NANYANG POLYTECHNIC
EG431D Data Acquisition

Lesson 3

Course: Diploma in Engineering (Electronic and Digital Engineering)

Module: EG431D Data Acquisition

Title: Data Visualization

Objective:

This lesson introduces learners to the fundamentals of Data Visualization, where data sets are transformed into clear and understandable visual formats to facilitate the identification of patterns, trends, and insights. Learners will gain proficiency with various visualization platforms and develop the ability to select the most appropriate platform based on specific project requirements, enhancing their capacity to communicate data-driven insights effectively.

Learning Objectives:

- Understand the core principles of data visualization and its importance in data analysis.
- Describe the process of real-time data visualization using edge devices, including data collection, processing, and transmission to cloud dashboards according to industry practices.
- Explain the current trends in data visualization technologies, focusing on the role of edge computing and cloud dashboards in industrial IoT and remote monitoring systems.
- Demonstrate how cloud-based platforms allow developers to visualize and analyze data from edge devices, enabling the creation of scalable, real-time dashboards for monitoring and decision-making.
- Demonstrate the ability to transform raw data into meaningful visual formats.
- Demonstrate the skills to select and apply the most suitable visualization method based on data characteristics and project requirements.

1. Overview

In the context of using a Microcontroller Unit (MCU) as a data acquisition and processing system, data visualization plays a critical role in effectively interpreting and communicating real-time sensor data. As MCUs collect data from various sensors—such as temperature, pressure, humidity, and motion—visualizing this data enables users to quickly understand and monitor system performance, environmental conditions, or operational status. With visual representation, raw sensor data is transformed into charts, graphs, or dashboards, allowing users to spot trends, identify anomalies, and make rapid, informed decisions.

For instance, in an industrial setting, an MCU may gather data from multiple sensors to monitor machinery health. By visualizing this data, maintenance teams can easily track performance trends, detect potential issues, and take preventive action before a failure occurs. Similarly, in environmental monitoring applications, MCUs can capture air quality, temperature, and humidity levels, which can be visualized to observe changes over time and respond to environmental shifts.

Data visualization is essential for managing large-scale or complex data streams, as it distills information into an intuitive format, making it accessible to users without technical expertise. Furthermore, by displaying real-time data, visualizations support immediate feedback and system control, allowing users to adjust parameters or respond to changes instantly. In summary, data visualization complements MCU-based data acquisition by enhancing data interpretation, improving situational awareness, and facilitating swift, data-informed decision-making.

In this lesson, we will explore and become familiar with Arduino IDE built-in data visualization tools, on-the-edge data visualization, Cloud Dashboards and SCADA.

2. Arduino IDE built-in Data Visualization Tools

The Arduino Integrated Development Environment (IDE) includes built-in data visualization tools that allow users to observe real-time data output from sensors and other components connected to Arduino boards. Key tools include:

- **Serial Monitor:** The Serial Monitor (or Serial Terminal) displays real-time text-based data from the Arduino, providing a straightforward way to monitor sensor readings, debug information, or any other output the Arduino generates. It’s an essential tool for developers to verify that their code is functioning as expected and to observe live data updates.
- **Serial Plotter:** The Serial Plotter offers a graphical view of data in real time, converting numerical data into line graphs. This tool is particularly useful for visualizing trends, patterns, and changes over time, making it easy to interpret data from sensors like temperature, humidity, or accelerometers. With the Serial Plotter, users can view multiple data sets simultaneously, each represented by a different line on the graph, allowing for comparative analysis and a more comprehensive understanding of system behavior.

These built-in tools in the Arduino IDE make it easier for users to interpret and analyze data from their projects without needing additional software, enhancing the development process by providing immediate feedback on the Arduino’s data output.

2.1 Serial Monitor

To demonstrate the use of the Serial Monitor, connect the Arduino MKR WiFi 1010 board along with the MKR Connector Carrier Board and attach the AMG8833 Thermopile Sensor Array (TSA) module as described in Lesson 2. Then, open the Arduino IDE and load *tempTSA_ST.ino* sketch.

