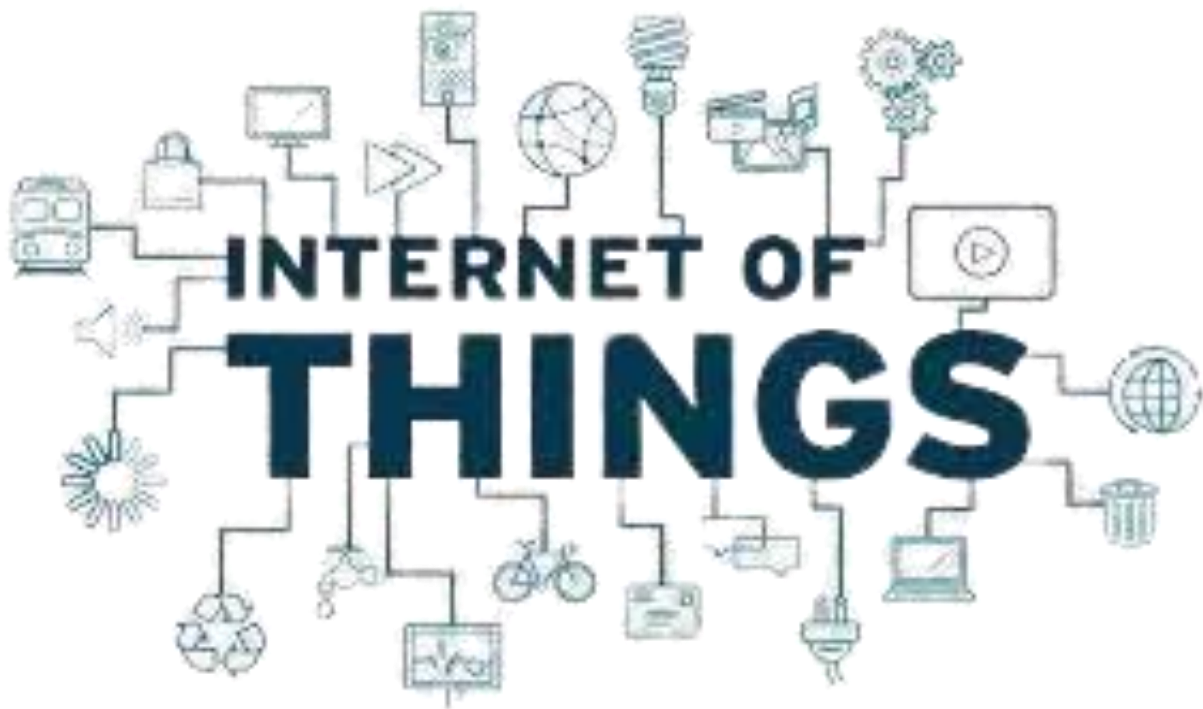


# INTERNET OF THINGS (IOT)

## SMART WATER MANAGEMENT:

## WATER QUALITY MONITORING PUBLIC SWIMMING POOL:



## PROJECT REPORT PHASE 4

SUBMITTED BY

**TEAM LEADER: MOAMMED JAFFER K**

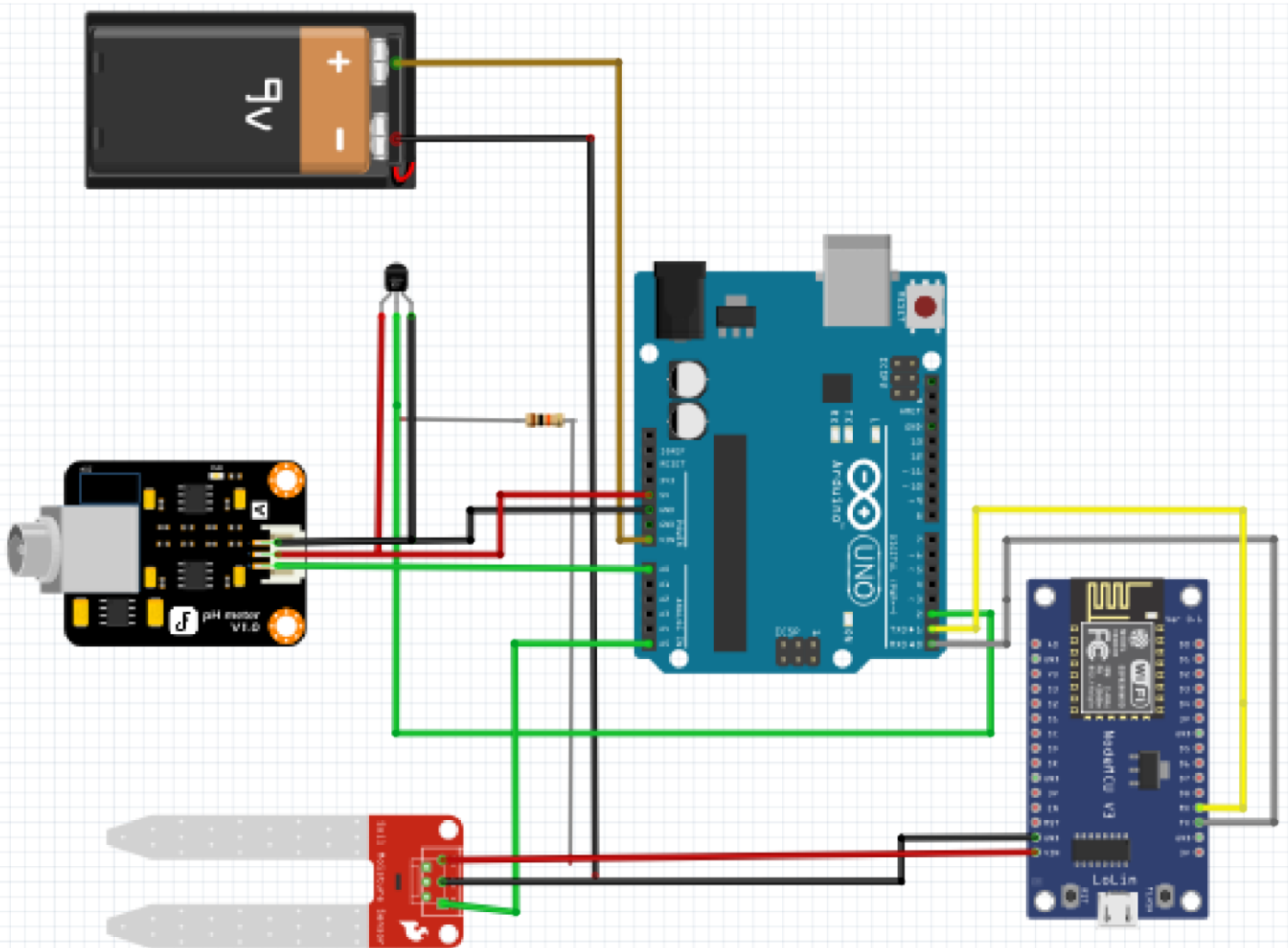
TEAM MEMBERS: HEMA KUMAR S

TEAM MEMBERS: NAVEEN KUMAR V

TEAM MEMBERS: THUTHESH KUMAR R

# CONTINUE BUILDING THE PROJECT BY DEVELOPING

## ◦ CIRCUIT DESIGN:



# PYTHON PROGRAM:

```
#include <OneWire.h>

#include <DallasTemperature.h>

#include <WiFi.h>


const int TdsSensorPin =
34; const int oneWireBus = 4;

const int sensorPin = 35;

const int SensorPin = 32;

#define VREF 5.0 #define SCOUNT
30

float volt; float ntu; unsigned long int avgValue; float b;
int buf[10], temp; int analogBuffer[SCOUNT]; int
analogBufferTemp[SCOUNT]; int analogBufferIndex = 0,
copyIndex = 0; float averageVoltage = 0, tdsValue = 0,
temperature = 25;


OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

WiFiClient client;


String apiWritekey = "7H3WMWDXLOCPBKF";
const char* ssid = "DXMJ"; const char*
password = "dxmdjaffer"; const char*
server = "api.thingspeak.com"; void
setup() {      Serial.begin(115200);
sensors.begin(); delay(1000);


// WiFi setup
WiFi.disconnect(); delay(10);

WiFi.begin(ssid, password);
```

