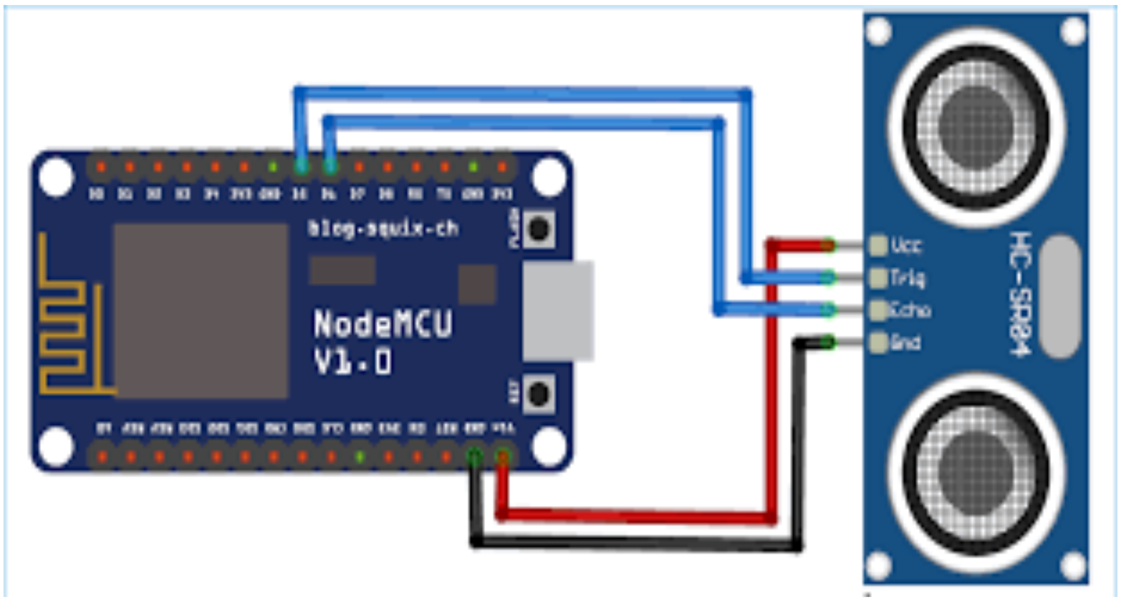
IOT Based Smart Jar using NodeMCU ESP8266 and Ultrasonic Sensor



Submitted By:6861 Shreya Bhattacharjee
&
6902 Suhana Rauthar

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE
(Autonomous)

NEW PANVEL

PROJECT REPORT ON

"IOT Based Smart Jar using NodeMCU ESP8266 and Ultrasonic Sensor"

IN PARTIAL FULFILLMENT OF

MASTER OF DATA ANALYTICS

SEMESTER III– 2024-25

PROJECT GUIDE
Name YOGESH KENE

SUBMITTED BY: SHREYA BHATTACHARJEE &
SUHANA RAUTHAR

ROLL NO: 6861 & 6902

INDEX

Abstract

The **IoT-based Smart Jar** project is designed to automate the process of monitoring and managing the contents of a jar or container, such as grains, liquids, or other consumables, by integrating an **ultrasonic sensor** and a **NodeMCU ESP8266 microcontroller**. This system solves the problem of manually checking container levels by providing a digital solution that tracks the content level in real-time. The smart jar utilizes the ultrasonic sensor to accurately measure the distance between the sensor and the top surface of the jar's contents, thereby determining how full or empty the jar is.

The data collected by the ultrasonic sensor is processed by the NodeMCU, which wirelessly transmits the information to the **ThingSpeak IoT platform** over Wi-Fi. On ThingSpeak, users can visualize the jar's content level through real-time graphs and dashboards, which are accessible from any device with an internet connection. This provides a convenient and efficient solution for managing inventory or consumables in homes, kitchens, offices, or industries.

One of the key features of the system is its ability to trigger notifications and alerts when the contents of the jar reach predefined threshold levels, such as when the jar is almost empty (e.g., below 25%) or nearly full. These alerts can be configured to send notifications via email, SMS, or other IoT notification services, ensuring that users are informed in a timely manner to refill or stop adding more contents to the jar.

Introduction

In today's fast-paced world, the demand for automation and intelligent systems has increased significantly across various domains, including homes, industries, and businesses. Automation plays a pivotal role in simplifying daily tasks, reducing manual effort, and enhancing overall efficiency. One such area where automation can be effectively applied is inventory and resource management, particularly in monitoring the stock levels of consumables such as grains, liquids, spices, and other household or industrial materials. Managing these resources manually can be time-consuming and prone to errors, leading to wastage, understocking, or overstocking. This is where **smart containers** come into play.

Smart containers, enabled by **Internet of Things (IoT)** technology, offer an automated solution to monitor the content levels in real time, ensuring that users are always informed about the quantity of materials stored in the containers. These intelligent systems can track inventory levels, detect when a container is nearing empty, and send alerts or notifications to the user. Such features are particularly valuable in homes, kitchens, warehouses, offices, and industries where consumables are frequently used and need constant replenishment. Moreover, smart containers help in optimizing resource management by providing data-driven insights into usage patterns and trends.

This project introduces a **Smart Jar**, a container equipped with IoT technology to monitor and manage the content levels of various materials such as grains, liquids, and other consumables. The Smart Jar system is built around a **NodeMCU ESP8266 microcontroller**, which serves as the core processing unit of the system, and an **ultrasonic sensor** that accurately measures the distance between the sensor and the top surface of the jar’s contents. The distance measured is directly proportional to the level of material present in the jar, allowing the system to calculate how full or empty the jar is at any given time.

On ThingSpeak, the jar’s content level is displayed graphically, enabling users to monitor the jar remotely through any internet-connected device, such as smartphones, tablets, or computers. The ThingSpeak platform provides real-time visualization in the form of graphs, charts, and dashboards, allowing users to gain insights into how much material remains in the jar at any given moment. This visual representation ensures that users can monitor the stock levels without physically inspecting the jar, making it highly convenient, especially for users managing multiple containers.

By integrating modern IoT technology with basic household items like jars, this project showcases a practical application that blends innovation with everyday life. The Smart Jar is a simple yet powerful tool for making resource management more efficient and automated, marking the next step in the evolution of smart homes and industries.

Literature Review

Numerous studies and projects have explored the potential of IoT for real-time monitoring of various systems, such as smart homes, healthcare, and industrial applications. These systems often use sensors combined with microcontrollers to gather and transmit data to a cloud platform, enabling users to access real-time insights and make data-driven decisions.

1. **Smart Bins and Inventory Systems**:
   IoT-based smart bins use ultrasonic sensors to detect waste levels and send notifications when bins are full. This concept of real-time monitoring and alerts is widely used in inventory management systems, where warehouses and industries automate stock level tracking using similar sensor technology.

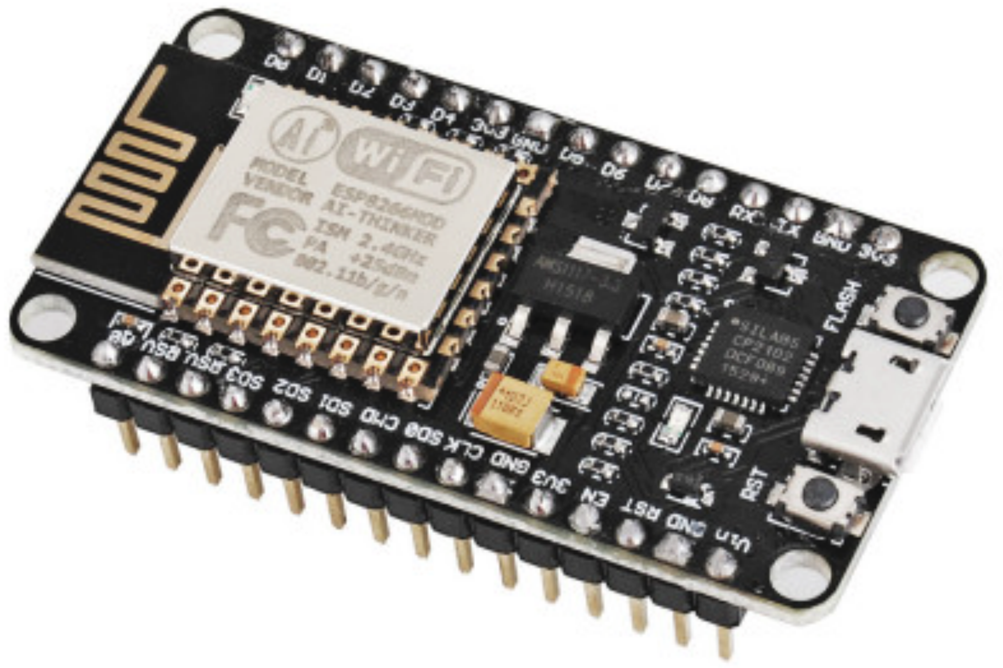
2. **Ultrasonic Sensors in Level Monitoring**:
   Ultrasonic sensors, like the **HC-SR04**, have been effectively used in various applications to measure the distance between the sensor and a surface. These sensors work by emitting ultrasonic waves and calculating the time it takes for the echo to return, making them highly accurate for distance and level measurement.

