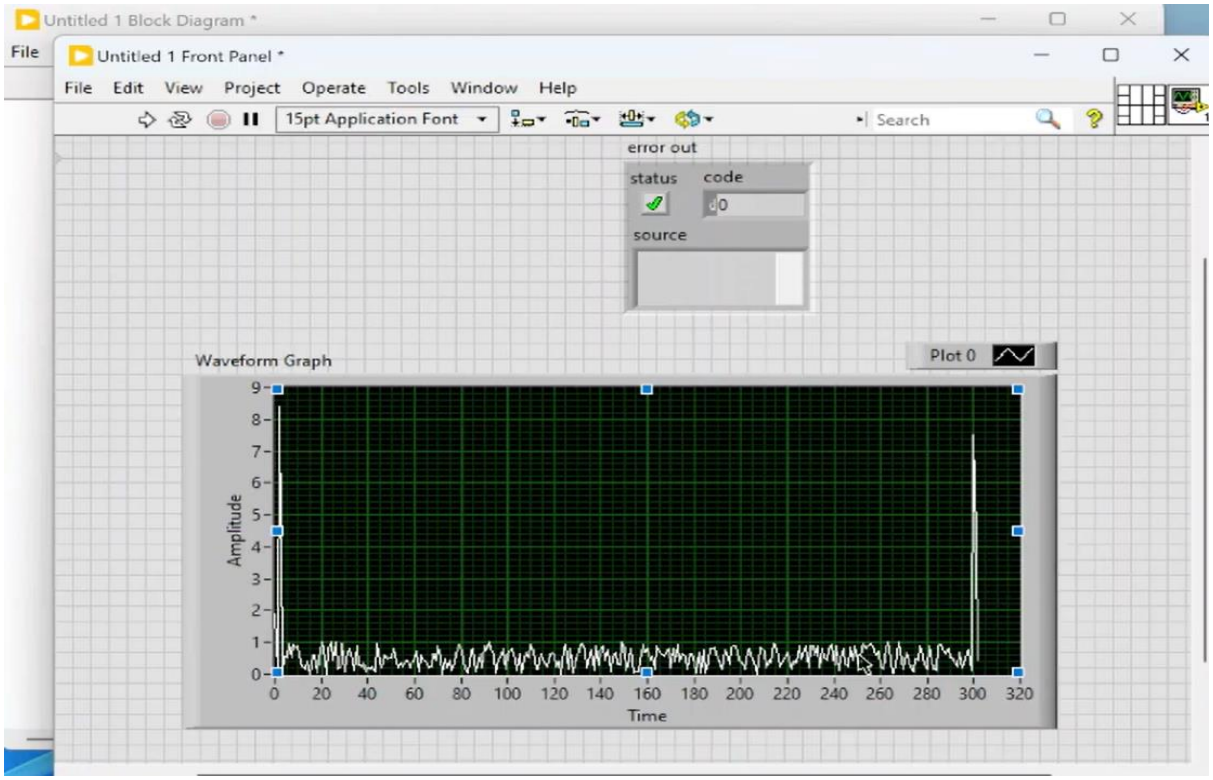
## Procedure:

- **Establish Database Connection:** Use LabVIEW's "DB Tools" functions to establish a connection to the local database. Configure the connection by providing the necessary credentials and database path.
- **Perform SQL Query:** Write an SQL query to fetch data from the database, applying the condition WHERE randomvalue < 0.5 to filter the data.
- **Retrieve Data:** Use the "DB Tools Select Data.vi" function to execute the SQL query and retrieve the relevant data from the database.
- **Process Data:** Once the data is retrieved, it is processed and formatted appropriately. The data is organized in columns and rows and passed to the next step for visualization.
- **Display Data Graphically:** Use a waveform graph to display the processed data. This will show the values of randomvalue that meet the query condition and allow visualization of trends or anomalies.
- **Handle Errors:** Implement error handling to ensure that any issues with database connectivity or data retrieval are addressed. Ensure the database connection is properly closed after the operation is complete.
- **Analyze Results:** Observe the waveform graph output, noting any anomalies or trends in the data. Investigate spikes or outliers, as observed in the output, to understand their cause.