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

```
pinMode(TdsSensorPin, INPUT);
```

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

```
void loop() { // TDS Sensor static unsigned long analogSampleTimepoint
= millis(); if (millis() -
analogSampleTimepoint > 400) { analogSampleTimepoint = millis();
analogBuffer[analogBufferIndex] = analogRead(TdsSensorPin);
analogBufferIndex++; if
(analogBufferIndex == SCOUNT)

analogBufferIndex = 0;

}
```

```
static unsigned long printTimepoint = millis(); if (millis() - printTimepoint > 800) { printTimepoint
= millis(); for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++) analogBufferTemp[copyIndex]
= analogBuffer[copyIndex]; averageVoltage = getMedianNum(analogBufferTemp, SCOUNT) *
(float)VREF / 4095.0; float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0); float
compensationVolatge = averageVoltage / compensationCoefficient;
```

```
tdsValue = (133.42 * compensationVolatge * compensationVolatge * compensationVolatge -
255.86 * compensationVolatge * compensationVolatge + 857.39 * compensationVolatge) * 0.5;
```

```

Serial.print("TDS: ");
Serial.print(tdsValue, 0);
Serial.println("ppm");
}

//          Temperature          Sensor
sensors.requestTemperatures();          float
temperatureC  =  sensors.getTempCByIndex(0);
float temperatureF = sensors.getTempFByIndex(0);
Serial.print("Temp: ");
Serial.print(temperatureC);
Serial.print("°C ---> ");
Serial.print(temperatureF);
Serial.println("°F");

// Turbidity Sensor
volt = 0; for (int i = 0; i < 800; i++) {  volt +=
((float)analogRead(sensorPin) / 4095) * 3.3;
}
volt = volt / 800;  volt = round_to_dp(volt,
1);  volt
= volt + 2.85;

if (volt < 2.5) {  ntu = 3000; } else {  ntu = -1120.4
* (volt * volt) + 5742.3 * volt - 4353.8;  ntu = ntu +
580.56;
}

Serial.println("Turbidity = " + String(ntu) + " NTU");

```

```

// pH Sensor  for (int i = 0; i <
10; i++) {    buf[i] =
analogRead(SensorPin);
delay(10);

}

for (int i = 0; i < 9; i++) {    for
(int j = i + 1; j < 10; j++) {    if
(buf[i] > buf[j]) {        temp =
buf[i];        buf[i] = buf[j];
buf[j] = temp;

    }

}

}

avgValue = 0;  for (int i = 2; i < 8; i++)
avgValue += buf[i];
float pHValue = (float)avgValue * 3.3 / 4095 / 6;  pHValue
= 5 + pHValue;

Serial.print("pH: ");
Serial.print(pHValue, 2);
Serial.println(" ");

// Upload data to ThingSpeak  uploadToThingSpeak();

}

```

```

int getMedianNum(int bArray[], int iFilterLen) {
int bTab[iFilterLen];  for (byte i = 0; i <
iFilterLen; i++)    bTab[i] = bArray[i];  int i, j,
bTemp;  for (j = 0; j < iFilterLen - 1; j++) {    for

```

```

(i = 0; i < iFilterLen - j - 1; i++) {    if (bTab[i] >
bTab[i + 1]) {    bTemp = bTab[i];    bTab[i]
= bTab[i + 1];    bTab[i + 1] = bTemp;

    }

}

}

if ((iFilterLen & 1) > 0)    bTemp = bTab[(iFilterLen - 1) / 2];

else    bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 -

1]) / 2;

return bTemp;

}