3. **IoT Cloud Platforms**:
   Platforms like **ThingSpeak**, **Blynk**, and **Adafruit IO** have emerged as popular tools for IoT data visualization. They provide features such as data storage, real-time graphs, alerts, and the ability to access data from anywhere via the internet. These platforms are widely used in projects that require real-time remote monitoring and analytics.

Methodology

Hardware Components

1. **NodeMCU ESP8266**:

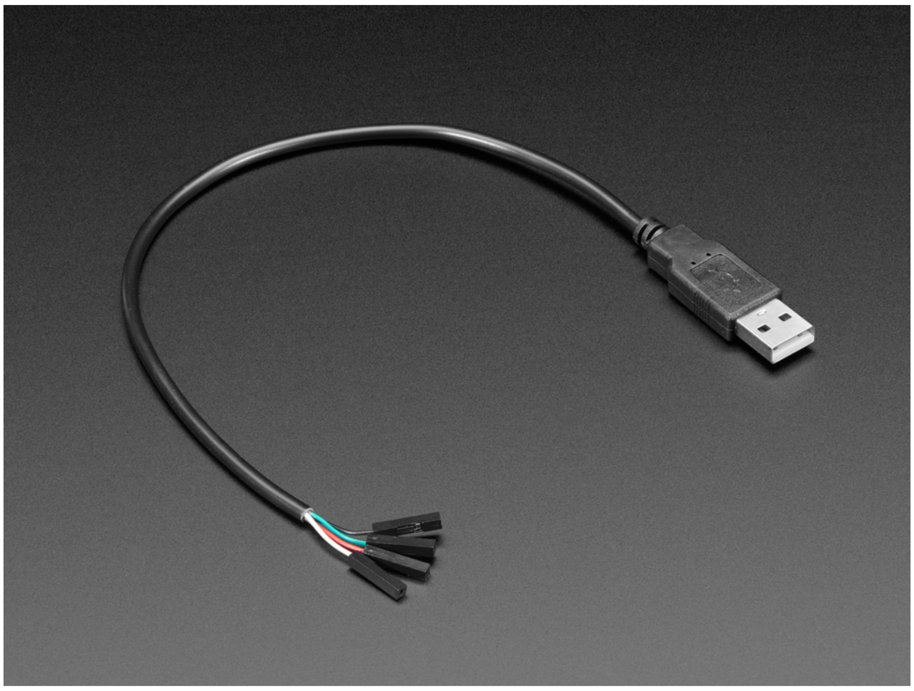

D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
ESP8266MOD
AI-THINKER
ISM 2.4GHz
FLASH
RST
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin

This is a Wi-Fi-enabled microcontroller that collects sensor data and transmits it to the cloud (ThingSpeak). The NodeMCU is responsible for controlling the ultrasonic sensor, reading distance data, and sending the information to the IoT platform.

2. **Ultrasonic Sensor (HC-SR04)**

This sensor is used to measure the distance between the sensor and the top surface of the contents inside the jar. It operates by emitting ultrasonic waves and measuring the time it takes for the waves to bounce back, allowing it to calculate the distance accurately

3. **Jumper wire**



A **jumper wire** is a short electrical wire used to make temporary or permanent connections between different points in a circuit. These wires are typically used with breadboards, development boards (like Arduino or NodeMCU), and other prototyping platforms to facilitate connections without soldering. Jumper wires come in three main types:

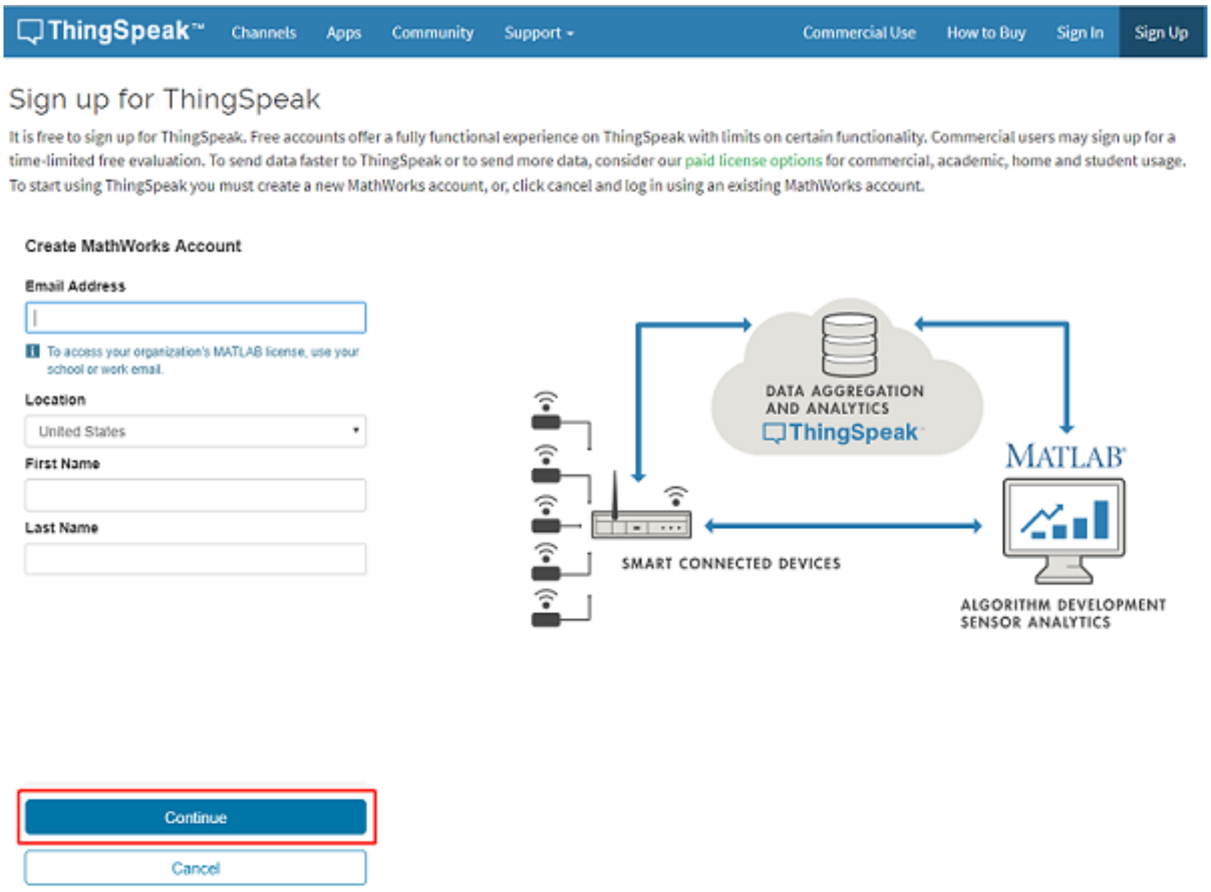
1. **Male-to-Male**: Both ends have pins that can be inserted into breadboards or female connectors.
2. **Male-to-Female**: One end has a pin, and the other has a socket, making it useful for connecting to headers or pins on devices.
3. **Female-to-Female**: Both ends have sockets, used for connecting between two male headers.

Software Components

1.Arduino IDE:

The Arduino IDE is used to write and upload the code to the NodeMCU. The code reads data from the ultrasonic sensor, calculates the distance, and sends it to the ThingSpeak platform.

2.ThingSpeak


ThingSpeak™ Channels Apps Community Support ▾ Commercial Use How to Buy Sign In Sign Up

Sign up for ThingSpeak

It is free to sign up for ThingSpeak. Free accounts offer a fully functional experience on ThingSpeak with limits on certain functionality. Commercial users may sign up for a time-limited free evaluation. To send data faster to ThingSpeak or to send more data, consider our paid license options for commercial, academic, home and student usage. To start using ThingSpeak you must create a new MathWorks account, or, click cancel and log in using an existing MathWorks account.

Create MathWorks Account

Email Address

To access your organization's MATLAB license, use your school or work email.