## Circuit Connectivity:




## Output:




## Results:

The experiment successfully connected LabVIEW to a local database, retrieved filtered data (randomvalue < 0.5), and displayed it on a waveform graph. The output showed mostly low values with a spike, confirming correct data retrieval and visualization.

## 11. To configuring the gateways and upload the data to cloud server using LABVIEW.

## Aim:

The aim of this LabVIEW program is to send data from a local source (such as a sensor or a device) to the ThingSpeak platform using its API. This data is updated in specific fields of a ThingSpeak channel for real-time monitoring and analysis. The program interacts with the ThingSpeak cloud service to push data by making HTTP requests via the ThingSpeak API.

## Components Required:

| Component | Description |
|---|---|
| LabVIEW Software | Used to create the program and communicate with ThingSpeak. |
| ThingSpeak API Key | Unique authentication key to interact with ThingSpeak platform. |
| ThingSpeak Channel | A channel on ThingSpeak where the data will be uploaded (requires at least one field). |
| Internet Connection | To send data via HTTP requests to the ThingSpeak cloud service. |
| Sensor/Data Source | (Optional) A device or sensor to collect and provide data to be uploaded. |
| LabVIEW VIs | Includes VIs like OpenHandle.vi, GET.vi, Concatenate Strings.vi, Flatten to String.vi, Format Into String.vi, and CloseHandle.vi for communication. |
| HTTP Request Interface | Web interface for sending HTTP requests (usually part of LabVIEW's built-in functions). |
| Error Handling VIs | To handle and troubleshoot any communication or data issues. |

**Procedure to Send Data to ThingSpeak Cloud**

1. **Create a ThingSpeak Account and Channel:**

- Sign up for an account on ThingSpeak.
- Create a new channel in ThingSpeak to store your data. Define the fields you want to store (e.g., Field 1, Field 2, etc.).

2. **Get the API Key:**

- Once your channel is created, ThingSpeak will provide you with an API Key.
- You can find this in the Channel Settings under the API Keys section.
- You'll need this API key to authenticate the requests you send from LabVIEW.

3. **LabVIEW Setup:**

- Open LabVIEW and create a new project or VI (Virtual Instrument).
- Construct the LabVIEW block diagram to send data to ThingSpeak using HTTP requests.

4. **Components Needed:**

- HTTP Request (GET or POST): Use LabVIEW's HTTP API or a TCP/IP connection to interact with ThingSpeak.
- Format Into String.vi: To construct the full URL for the API request.
- Concatenate Strings.vi: To build the URL string (including the API key and data values).
- Error Handling: Use error indicators to handle any errors during communication.
- Wait.vi: To pause between consecutive operations, allowing enough time for data submission.

5. **Construct the URL with Data:**
   **The data is sent via a URL in this format:**