Compile and upload the sketch to the Arduino MKR WiFi 1010 board. Once the sketch is running, go to the **Tools** menu in the Arduino IDE and select **Serial Monitor** to open the monitor window. You should see an array of temperature values displayed in real-time on the Serial Monitor (Figure 2A), representing temperature readings from each pixel in the AMG8833 sensor array. This setup allows you to observe and analyze temperature data directly through the Serial Monitor.

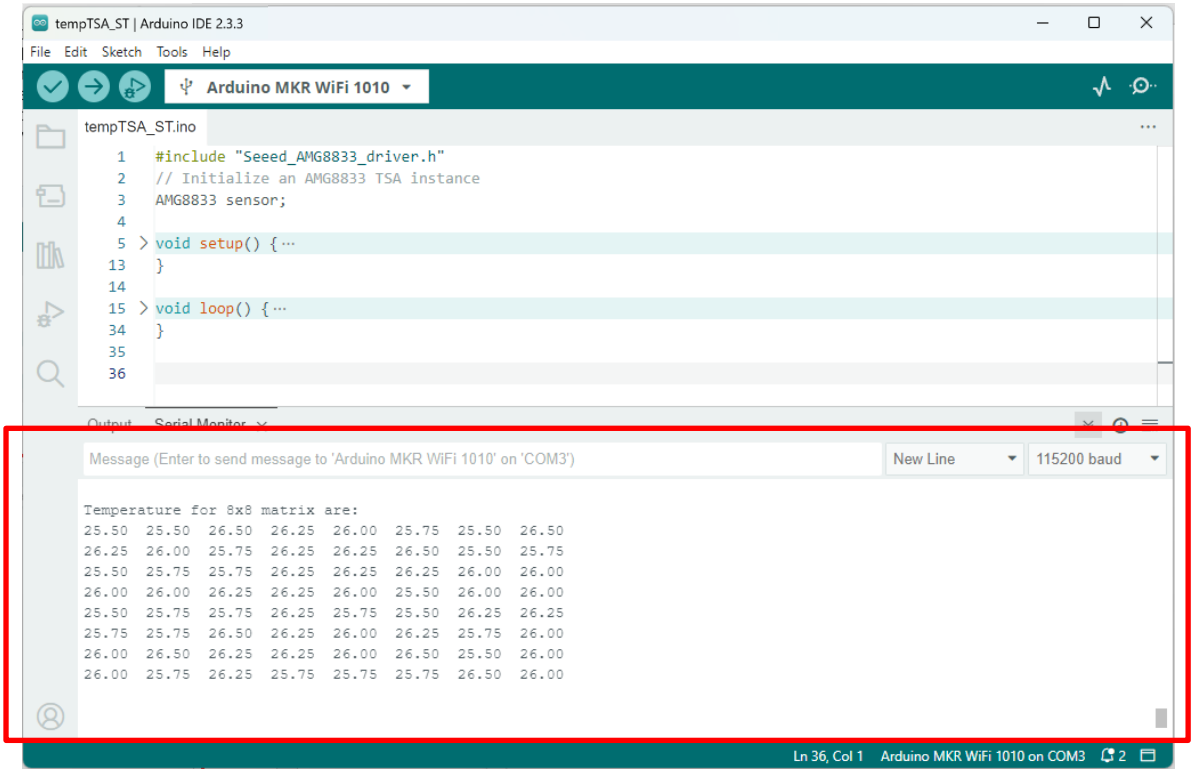

Figure 2A – An Arduino IDE with a Serial Monitor window displaying temperature readings for each pixel in the AMG8833 TSA.

The temperature values displayed in Figure 2A are neatly formatted in the Serial Monitor window, thanks to the use of a combination of *for* loops and *Serial.print()* commands, as illustrated in Listing 2A. This approach allows the programmer to control how the data appears, creating a structured and readable output. By iterating through each temperature value in the AMG8833 TSA array with *for* loops and carefully using *Serial.print()* to format each value, the programmer can adjust the arrangement, spacing, and readability of the data.