Location: United States

First Name

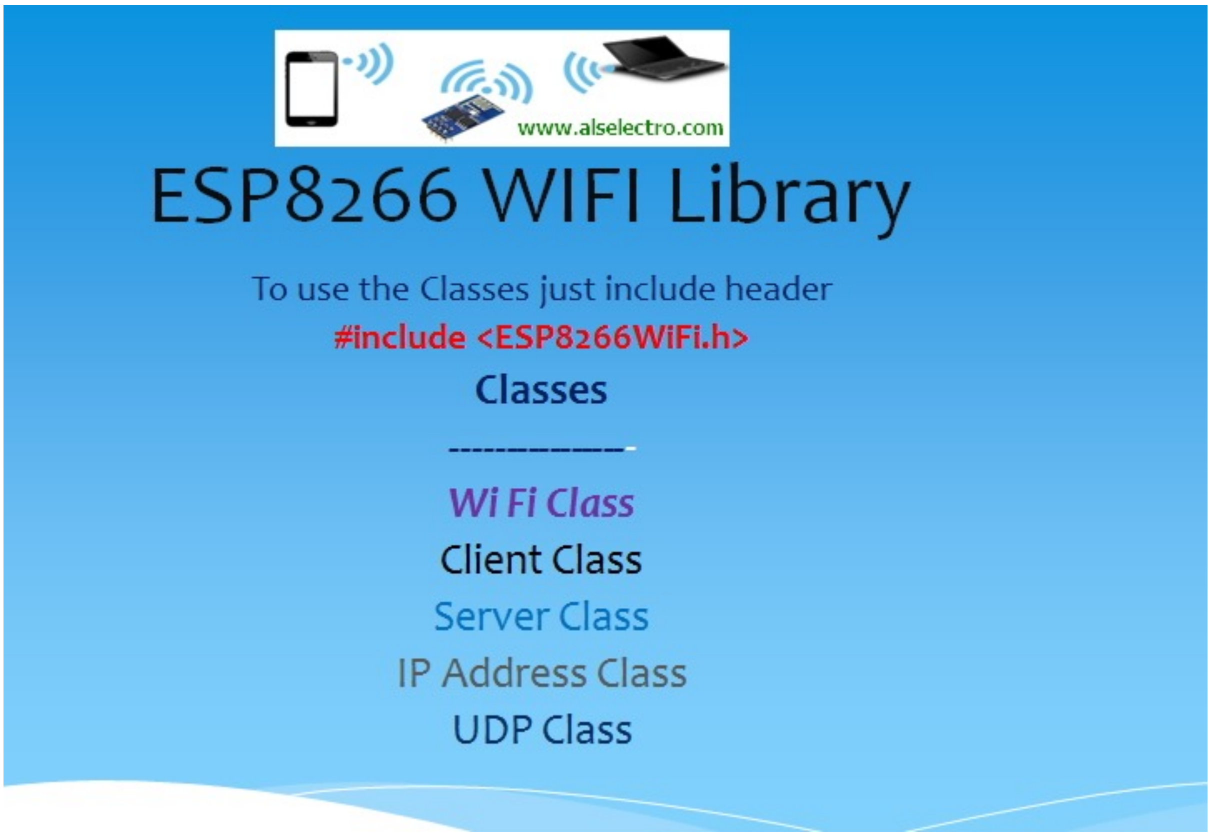
Last Name

Continue

Cancel

ThingSpeak is an IoT analytics platform that allows the NodeMCU to store and visualize data. ThingSpeak provides real-time charts for the jar's content level and can trigger alerts when certain thresholds are met.

4) **ESP8266WiFi Library**:



ESP8266 WIFI Library

To use the Classes just include header

#include <ESP8266WiFi.h>

Classes

Wi Fi Class

Client Class

Server Class

IP Address Class

UDP Class

www.alselectro.com

This library is used in the Arduino code to enable Wi-Fi connectivity for the NodeMCU. It helps establish a connection with the local Wi-Fi network and communicate with the ThingSpeak server.

Implementation Steps

1.Sensor Setup:
The ultrasonic sensor is mounted on the top of the jar, pointing downward to measure the distance between the sensor and the top of the contents. The sensor will give accurate readings whether the jar is full or empty.

2. NodeMCU Programming:

Using the Arduino IDE, the NodeMCU is programmed to:

- Initialize the ultrasonic sensor.

- Continuously measure the distance between the sensor and the contents.
- Calculate the content level based on the distance.
- Send the data to the ThingSpeak platform at regular intervals.

3.ThingSpeak Setup:

On ThingSpeak, a new channel is created where the data from the NodeMCU will be uploaded. The channel can be configured to display real-time graphs showing the content level and to send notifications via email or SMS when certain conditions (e.g., jar is empty) are met.

Input and Sample Data

1) **Sensor Data (Distance Measurements)**:

   The ultrasonic sensor will provide distance measurements ranging from 0 cm (jar is full) to 20 cm (jar is empty).

2) **Sample Data**:

   0 cm: Jar is full (100%).

   5 cm: Jar is about 75% full.

   10 cm: Jar is about 50% full.

   15 cm: Jar is about 25% full.

   20 m: Jar is empty.

3) **Threshold Levels**:

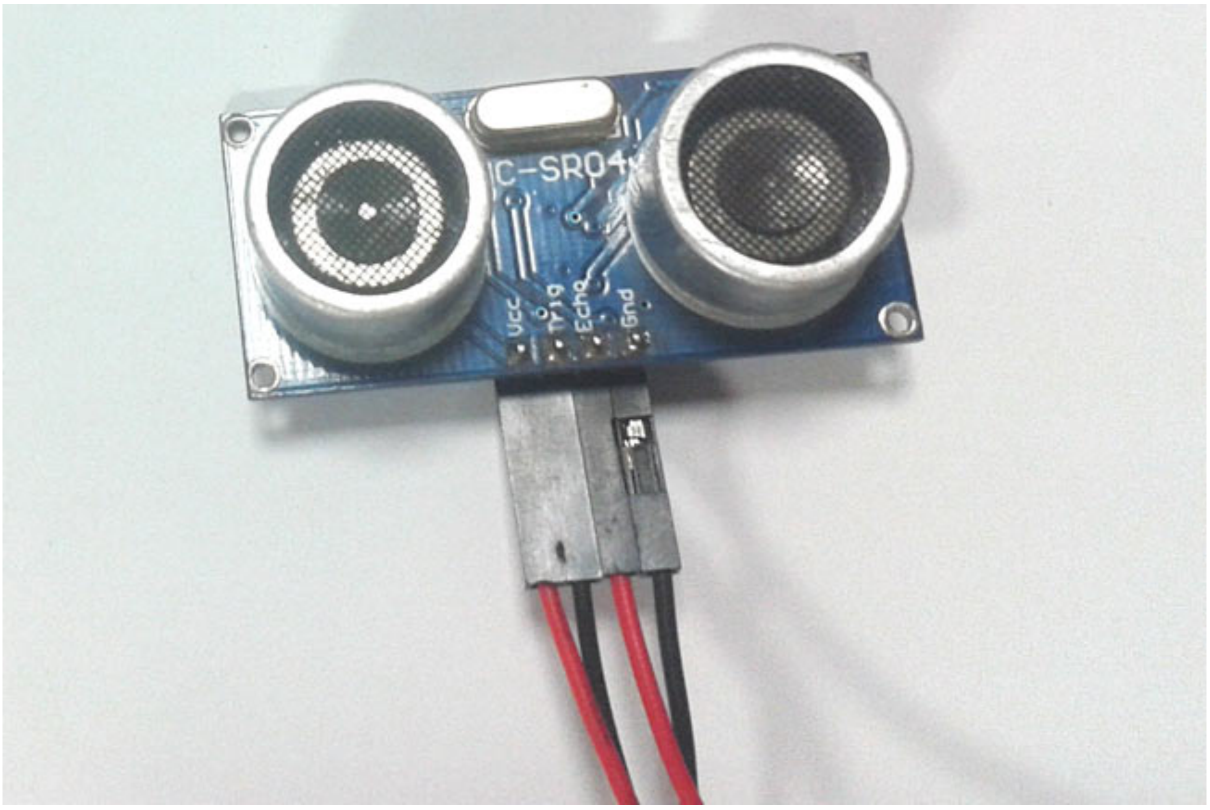
Alerts will be set at:

15 cm: Low-level alert (25% full).

20 cm: Empty jar alert.

This **smart Jar** allows us to keep track of the stocks, and it is easily accessible from using the internet. The Jar includes an ultrasonic sensor at the top of it and uses the ultra-sonic reflected waves to figure out at what extent the Jar is filled and how much space is left inside the jar. Whenever the amount of content changes in the jar, it is sensed by the NodeMCU, and the same is updated on the webserver. This can be helpful to track supplies and plan for restocking from anywhere in the world. Previously we used Arduino and ESP8266-01 to build smart dustbin using the same methodology.