   **https://api.thingspeak.com/update?api_key=YOUR_API_KEY&field1=value1&field2=value2**

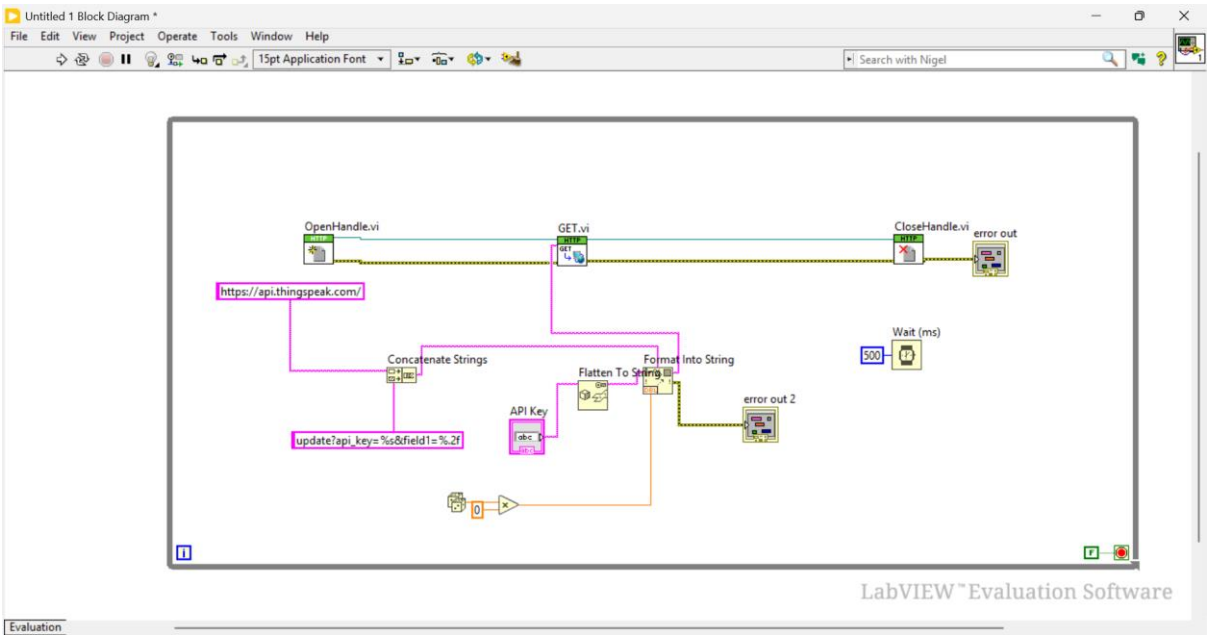
6. **Replace YOUR_API_KEY with your actual ThingSpeak API Key.**
7. **Replace value1, value2 with the data you want to send (e.g., temperature, humidity).**

**Step-by-Step Block Diagram Setup:**

- OpenHandle.vi: Initialize communication with the ThingSpeak API.
- Concatenate Strings.vi: Combine the base API URL with the API key and field data to create the complete URL.
  - Example:
  - https://api.thingspeak.com/update?api_key=ABCD1234APIKEY&field1=25&field2=50
- Format Into String.vi: Use this block to ensure the string is in the correct format for HTTP requests.
- GET.vi or POST.vi: Use this block to send the constructed URL to the ThingSpeak API. This sends the data to your ThingSpeak channel.

- Error Handling: Use error out and error in to monitor the success or failure of the request.