```

```

float round_to_dp(float in_value, int decimal_place) {
float multiplier = powf(10.0f, decimal_place); in_value
= roundf(in_value * multiplier) / multiplier; return
in_value;
}

```

```

void uploadToThingSpeak() {

    String url = "/update";

    String headers = "X-THINGSPEAKAPIKEY: " + apiWritekey + "\r\n";
    String dataPayload = "field1=" + String(123) + "&field2=" + String(7.2)
        + "&field4=" + String(25.5) + "&field5=" + String(150);

    int contentLength = dataPayload.length();

    String request = "POST " + url + " HTTP/1.1\r\n" +
        "Host: " + server + "\r\n" +
        "Connection: close\r\n" +

```

```

        "Content-Type: application/x-www-form-urlencoded\r\n" +
"Content-Length: " + String(contentLength) + "\r\n" +
        headers +
        "\r\n" + dataPayload + "\r\n\r\n";

client.print(request);

// Handle the response from ThingSpeak if needed
while (client.available()) {
    String line = client.readStringUntil('\r\n');
    // Process the response as needed (e.g., check for success or error messages)
    // You can print the response to Serial for debugging purposes    Serial.println(line);
}

// Close the connection    client.stop();
}

```



# ARDUINO:

Arduino is an open-source electronics platform and ecosystem that provides both hardware and software tools for building a wide range of electronic projects. It was created to make it easier for people, particularly hobbyists, students, and makers, to develop and prototype their own electronic devices and interactive projects. Arduino is widely popular due to its accessibility and flexibility.

Here are some key components and features of the Arduino platform:

1.Arduino Boards: Arduino boards are the hardware foundation of the platform. They come in various models and sizes, but they typically consist of a microcontroller (e.g., ATmega series) and input/output pins. The Arduino Uno is one of the most well-known and widely used boards.

2.Arduino Software (IDE): The Arduino Integrated Development Environment (IDE) is the software used for programming Arduino boards. It is a user-friendly environment that supports the Arduino programming language, which is a simplified version of C/C++. Users can write and upload code to their Arduino boards via the IDE.

3.Arduino Shields: Shields are add-on boards that can be stacked on top of Arduino boards to extend their capabilities. There are various shields available for functions like Ethernet connectivity, wireless communication, motor control, and sensor interfacing.