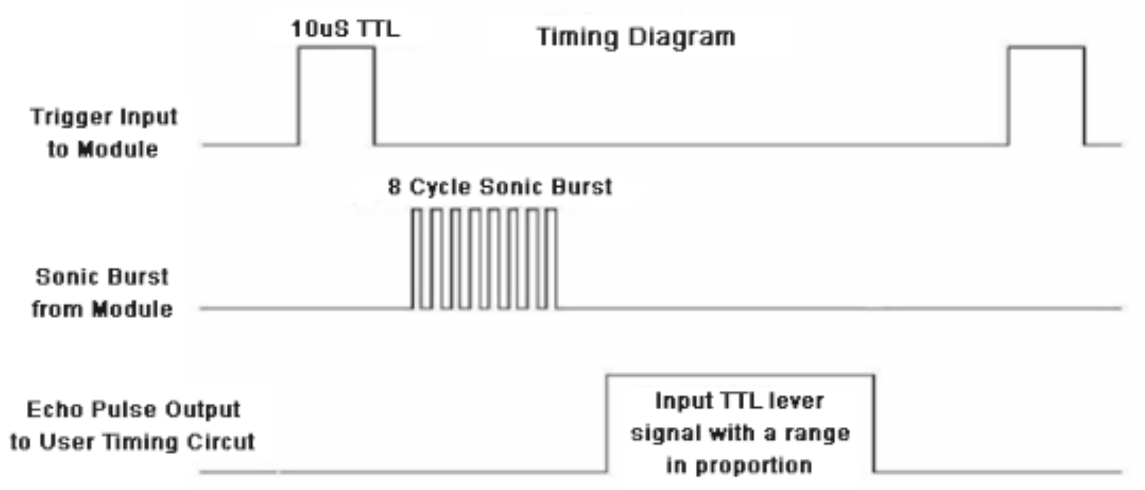
Working of Ultrasonic sensor



As you can see it has two circular eyes like projections and four pins coming out of it. The two eye like projections are the Ultrasonic wave (hereafter referred as US wave) Transmitter and receiver. The transmitter emits an US wave at a frequency of 40Hz, this wave travels through the air and gets reflected back when it senses an object. The returning waves are observed by the receiver. Now we know the time taken for this wave to get reflected and come back and the speed of the US wave is also universal (3400cm/s). Using this information and the below high school formulae we can calculate the distance covered.

```
Distance = Speed × Time
```

Now that we know how an US sensor works, let us how it can be interfaced with any MCU/CPU using the four pins. These **four pins are Vcc, Trigger, Echo and Ground** respectively. The module works on +5V and hence the Vcc and ground pin is used to power the module. The other two pins are the I/O pins using which we communicate to our MCU. The **trigger pin should be declared as an output pin** and made high for a 10uS, this will transmit the US wave into the air as 8 cycle sonic burst. Once the wave is observed the Echo pin will go high for the exact interval of time which was taken by the US wave to return back to the sensor module. Hence this **Echo pin will be declared as input** and a timer will be used to measure how long the pin was high. This could further be understood by the timing diagram below.


10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timing Circut
Input TTL lever signal with a range in proportion

Code For Project

Code for cloud reading

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

// Pin Definitions
const int trigPin = D5;
const int echoPin = D6;

// Sensor Variables
long duration;
int distance;
float level;

// WiFi Credentials
const char* ssid = "Rauthars";
const char* password = "Sahul@0517";

// ThingSpeak Settings
const char* host =
"api.thingspeak.com"; const
char* writeAPIKey =
"FVU942CPIEW58UNW"; const int
httpPort = 80;
WiFiClient client;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);

  // Connect to WiFi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
```

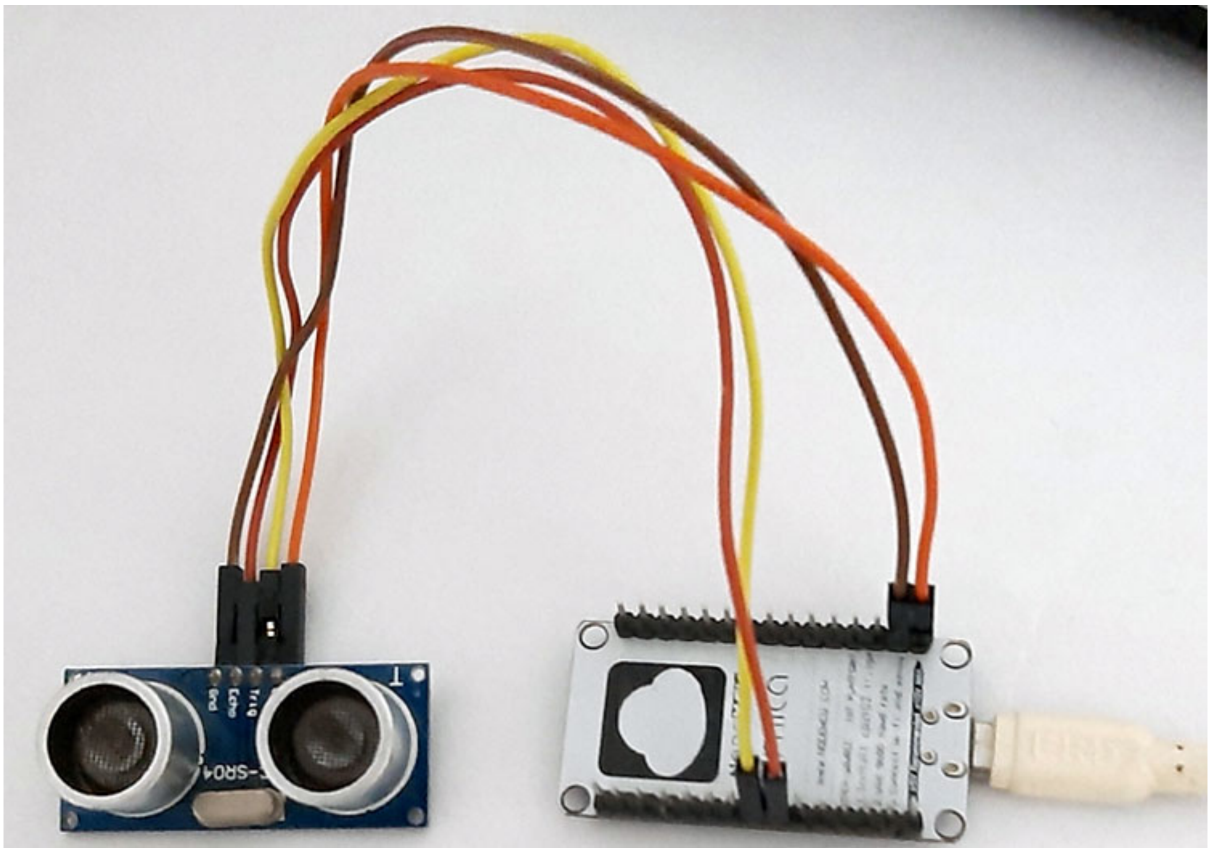
```
while (WiFi.status() !=
  WL_CONNECTED) { delay(500);
    Serial.print(".");
   }


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

void loop() {
  // Calculate distance
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);

  // Calculate distance in cm
  distance = duration * 0.0340 / 2;

  // Assuming the jar's height is 17 cm
  level = ((17 - distance) / 17.0) * 100;

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

  Serial.print("Level: ");
  Serial.print(level);
  Serial.println("%");

  // Update ThingSpeak
  if (client.connect(host, httpPort)) {
    String url = "/update?api_key=";
    url += writeAPIKey;
    url += "&field1=";
    url += String(level);

    Serial.print("Requesting URL: ");
    Serial.println(url);

     client.print(String("GET ") + url + "
       HTTP/1.1\r\n" + "Host: " + host +
                    "\r\n" +
               "Connection: close\r\n\r\n");

    while (client.connected()) {
      if (client.available()) {
        String line = client.readStringUntil('\r');
        Serial.print(line);
      }
    }

    Serial.println();
    Serial.println("Data sent to ThingSpeak");
  } else {
    Serial.println("Connection to
  ThingSpeak failed."); }

  client.stop();