- Send Data (HTTP Request):
- After constructing the URL with the necessary fields and API key, use LabVIEW's GET.vi (or POST.vi depending on your needs) to send the data.
- GET.vi sends a simple HTTP request to the server to update the data.
- Close Connection:
- After sending the data, close the connection with CloseHandle.vi.
- Verify Data in ThingSpeak:
- Go to the ThingSpeak channel page and check if the data was updated successfully.
- The Field values on the channel will reflect the new data points.

**Circuit Diagram:**




**Output:**

**New Channel Setup:**

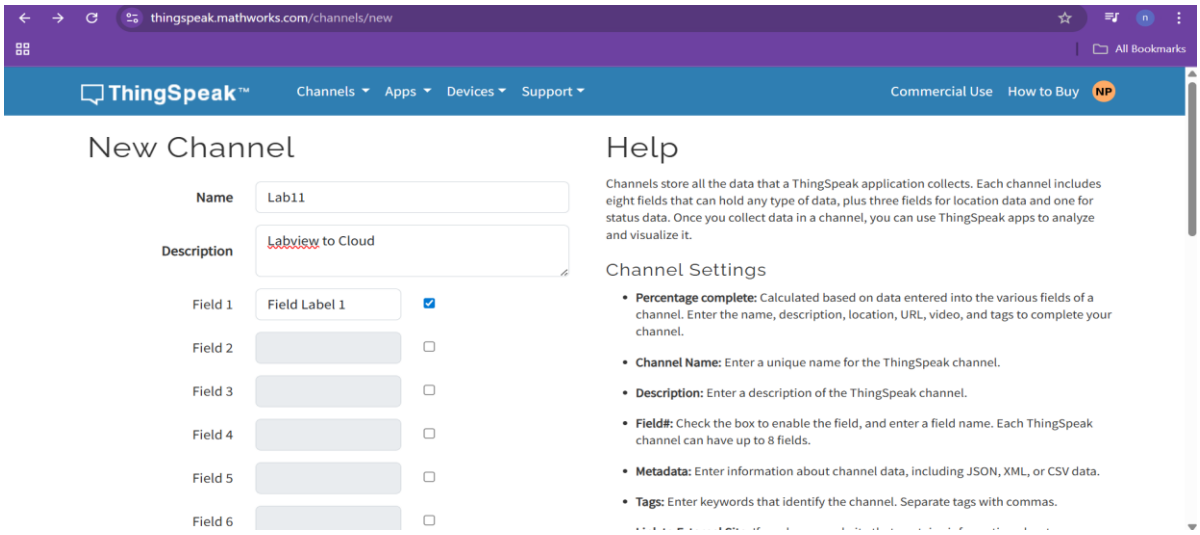
- **Name:** "Lab11" is the name given to the new ThingSpeak channel.
- **Description:** "Labview to Cloud" is the description for the channel, which indicates the data might be coming from LabVIEW and being uploaded to the cloud.