This flexibility in formatting ensures that the data visualization in the Serial Monitor meets specific readability and layout preferences, making it easier to interpret and analyze the temperature array.

tempTSA_ST.ino

```
#include "Seeed_AMG8833_driver.h"
// Initialize an AMG8833 TSA instance
AMG8833 sensor;

void setup() {···
}

void loop() {
    u8 val = 0;
    float pixels_temp[PIXEL_NUM] = {0};

    // Read temperature of all the 64 pixels
    sensor.read_pixel_temperature(pixels_temp);
    // Print 8x8 pixels temperature value in 8 rows by 8 columns
    Serial.println("Temperature for 8x8 matrix are:");

    for (int i = 0; i < PIXEL_NUM; i++) {
        Serial.print(pixels_temp[i]);
        Serial.print("  ");
        if (0 == (i + 1) % 8) {
            Serial.println(" ");
        }
    }
    Serial.println(" ");
    Serial.println(" ");
    delay(500);
}
```

Listing 2A – Codes for printing temperature results to Serial Monitor window

2.2 Serial Plotter

To demonstrate the use of the Serial Plotter, connect the Arduino MKR WiFi 1010 board with the MKR Connector Carrier Board and attach the Grove – Digital Light Sensor, as described in Lesson 2. Open the Arduino IDE, load *lightSensorIR_SP.ino* sketch, compile and upload the sketch to the Arduino board. Once the sketch is running, go to the **Tools** menu in the Arduino IDE and select **Serial Plotter** to open the plotter window.

In the Serial Plotter window, you should see two line plots displayed in real-time, as shown in Figure 2B, representing the infrared (IR) and full spectrum light intensity readings from the sensor. Each light intensity is visualized with a distinct line color, making it easy to observe and compare changes in IR and full spectrum light levels over time. This simultaneous visualization of multiple variables allows for more intuitive data interpretation and is especially beneficial for applications that require continuous monitoring and real-time analysis.

For the Serial Plotter to correctly display multiple plots, the sketch must output multiple data points separated by a delimiter, such as a comma or space. This can be achieved by using a series of *Serial.print()* functions for each data point (and delimiter), followed by a *Serial.println()* for the last data point and move to the next line for each new set of readings. For instance, if you are plotting IR and full spectrum light intensities, the sketch (Listing 2B) should use *Serial.print()* to output the first value, add a comma or space, and *Serial.println()* for the second value and move to the next line. This format enables the Serial Plotter to interpret and display

each data point as a separate line plot, allowing for clear, real-time visualization of multiple variables.

Figure 2B – Serial Plotter window showing real-time light intensities plots.

lightSensorIR_SP.ino

```
1   #include <Wire.h>
2   #include <Digital_Light_TSL2561.h>
3
4   void setup() {
5       Wire.begin();
6       Serial.begin(9600);
7       TSL2561.init();
8   }
9
10  void loop() {
11      // read Infrared channel raw intensity value only, not converted to Lux
12      Serial.print(TSL2561.readIRLuminosity());
13      Serial.print(" ");
14      // read Full Spectrum channel raw intensity value only, not converted to Lux
15      Serial.println(TSL2561.readFSpecLuminosity());
16      delay(1000);
17  }
```

Delimiter

Listing 2B – Codes for generating proper data format for Serial Plotter

3 On-The-Edge Data Visualization

Arduino Boards support a variety of output modules that can be used for basic, localized data visualization. These modules include components like LEDs, OLED displays, LCD screens, and seven-segment displays allowing for simple visual feedback directly on the board or nearby. Such visualizations are ideal for providing

immediate insights without the need for external tools or interfaces. For example, LEDs can indicate system status or thresholds, an LCD can display sensor readings, and an OLED screen can show multiple metrics at once. While limited in scope compared to more advanced visualization tools, these output modules offer an accessible and efficient way to monitor data and system states in real-time, especially useful for quick checks, debugging, and compact projects.

3.1 LED Indicator

An LED indicator is one of the simplest and most widely used forms of data visualization hardware. It can provide immediate visual feedback, such as signaling power status, operational states, or threshold alerts. Controlled by an output port on any MCU, an LED can be programmed to turn on, off, or blink at specific intervals to convey different types of information. For instance, a solid light might indicate normal operation, while a blinking LED could signal an error or specific event. Due to its simplicity, low power consumption, and ease of integration, the LED is a practical tool for quick, localized data visualization in a wide range of applications.