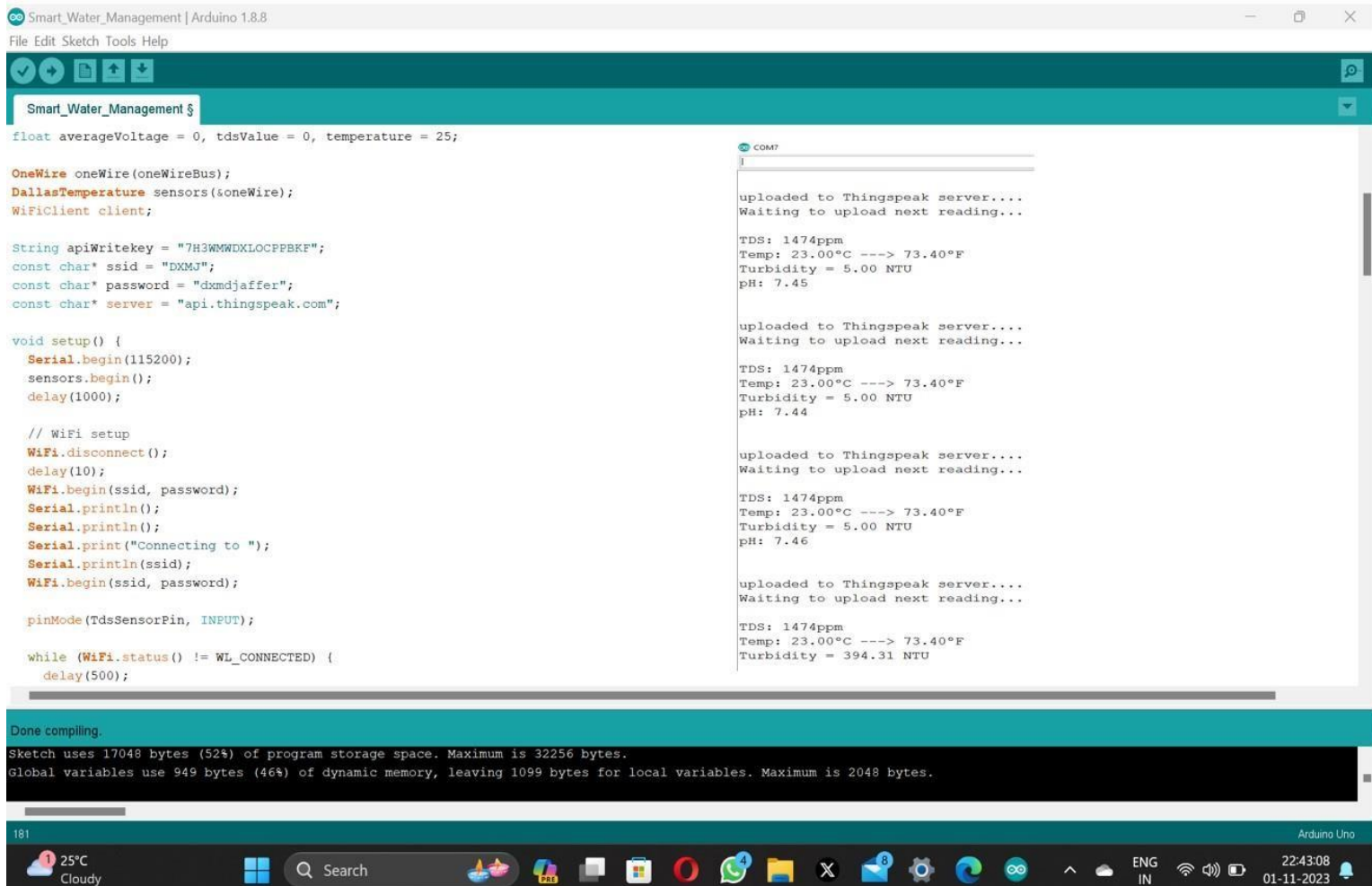
4.Arduino Libraries: Arduino provides a collection of libraries that simplify working with various components and modules. These libraries contain pre-written code that can be easily integrated into your projects.

5.Versatile Inputs and Outputs: Arduino boards have digital and analog input/output pins, making it easy to interface with sensors, actuators, and other electronic components. This flexibility allows for a wide range of applications.

6.Community and Documentation: Arduino has a large and active community of users, makers, and developers. There are numerous online resources, tutorials, and forums where users can seek help, share their projects, and find inspiration.

7.Open Source: Arduino is open source, which means that the hardware and software designs are available to the public. This encourages innovation and allows users to modify and adapt the platform to their specific needs.

# ARDUINO OUTPUT:



The screenshot displays the Arduino IDE interface. The main editor window shows the 'Smart\_Water\_Management' sketch with the following code:

```
float averageVoltage = 0, tdsValue = 0, temperature = 25;

OneWire oneWire(oneWireBus);
DallasTemperature sensors(&oneWire);
WiFiClient client;

String apiWritekey = "7H3WMWDXLOCPPBRK";
const char* ssid = "DXMJ";
const char* password = "dxmdjaffer";
const char* server = "api.thingspeak.com";

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

  // WiFi setup
  WiFi.disconnect();
  delay(10);
  WiFi.begin(ssid, password);
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  pinMode(TdsSensorPin, INPUT);

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

The serial monitor (COM7) shows the output of the sketch, which is uploading data to the Thingspeak server. The output is as follows:

```
uploaded to Thingspeak server...
Waiting to upload next reading...