  // Wait before sending the next data point
  delay(20000); // ThingSpeak allows updates
 every 15 seconds
  }
```

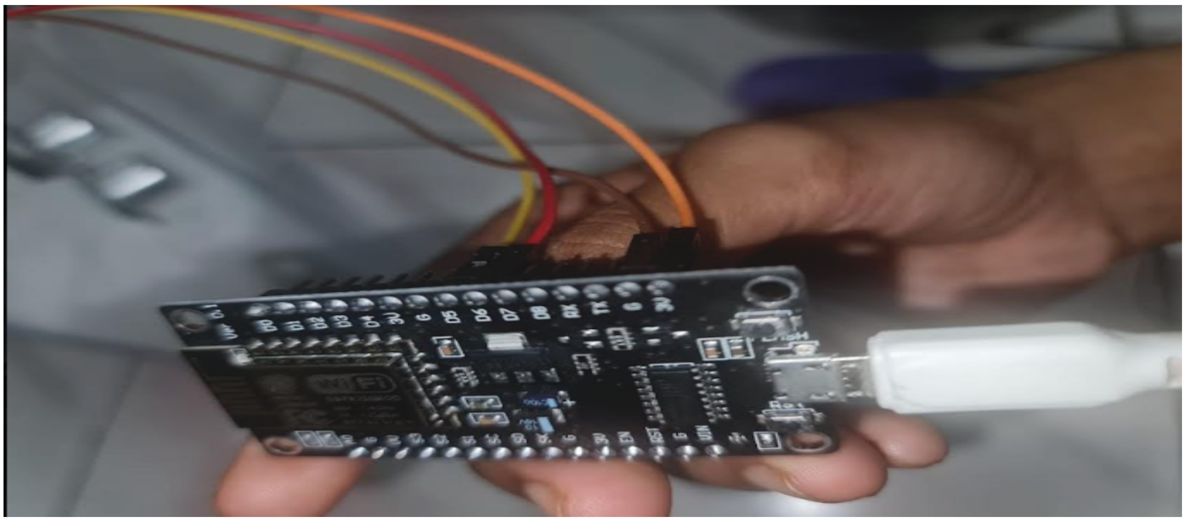
Code for website reading

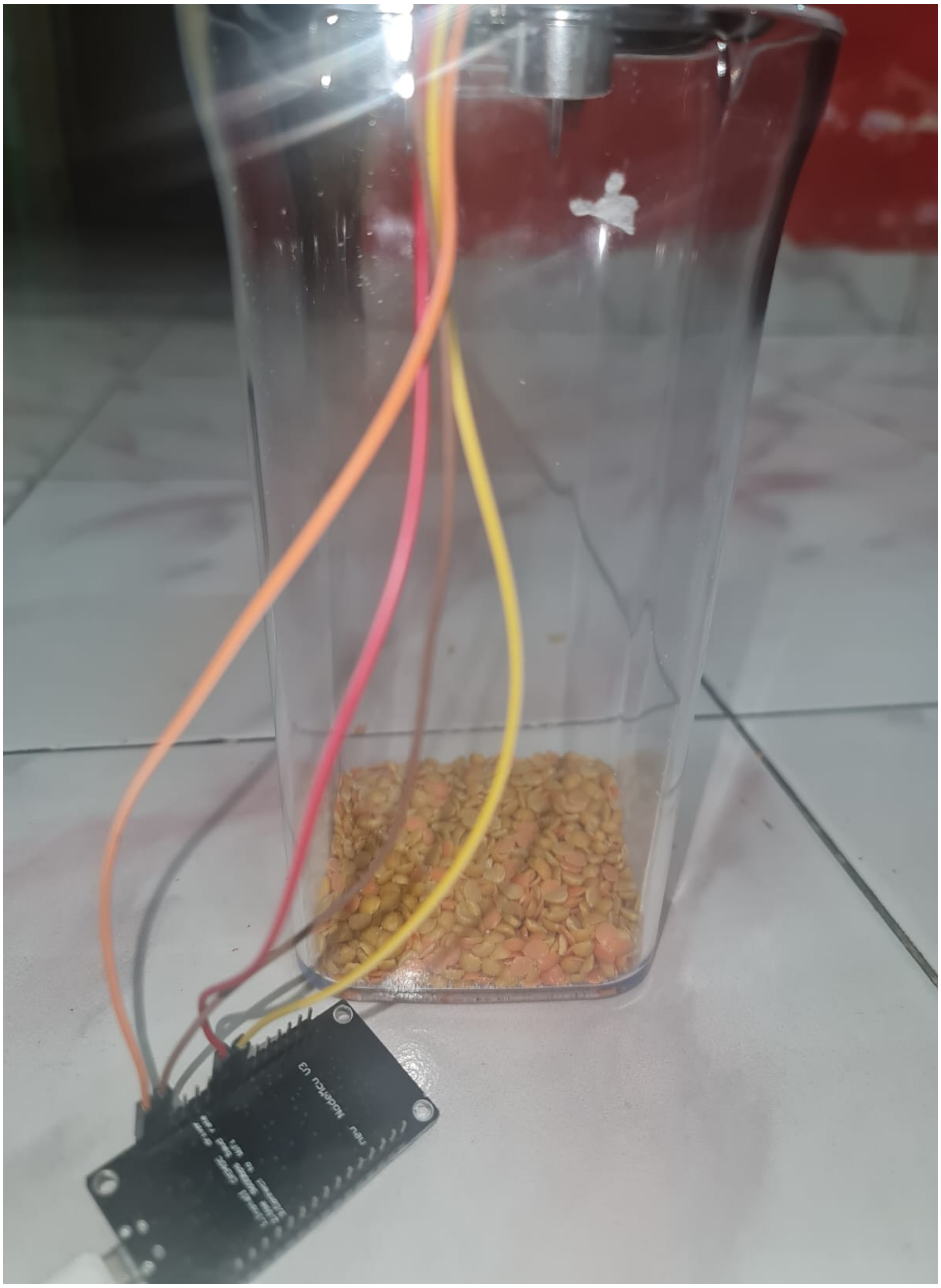
```
#include <ESP8266WiFi.h>

// Define pins for ultrasonic sensor
const int trigPin = D5;
const int echoPin = D6;

// Variables to store duration,
distance, and level long duration;
int distance;
float level;

// WiFi credentials
const char* ssid = "Rauthars";
const char* password = "Sahul@0517";
// Initialize the server at port 80
WiFiServer server(80);

void setup() {
  // Initialize serial communication
  Serial.begin(9600);


  // Initialize pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Connect to WiFi
  Serial.print("Connecting to WiFi
  Network: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

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


    // Once connected, print IP address
    Serial.println("");
    Serial.println("Successfully
    connected to WiFi.");
    Serial.print("IP address is: ");
    Serial.println(WiFi.localIP());

    // Start the server
    server.begin();
    Serial.println("Server started");
  }

void loop() {

    // Trigger the ultrasonic sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Measure the duration of the echo pulse
    duration = pulseIn(echoPin, HIGH);
    distance = duration * 0.0340/ 2;

    // Print distance and level to Serial Monitor
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    // Calculate level as percentage (assuming max
    height is 14 cm) level = ((17.5 - distance) /
    17.5)* 100;
    Serial.print("Level: ");
    Serial.print(level);
    Serial.println(" %");

    // Handle web client
    WiFiClient client = server.available();
    if (client) {
      Serial.println("Web Client connected");

      // Read the client's request
      String request = client.readStringUntil('\r');
      client.flush();

      // Serve the webpage
      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/html");
      client.println("Connection: close"); // Close
  connection after response
      client.println("Refresh: 10"); // Refresh the
  page every 10 seconds