In this lesson, Dual LED Indicators and Buzzer module (Figure 3A), which includes LED1, LED2, and an unconnected Buzzer, is used to set up a real-time ambient temperature monitoring system. LED1 and LED2 are connected to Port D5 and D6 of the Arduino MKR WiFi 1010 board via the MKR Connector Carrier Board. A Grove – Temperature Sensor is attached to Port A1 to capture the surrounding temperature. This setup enables the system to monitor temperature in real-time, using the LEDs to provide visual feedback based on temperature thresholds. The combination of temperature sensing and LED indicators offers a straightforward way to observe temperature changes and alerts, while the Buzzer can be added later for audio notifications if required.

Figure 3A – Dual LED Indicators and Buzzer module

The *tempSensorLED.ino* sketch (Listing 3A) integrates a Grove - Temperature Sensor and two LEDs to create a simple temperature monitoring and alert system. The temperature sensor is connected to Port A1, and LEDs are connected to Ports D5 and D6. The sketch reads analog data from the temperature sensor and calculates the resistance value of the thermistor, which is then converted into temperature using the Steinhart–Hart equation. The calculated temperature is printed on the Arduino IDE's Serial Monitor. When the temperature is below the threshold, LED1 and LED2 remain OFF. If the temperature exceeds a predefined threshold (TEMP_TH), LED1 blinks to indicate the threshold breach, and LED2 turns ON as a persistent warning. When the temperature is again below the threshold, LED1 turns OFF, and LED2 remains ON. The sketch uses a 500ms delay to manage the blinking behavior of LED1, providing real-time visual feedback based on the temperature readings.

tempSensorLED.ino

```ino
#include <math.h>
#define TEMP_SENSOR  A1       // Grove - Temperature Sensor connects to Analog Port A1
#define LED1         5        // LED1 connects to Digital Port D5 (LED is Active LOW)
#define LED2         6        // LED2 connects to Digital Port D6 (LED is Active LOW)
#define TEMP_TH      27       // Temperature Threshold
const int B  = 4275;          // B value of the thermistor (from Datasheet)
const int R0 = 100000;        // R0 = 100k
bool _led1 = 1;               // LED1 State (1 = LED OFF, 0 = LED ON)
bool _led2 = 1;               // LED2 State (1 = LED OFF, 0 = LED ON)

void setup()
{…
}

void loop()
{
    int a = analogRead(TEMP_SENSOR);
    float R = 4095.0/a - 1.0;
    R = R0 * R;

    // converts R to temperature using simplified Steinhart-Hart equation
    float temperature = 1.0/(log(R/R0)/B + 1/298.15) - 273.15;
    Serial.print("temperature = ");
    Serial.println(temperature);

    if (temperature > TEMP_TH)
    {
      _led1 = !(_led1);         // Blink LED1
      _led2 = 0;                // Turns ON LED2
    }
    else
      _led1 = 1;                // Turns OFF LED1

    digitalWrite(LED1, _led1);  // Write _led1 State to Port D5
    digitalWrite(LED2, _led2);  // Write _led2 State to Port D6
    delay(500);
}
```

Listing 3A – A Sketch to measure ambient temperature, displays temperature value to the Serial Terminal, and lights up LEDs when temperature threshold exceeded.

3.2 Alpha-Numeric LCD

An alphanumeric LCD is another widely used form of data visualization hardware, commonly employed in projects requiring clear, text-based feedback. This type of display allows for the presentation of both letters and numbers, making it ideal for showing sensor readings, status messages, and simple data outputs in real-time.

Alphanumeric LCDs, often available in configurations like 16x2 or 20x4 (indicating the number of characters per line and lines available), are easy to interface with microcontrollers, such as Arduino Board, via parallel or I2C connections. This makes them highly versatile and suitable for applications where concise information needs to be displayed locally, such as temperature values, system status, or user prompts. Due to their readability, low power consumption, and affordability, alphanumeric LCDs are popular in embedded systems, control panels, and various portable devices.