TDS: 1474ppm
Temp: 23.00°C ---> 73.40°F
Turbidity = 5.00 NTU
pH: 7.45

uploaded to Thingspeak server...
Waiting to upload next reading...

TDS: 1474ppm
Temp: 23.00°C ---> 73.40°F
Turbidity = 5.00 NTU
pH: 7.44

uploaded to Thingspeak server...
Waiting to upload next reading...

TDS: 1474ppm
Temp: 23.00°C ---> 73.40°F
Turbidity = 5.00 NTU
pH: 7.46

uploaded to Thingspeak server...
Waiting to upload next reading...

TDS: 1474ppm
Temp: 23.00°C ---> 73.40°F
Turbidity = 394.31 NTU
```

The status bar at the bottom indicates that the sketch is compiled and ready to be uploaded to an Arduino Uno. The status bar also shows the current temperature (25°C) and the date (01-11-2023).

# THINGSPEAK:

ThingSpeak is an Internet of Things (IoT) platform and open-source application for collecting, processing, and visualizing data from various IoT devices. It was created by MathWorks, the company behind MATLAB, and is designed to simplify the process of connecting and working with IoT devices and data. ThingSpeak provides several key features:

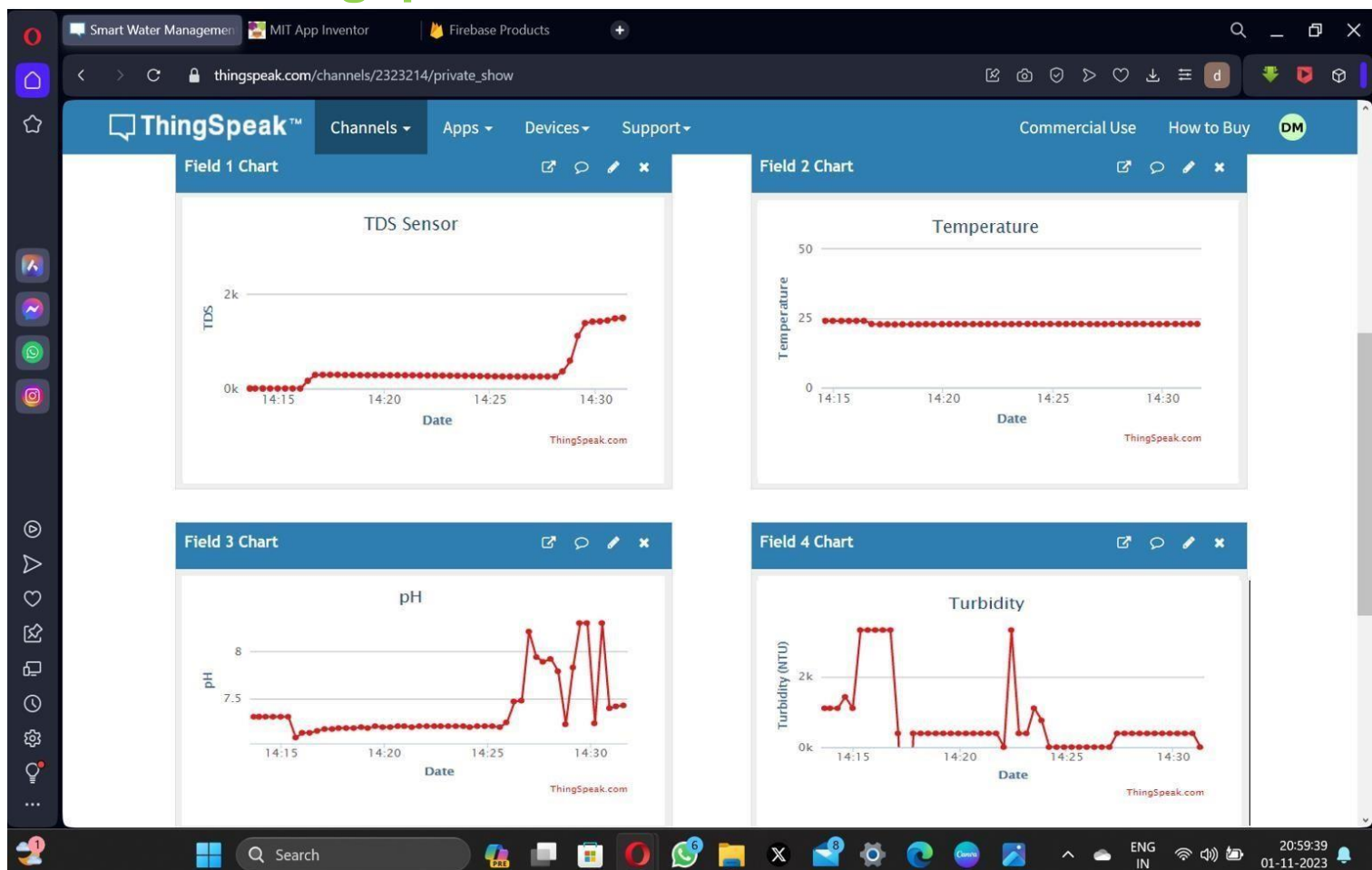
- 1.Data Collection: ThingSpeak allows users to collect data from a variety of sources, such as sensors, devices, and applications. This data can be sent to ThingSpeak using various communication protocols, including HTTP, MQTT, and the ThingSpeak API.
- 2.Data Processing: The platform provides built-in MATLAB analytics that can be applied to the collected data. This enables users to perform real-time data processing, analysis, and visualization directly on the platform.

3.Data Visualization: ThingSpeak offers customizable charts, graphs, and visualizations to help users understand their IoT data. Users can create real-time dashboards to monitor their devices and sensors.