      client.println();
      client.println("<!DOCTYPE HTML>");
      client.println("<html>");
      client.println("<style>html { font-family: Cairo;
  display: block; margin: 0px auto; text-align:
  center; color: #333333; background-color: ##f3ffee;
  }");
      client.println("body { margin-top: 50px; }");
      client.println("h1 { margin: 50px auto 30px;
  font-size: 50px; text-align: center; }");
      client.println(".side_adjust { display: inline-
  block; vertical-align: middle; position: relative;
  }");
      client.println(".text1 { font-weight: 180;
  padding-left: 5px; font-size: 50px; width: 170px;
  text-align: left; color: #3498db; }");
  client.println(".data1 { font-weight: 180;
  padding-left: 1px; font-size: 50px; color:
  #3498db; }");
```

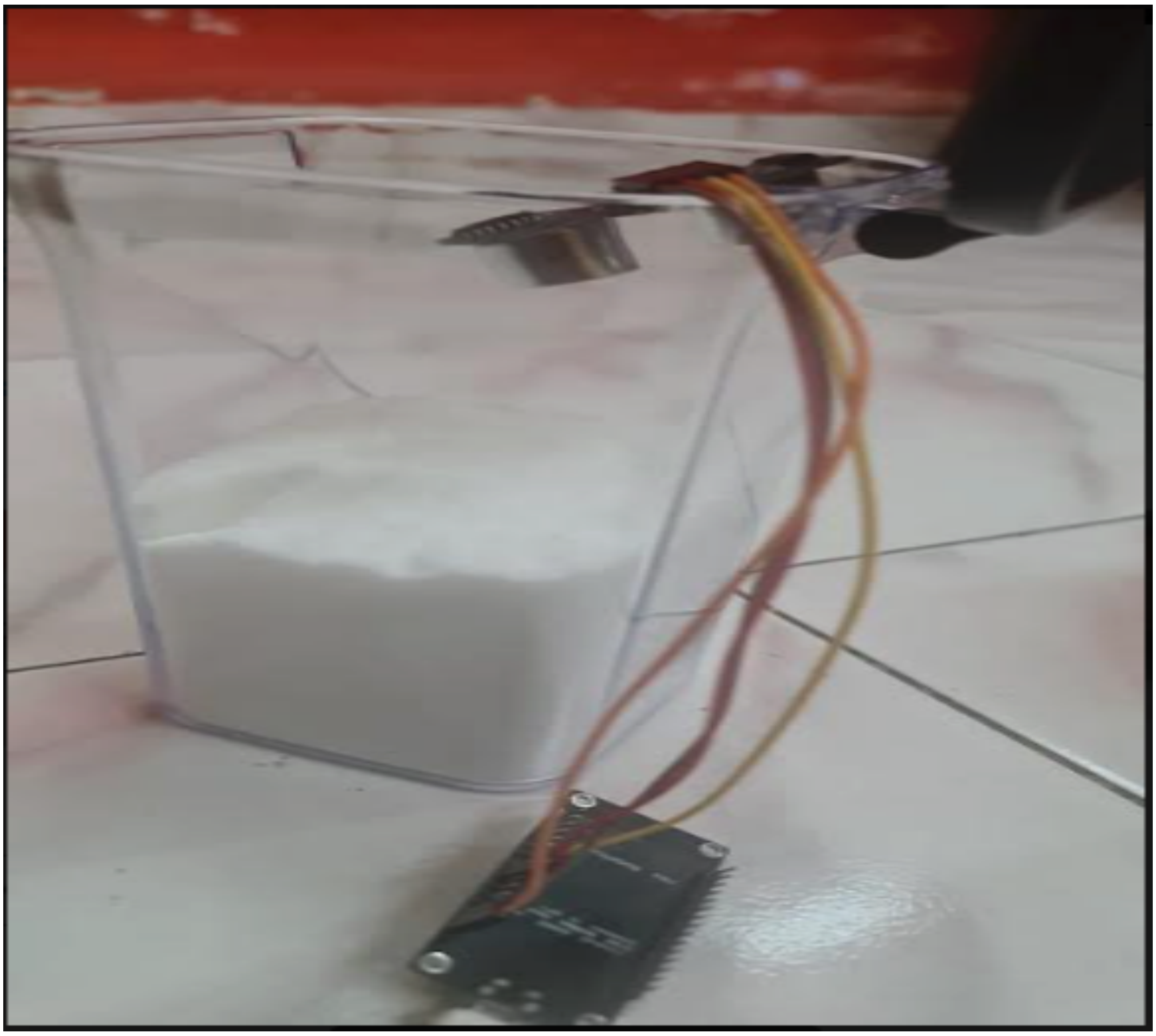
```
    client.println(".data { padding: 1px; }");
    client.println("</style>");
    client.println("</head>");
    client.println("<body>");
    client.println("<div id=\"webpage\">");
    client.println("<h1>IoT Based Jar</h1>");
    client.println("<div class=\"data\">");
    client.println("<div class=\"side_adjust
    text1\">Status:</div>"); client.println("<div
    class=\"side_adjust data1\">");
    client.print(level); // Print the level as
    percentage on the webpage client.println("<div
    class=\"side_adjust text1\">% filled</div>");
    client.println("</div>");
    client.println("</div>");
    client.println("</body>");
    client.println("</html>");

    Serial.println("Data served to client");
  }

  // Wait 1 second before next measurement
  delay(1000);
}
```

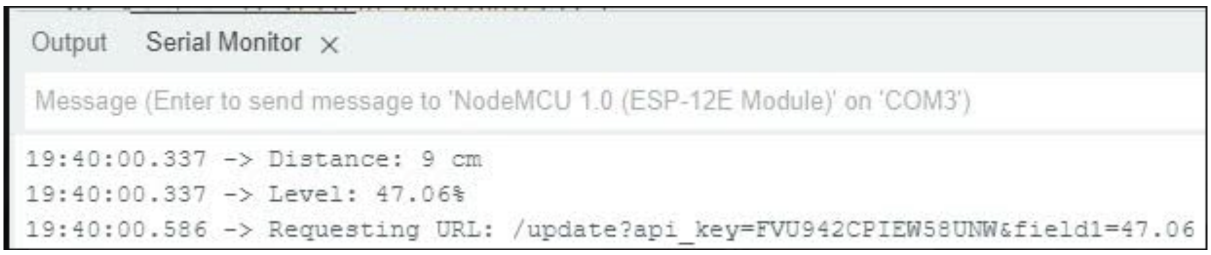
Output


Trig
Echo
Gnd
T
-SR04




new NodeMcu V3

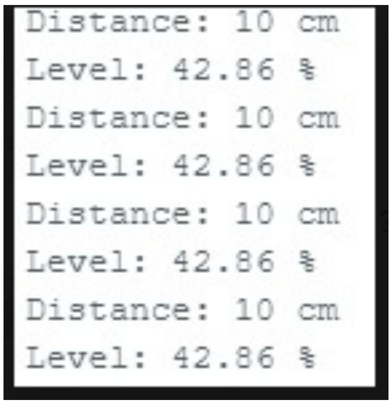




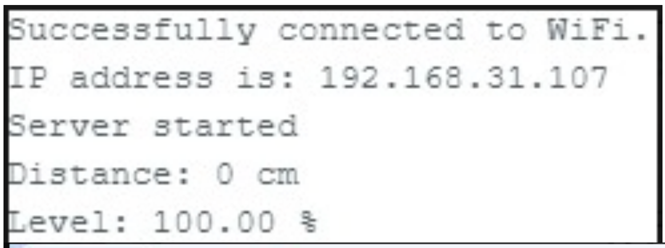
Output   Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM3')

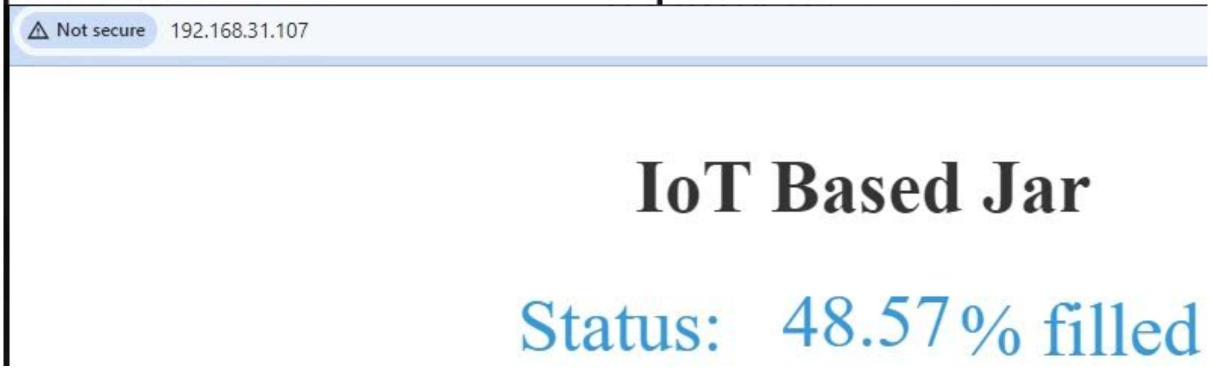
```
19:40:00.337 -> Distance: 9 cm
19:40:00.337 -> Level: 47.06%
19:40:00.586 -> Requesting URL: /update?api_key=FVU942CPIEW58UNW&field1=47.06
```