In this lesson, Grove – LCD RGB Backlight module (Figure 3B) and the MAX30100 Heart Rate and Pulse Oximetry sensor module (details in Lesson 2) are utilized to create a system for real-time monitoring of heart rate and blood oxygen saturation levels. Both the LCD (address 0x30) and the MAX30100 sensor (address 0x57) connect to the TWI (I2C) port on the Arduino MKR WiFi 1010 board via the MKR Connector Carrier Board, ensuring smooth and efficient data communication. Additionally, a Grove – Buzzer is connected to Port D2, which sounds an alert if the heart rate exceeds a specified threshold. The LCD provides continuous visual feedback by displaying the user's current heart rate and oxygen saturation levels, while the Buzzer functions as an alert for high heart rate conditions.

Figure 3B – Grove LCD RGB Backlight

The *heartRateSensorLCD.ino* sketch (Listing 3B) integrates the MAX30100 Pulse Oximeter with an LCD and a Buzzer to create a real-time heart rate and blood oxygen saturation (SpO2) monitoring system. The sketch reads heart rate and SpO2 data from the sensor, displaying the values on the Arduino IDE's Serial Terminal and LCD. The LCD shows heart rate (HR) on the first line and SpO2 on the second line. If the heart rate exceeds a defined threshold (100 bpm), the Buzzer connected to Port D2 is activated to provide an audible alert; otherwise, the Buzzer remains OFF. The system updates data at regular intervals (1 second) using a timer mechanism *(millis())*, ensuring efficient and timely monitoring.

To use the LCD, a custom library is required, which can be downloaded from:

https://github.com/Seeed-Studio/Grove_LCD_RGB_Backlight/archive/master.zip

Detailed information on the ultrasonic sensor can be found at:

https://wiki.seeedstudio.com/Grove-LCD_RGB_Backlight/

heartRateSensorLCD.ino

```
#include <Wire.h>
#include "rgb_lcd.h"
#include "MAX30100_PulseOximeter.h"
#define REPORTING_PERIOD_MS 1000
#define HR_THRES            100   // Heart rate threshold
#define BUZZER              2     // Buzzer connects to Digital Port D2

uint32_t tsLastReport = 0;
// Initialize LCD
rgb_lcd lcd;
// Initialize a Pulse Oximeter instance (beat detection, heart rate, SpO2)
PulseOximeter pox;
// Callback (registered below) fired when a pulse is detected
void onBeatDetected()
{···
}

void setup()
{···
}
void loop()
{
    float hr;
    unsigned char spo2;
    pox.update();
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        hr   = pox.getHeartRate();
        spo2 = pox.getSpO2();
        Serial.print("Heart rate:"); Serial.print(hr);
        Serial.print("bpm / SpO2:"); Serial.print(spo2);
        Serial.println("%");

        lcd.clear();
        lcd.setCursor(0, 0); lcd.print("HR  : "); lcd.print(hr);
        lcd.setCursor(0, 1); lcd.print("SPO2: "); lcd.print(spo2);

        if(hr >= HR_THRES) {digitalWrite(BUZZER, HIGH);}
        else {digitalWrite(BUZZER, LOW);}

        tsLastReport = millis();
    }
}
```

Listing 3B – A Sketch to measure heart rate and SpO2, displays values to LCD and Serial Terminal, and sounds Buzzer when heart rate threshold exceeded.

4 Cloud Dashboards

A Cloud Dashboard is an online interface that provides real-time data visualization and management by connecting IoT devices, sensors, or systems to cloud-based platforms. This dashboard enables users to remotely monitor, analyze, and control processes or devices from any location with internet access. Designed for ease of use and interactivity, cloud dashboards typically display data in graphical formats, such as charts, graphs, and gauges, making it easy to interpret trends, detect anomalies, and make informed decisions. They are widely used across various applications, including industrial automation, smart homes, healthcare, agriculture, and energy management.