4.Data Sharing: Users can choose to make their data public or keep it private. This feature is useful for collaborative projects, research, and sharing data with others.

5.Integration: ThingSpeak can be integrated with other IoT platforms, cloud services, and applications. It provides support for third-party services and devices, making it easier to connect your IoT ecosystem.

## ◦ Thingspeak simulation:



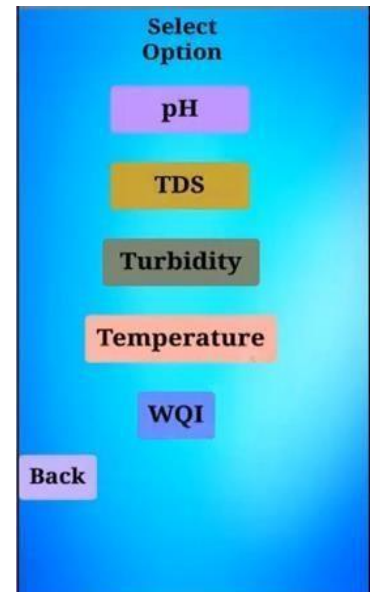
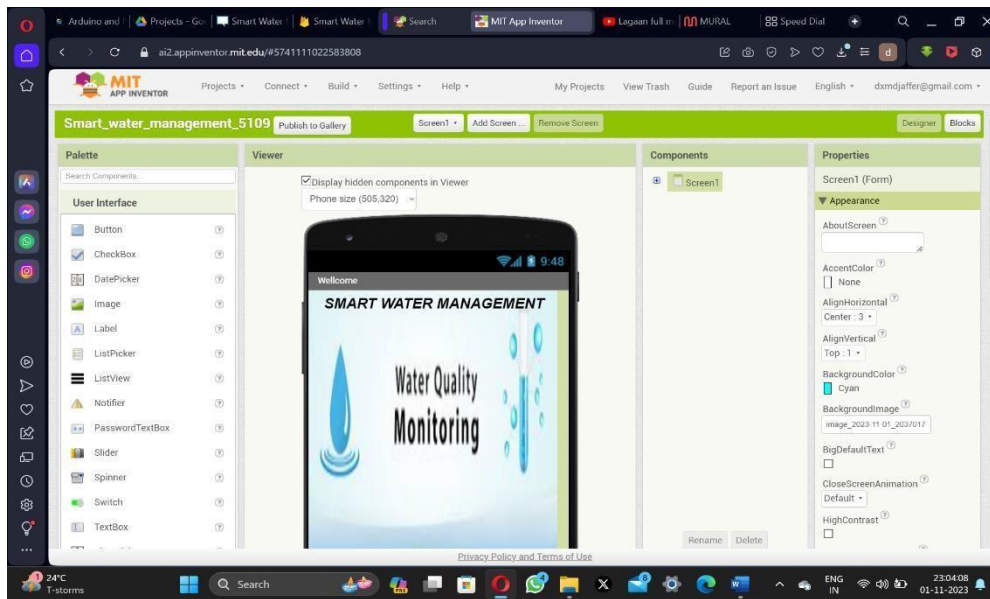
# MIT APP INVENTOR:

MIT App Inventor is an open-source visual development platform that allows people, including those with little to no programming experience, to create mobile applications for Android devices. It was initially created by Google and then transferred to the Massachusetts Institute of Technology (MIT) for further development and maintenance. MIT App Inventor uses a block-based programming approach that simplifies app development, making it accessible to a wide range of users.

Here are some key features and aspects of MIT App Inventor:

1. **Visual Programming:** MIT App Inventor uses a visual, drag-and-drop approach to app development. Instead of writing code, users assemble blocks representing different app components and actions. This makes it easy for beginners to create functional apps without having to write traditional code.
2. **Real-Time Testing:** Users can test their apps on Android devices in real-time by using the MIT App Inventor Companion app. This allows for quick and iterative development, and changes made in the development environment are immediately reflected on the connected Android device.