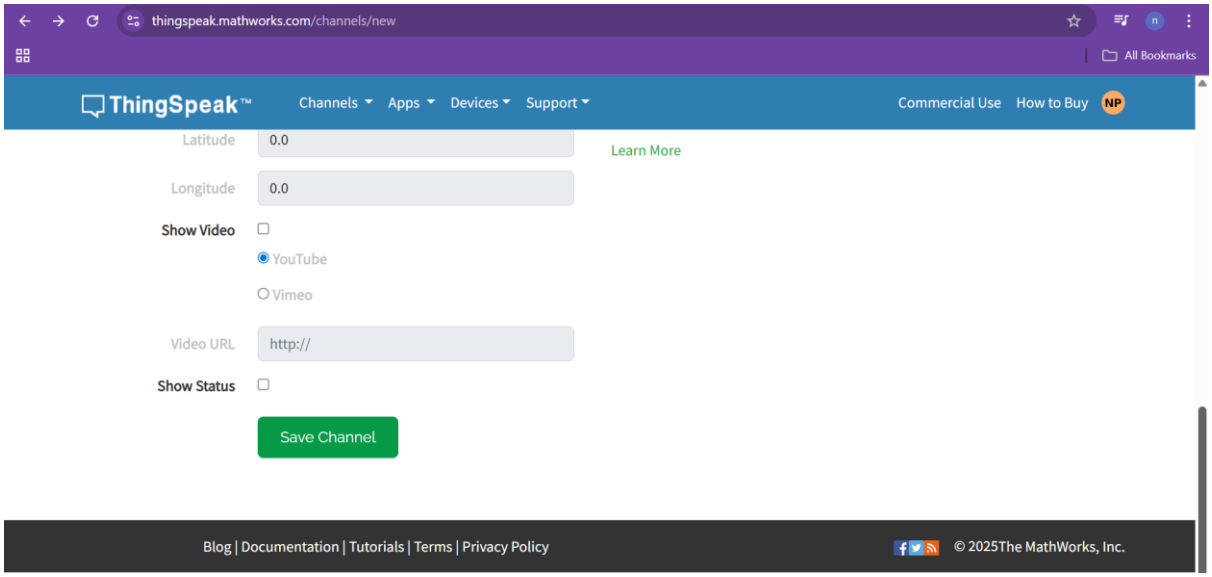
**Field Setup:**

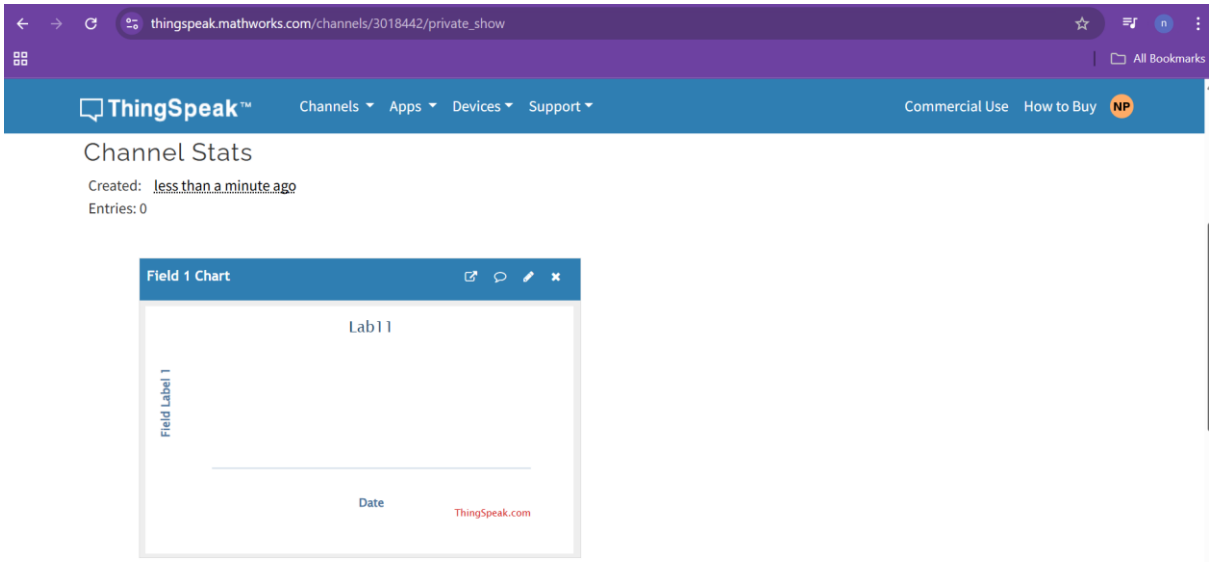
- This section shows fields that can hold different types of data. In this case, there are 6 fields available to label. For example, "Field Label 1" is already labeled as the first data field for the channel.