```
Distance: 10 cm
Level: 42.86 %
Distance: 10 cm
Level: 42.86 %
Distance: 10 cm
Level: 42.86 %
Distance: 10 cm
Level: 42.86 %
```



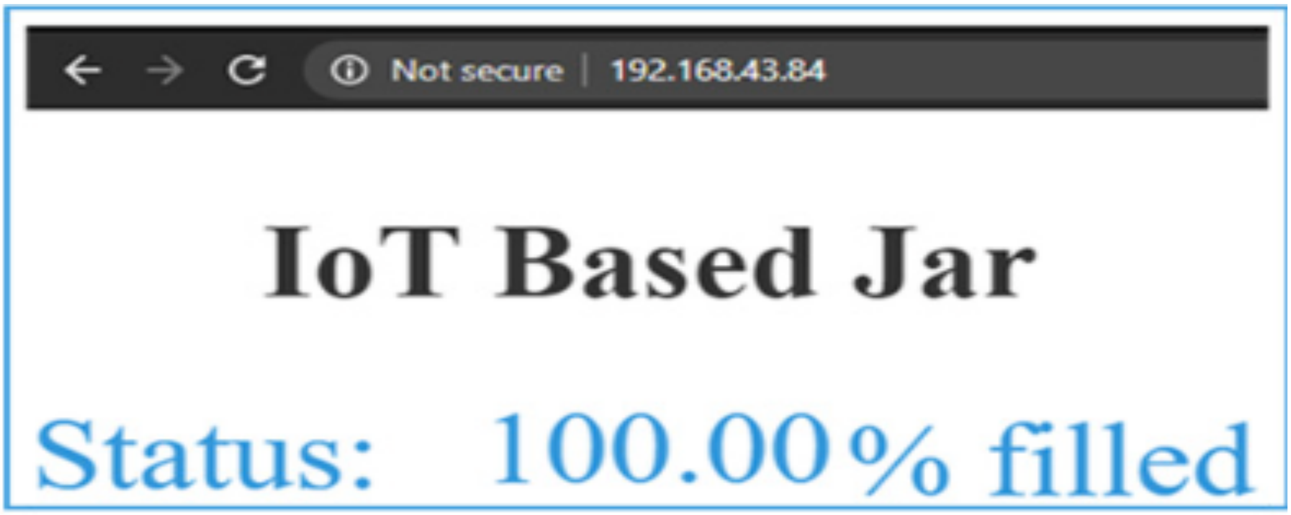
```
Successfully connected to WiFi.
IP address is: 192.168.31.107
Server started
Distance: 0 cm
Level: 100.00 %
```



Not secure 192.168.31.107

IoT Based Jar

Status: 48.57 % filled

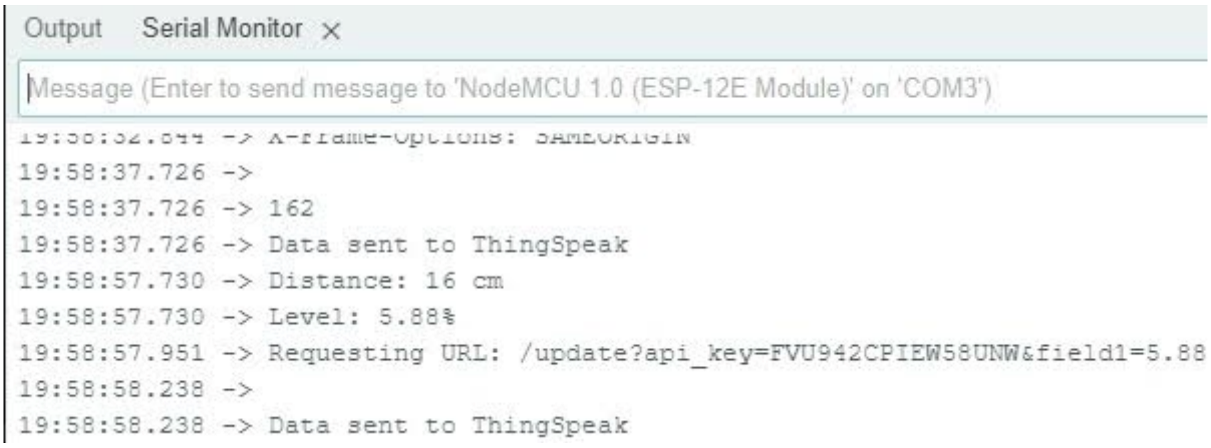


Not secure 192.168.43.84

IoT Based Jar

Status: 100.00 % filled

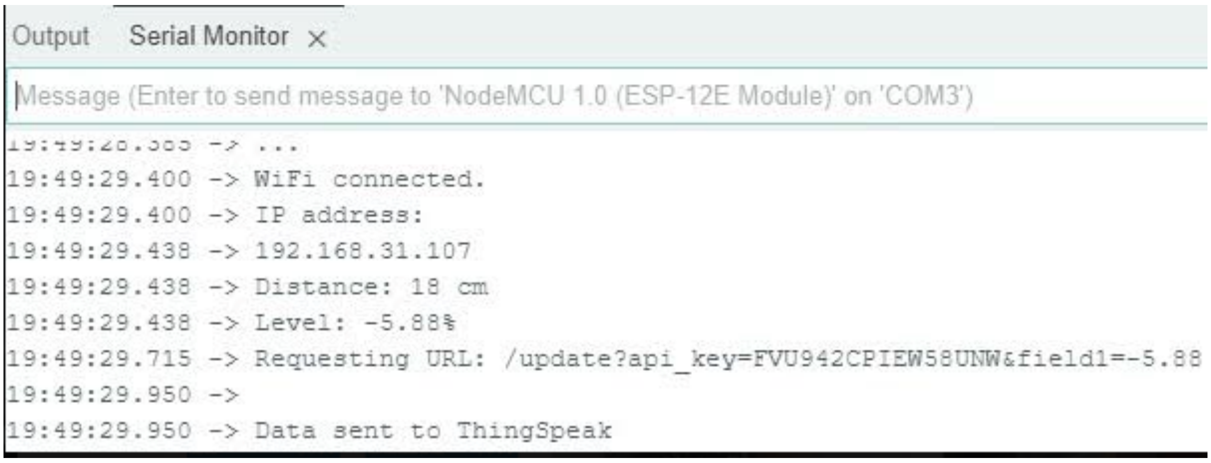
Connecting with Thing speak



Output   Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM3')

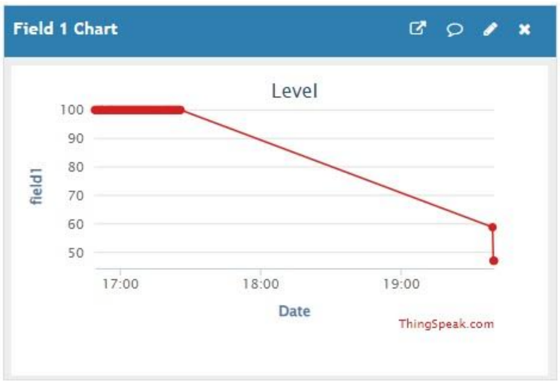
```
19:58:32.844 -> X-Frame-Options: SAMEORIGIN
19:58:37.726 ->
19:58:37.726 -> 162
19:58:37.726 -> Data sent to ThingSpeak
19:58:57.730 -> Distance: 16 cm
19:58:57.730 -> Level: 5.88%
19:58:57.951 -> Requesting URL: /update?api_key=FVU942CPIEW58UNW&field1=5.88
19:58:58.238 ->
19:58:58.238 -> Data sent to ThingSpeak
```

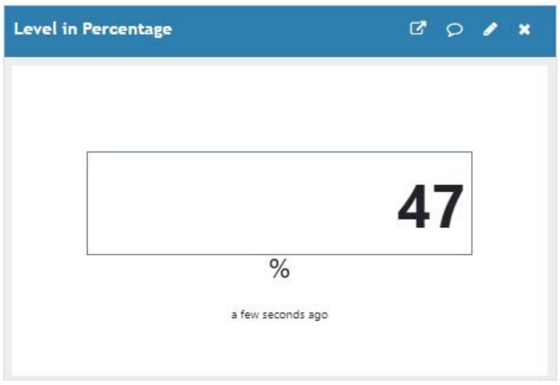


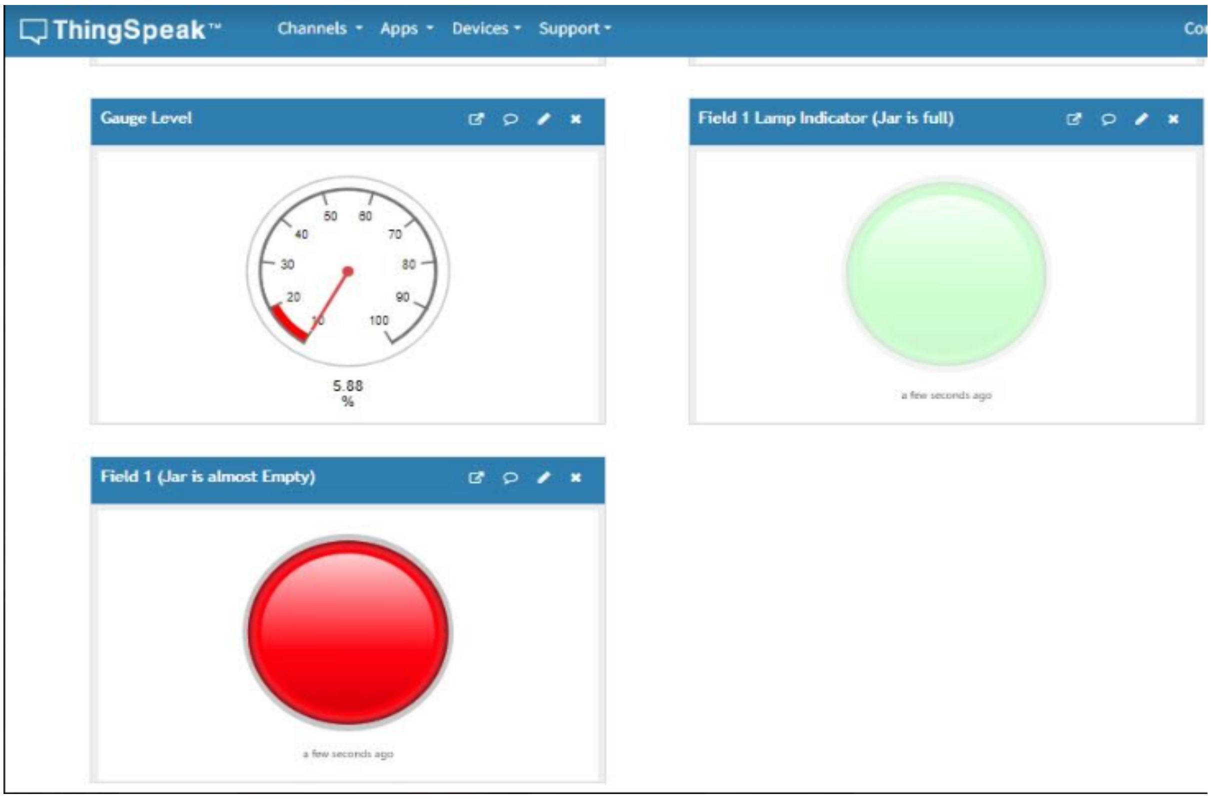
Output   Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM3')