3. **Extensive Component Library:** MIT App Inventor includes a wide range of components that can be used to build apps. These components cover UI elements, sensors (such as GPS and accelerometer), multimedia, data storage, and communication features.
4. **Cloud-Based Development:** MIT App Inventor is a cloud-based platform, which means that users can access their projects from anywhere with an internet connection. This also simplifies collaboration and sharing of projects.
5. **Custom Blocks:** While the platform offers a wide range of pre-built blocks, users can also create custom blocks, allowing for the encapsulation of complex logic into reusable components.
6. **Educational Focus:** MIT App Inventor is often used in educational settings to teach programming and app development. It's designed to be user-friendly and intuitive, making it a great tool for introducing students to the world of app creation.
7. **Open Source:** MIT App Inventor is an open-source project, which means that the source code is available to the public. This openness allows for the platform to be extended and modified by the community.

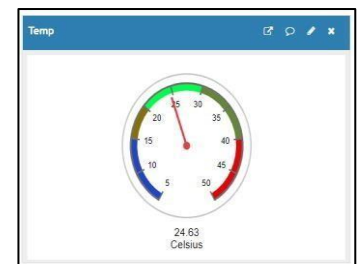
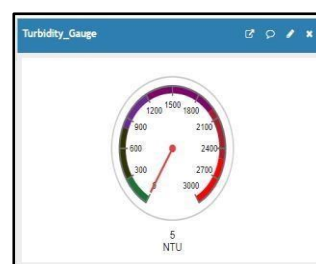
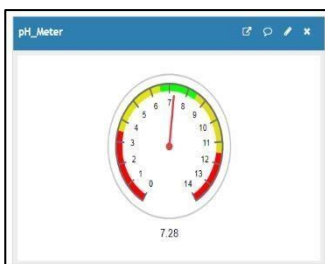
## Mit app inventor simulation:



## Mobile app design in mit app inventor



## Select option



## **CONCLUSION:**

**The system is versatile and economical. The real time IoT based water quality monitoring system enables to monitor the parameters of water i.e.**

**pH, TDS, turbidity and temperature in real automatically. This system reduces the cost and time, allow authorities to take decisions timely.**