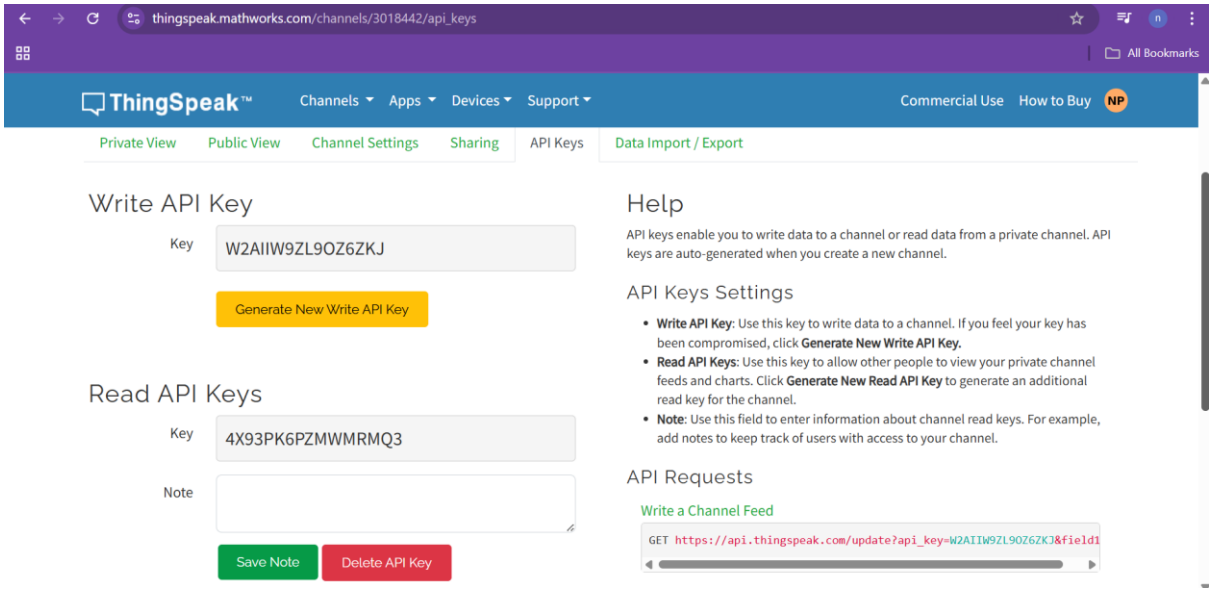
**Channel Settings Help Section:**

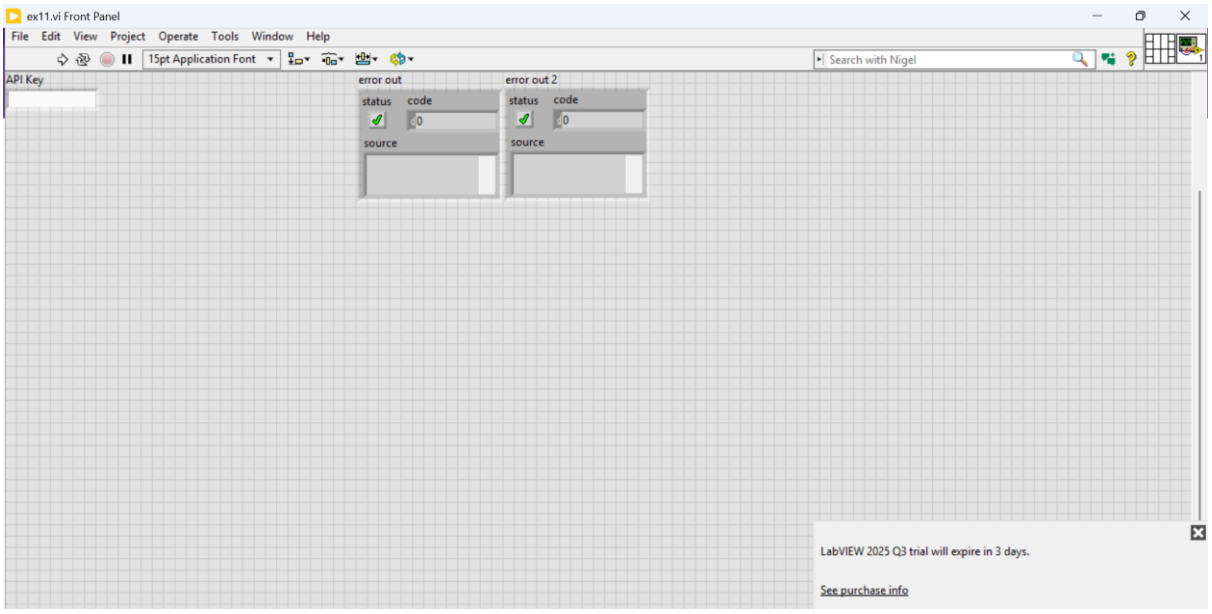
- **Percentage Complete:** A progress indicator for setting up the channel, which updates based on how much information to be entered.
- **Channel Name:** A field to define the channel's name (already labeled as "Lab11").
- **Description:** Describes the purpose of the channel (already filled with "Labview to Cloud").
- **Metadata:** A place to enter additional information about the channel (e.g., file formats like JSON, XML, or CSV for export).
- **Tags:** A section to add keywords or tags for easy identification and searchability (though it's not filled in the screenshot).