```
19:49:28.383 -> ...
19:49:29.400 -> WiFi connected.
19:49:29.400 -> IP address:
19:49:29.438 -> 192.168.31.107
19:49:29.438 -> Distance: 18 cm
19:49:29.438 -> Level: -5.88%
19:49:29.715 -> Requesting URL: /update?api_key=FVU942CPIEW58UNW&field1=-5.88
19:49:29.950 ->
19:49:29.950 -> Data sent to ThingSpeak
```







Use cases of IoT based Smart Jar using NodeMCU ESP8266 and Ultrasonic Sensor

1. Household Inventory Management

- **Problem**: Keeping track of food items like rice, lentils, sugar, etc., stored in jars can be difficult, especially in large households.
- **Solution**: The Smart Jar continuously monitors the levels of stored consumables and notifies the user when levels are low. This eliminates the need for manual checking and helps ensure that the household stock is always replenished on time.
- **Use Case**: When the jar of rice falls below 20% capacity, the system automatically sends a notification to the user's smartphone via the ThingSpeak platform, reminding them to restock.

2. Kitchen Automation for Restaurants

- **Problem**: In busy restaurant kitchens, managing large quantities of ingredients can be challenging. Running out of critical ingredients during service hours can disrupt operations.

- **Solution**: Smart Jars can monitor ingredient levels and alert kitchen staff when specific items (e.g., spices, grains) are running low, ensuring that key ingredients are always available without the need for manual inventory checks.
- **Use Case**: A Smart Jar storing flour in a bakery sends a notification to the chef's phone when the flour level drops to 10%, prompting them to order more before it runs out.

3. Automated Replenishment for Grocery Stores

- **Problem**: Managing stock levels for bulk items in a grocery store, such as grains, cereals, or pulses, often requires frequent manual checks, which can be time-consuming and prone to errors.
- **Solution**: Smart Jars deployed in bulk storage bins continuously monitor the stock levels. When a bin's level drops below a predefined threshold, the system can notify the store's inventory management system or staff to restock the items.
- **Use Case**: A Smart Jar monitoring the rice bin sends an automatic update to the store's inventory system when it drops below 15%, triggering a restock request.

4. Industrial Resource Monitoring

- **Problem**: In manufacturing or industrial settings, managing consumable materials like lubricants, chemicals, or raw materials requires constant monitoring. Running out of materials can lead to production delays and increased downtime.
- **Solution**: A Smart Jar can be used to monitor critical materials, ensuring that industrial processes always have the necessary resources. The system provides alerts before materials run out, allowing for timely refills and reduced downtime.
- **Use Case**: A Smart Jar monitors the lubricant levels in an industrial machine. When the lubricant level falls to 30%, an alert is sent to the maintenance team for a refill before the machine runs dry.

5. Pharmaceutical Inventory Management

- **Problem**: Pharmacies and medical stores often stock large quantities of medicinal powders or liquids that must be carefully managed to avoid running out or overstocking.
- **Solution**: Smart Jars can be used to track the levels of powders or liquids in real-time, reducing the likelihood of stock shortages. Notifications help pharmacists know when to reorder stock before running low.

- **Use Case**: A pharmacy uses a Smart Jar to monitor a container of a critical powdered medication. When the container drops to 25%, a reorder request is triggered automatically.

6. Supply Chain and Logistics Management

- **Problem**: In logistics and supply chain management, tracking stock levels of materials during transportation or at distribution centers can be difficult, especially for goods stored in large containers.
- **Solution**: The Smart Jar concept can be scaled to monitor containers in distribution centers or during transport. This allows businesses to track stock levels in real-time, improving supply chain efficiency.
- **Use Case**: A Smart Jar monitors the content of a shipment container in a distribution warehouse. When the container reaches 80% depletion, an automatic reorder request is sent to the supplier to ensure the warehouse stays stocked.

Advantages:

1.Real-Time Monitoring:

The Smart Jar provides real-time tracking of content levels. Users can monitor the status of the jar remotely from anywhere via the ThingSpeak platform or any other IoT dashboard, ensuring they always know how much material is left without manual checks.

2. Automation of Stock Management:

Automating the process of checking and refilling jars reduces the need for manual intervention. This is especially useful in environments where multiple containers need monitoring, such as kitchens, industries, or laboratories.

3.Cost-Effective Solution:

The project uses relatively inexpensive components like the NodeMCU ESP8266 and ultrasonic sensors, making it an affordable option for personal and small-scale commercial use.

4. Scalability:

The system can easily be scaled to monitor multiple jars or containers at once. Each jar can have its own dedicated sensor and ThingSpeak channel, allowing users to manage a larger inventory efficiently.

5. Remote Access:

The ThingSpeak platform allows users to access data from anywhere via the internet. Whether users are in the same building or a different location, they can monitor the contents of their Smart Jars with ease.

6.Energy Efficient:

The NodeMCU ESP8266 is designed to be energy-efficient, consuming low power. This allows the Smart Jar to be operational for extended periods without the need for frequent recharging or replacing of power sources.

Disadvantages

1.Limited to Specific Materials:

The ultrasonic sensor used in this project might not be suitable for all types of materials. It works best with dry grains, powders, and liquids but may struggle with irregularly shaped items or materials that absorb sound waves.

2.Internet Dependency:

Since the system relies on IoT platforms like ThingSpeak for data transmission and monitoring, it requires a stable internet connection. Any disruption in Wi-Fi connectivity can lead to a failure in receiving real-time updates or alerts.

3.Power Requirements:

Although the NodeMCU ESP8266 is energy-efficient, it still requires a consistent power source. In battery-operated setups, there may be a need for regular charging or battery replacement, which could be inconvenient.

4.Accuracy Limitations:

The accuracy of the ultrasonic sensor might vary based on environmental conditions (e.g., temperature, humidity) and the shape or arrangement of the jar's contents. For highly precise measurements, more sophisticated (and expensive) sensors may be required.

5.Limited Range of Detection:

Ultrasonic sensors have a limited detection range (usually up to a few meters). If the jar is too tall or too deep, the sensor may not be able to measure the content levels accurately.

6.Data Overload for Small Applications:

For smaller, personal applications (like a household kitchen), the constant flow of data and notifications could feel excessive. Some users may prefer a simpler manual approach rather than being inundated with data about everyday items.

Conclusion

The IoT-based Smart Jar project successfully demonstrates the use of IoT technology for real-time monitoring of container levels. By combining a NodeMCU ESP8266 with an ultrasonic sensor and ThingSpeak, the system automates the task of monitoring the jar's contents, providing users with valuable data in a user-friendly format. Alerts for low and empty jar conditions help optimize resource management, ensuring timely refills and reducing manual checks. This system can be expanded for multiple jars or containers, making it scalable for both personal and industrial applications. Overall, this project highlights the potential of IoT in transforming simple tasks into automated, efficient processes.