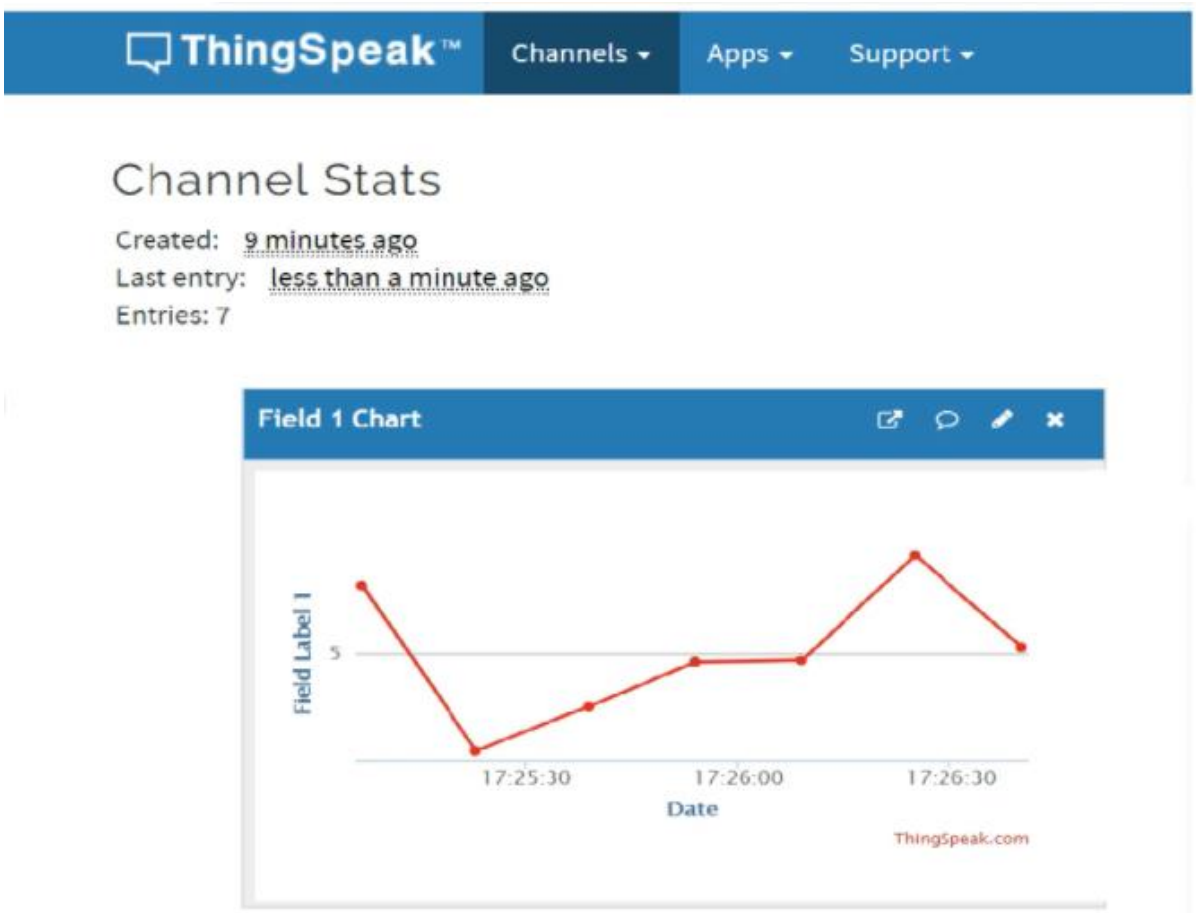



thingspeak.mathworks.com/channels/new
All Bookmarks
ThingSpeak™
Channels
Apps
Devices
Support
Commercial Use
How to Buy
NP
Latitude
0.0
Learn More
Longitude
0.0
Show Video
YouTube
Vimeo
Video URL
http://
Show Status
Save Channel
Blog | Documentation | Tutorials | Terms | Privacy Policy
© 2025The MathWorks, Inc.


thingspeak.mathworks.com/channels/3018442/private_show
All Bookmarks
ThingSpeak™
Channels
Apps
Devices
Support
Commercial Use
How to Buy
NP
Channel Stats
Created: less than a minute ago
Entries: 0
Field 1 Chart
Lab11
Field Label 1
Date
ThingSpeak.com


thingspeak.mathworks.com/channels/3018442/api_keys
All Bookmarks
ThingSpeak™
Channels
Apps
Devices
Support
Commercial Use
How to Buy
NP
Private View
Public View
Channel Settings
Sharing
API Keys
Data Import / Export
Write API Key
Key
W2AIIW9ZL9OZ6ZKJ
Generate New Write API Key
Read API Keys
Key
4X93PK6PZMWMRMQ3
Note
Save Note
Delete API Key
Help
API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.
API Keys Settings
Write API Key: Use this key to write data to a channel. If you feel your key has been compromised, click Generate New Write API Key.
Read API Keys: Use this key to allow other people to view your private channel feeds and charts. Click Generate New Read API Key to generate an additional read key for the channel.
Note: Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.
API Requests
Write a Channel Feed
GET https://api.thingspeak.com/update?api_key=W2AIIW9ZL9OZ6ZKJ&field1


ex11.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Search with Nigel
API Key
error out
status
code
source
error out 2
status
code
source
LabVIEW 2025 Q3 trial will expire in 3 days.
See purchase info




## Results:

The experiment successfully transmitted data from LabVIEW to the ThingSpeak cloud platform. Using an API key and constructing the appropriate URL with field values for random values, the program sent real-time data updates to the ThingSpeak channel.