Cloud dashboards offer several advantages over traditional on-premises systems. They provide scalability, allowing users to integrate numerous devices and handle large volumes of data without the need for extensive local infrastructure. Additionally, cloud platforms enable advanced features like real-time alerts, data sharing, predictive analytics, and remote-control capabilities. For example, in a smart agriculture setup, a cloud dashboard can display soil moisture, temperature, and humidity levels, while enabling automated irrigation based on real-time data.

In this lesson, we will explore two popular cloud dashboard platforms, **AllThingsTalk** and **ThingSpeak**, for use with Arduino. These platforms enable real-time data visualization, analysis, and management, making them ideal for IoT projects.

AllThingsTalk provides a user-friendly interface for connecting Arduino devices to the cloud, allowing users to visualize sensor data, control actuators remotely, and create automation rules through its intuitive dashboard. It supports protocols like MQTT and LoRaWAN, enabling seamless integration with various IoT ecosystems.

ThingSpeak, on the other hand, focuses on data collection, storage, and analysis. It integrates with MATLAB for advanced analytics and supports visualization of sensor data in graphs, charts, and other formats. It is particularly useful for projects requiring historical data analysis and predictive modeling.

4.1 AllThingsTalk

AllThingsTalk is a cloud-based Internet of Things (IoT) platform that provides tools and infrastructure to build, monitor, and manage connected devices and systems. It allows developers and businesses to collect, store, visualize, and analyze data from IoT devices in real-time. The platform supports a wide range of IoT devices, including Arduino, Raspberry Pi, ESP8266, and other microcontrollers, enabling seamless integration and communication between devices and the cloud.

AllThingsTalk offers an intuitive dashboard where users can create and customize widgets to visualize sensor data, set up real-time notifications, and control actuators. It supports various communication protocols such as MQTT, HTTP, and LoRaWAN, making it flexible and adaptable to different IoT ecosystems. For instance, a smart home project can use AllThingsTalk to monitor temperature, control lights, and track energy consumption through a user-friendly cloud dashboard.

Key features of AllThingsTalk include real-time data monitoring, device management, rule-based automation, and integrations with third-party services. Its scalability and ease of use make it suitable for prototyping, educational projects, and commercial IoT deployments. By leveraging its capabilities, users can quickly build IoT solutions without worrying about the complexities of backend infrastructure, enabling faster development cycles and improved connectivity.

To enable Arduino integration with AllThingsTalk, there are several libraries to be downloaded and installed:

- AllThingsTalk Arduino Library

  https://github.com/allthingstalk/arduino-wifi-sdk/archive/refs/heads/master.zip
- WiFiNINA

  https://github.com/arduino-libraries/WiFiNINA/archive/refs/heads/master.zip
- ArduinoJson

  https://github.com/bblanchon/ArduinoJson/archive/refs/heads/7.x.zip
- Scheduler

  https://github.com/arduino-libraries/Scheduler/archive/refs/heads/master.zip

User Account Creation

To get started with AllThingsTalk, follow these steps:

a) Open a web browser and navigate to the official website at https://www.allthingstalk.com/.

b) Scroll to the bottom of the webpage, locate the AllThingsTalk Maker section, and click on it.

c) On the AllThingsTalk Maker page, find and click the Sign Up button to begin the account creation process.

d) Fill in the required information, such as your name, email address, and a secure password. Alternatively, you may have options to sign up using social media accounts, depending on the platform.

e) Agree to the terms and conditions and complete any CAPTCHA verification if prompted.

f) Submit the form to create your AllThingsTalk account.

g) Check your email for a confirmation message and follow the instructions provided to verify your account (and set up multi-factor authentication).

Once you've successfully created an account, you'll be ready to log in, connect your Arduino devices, and start exploring the AllThingsTalk dashboard for data visualization and IoT project development.

Connect to an MCU Board

Login to AllThingsTalk Maker using a valid username and password. When the account is newly created, Figure 4A is shown. Proceed to click **+ CONNECT A DEVICE** (Connect to an MCU board). Select **Arduino MKR WiFi 1010** (Figure

4B) from a list of MCU boards and enter a unique name (Figure 4C) for the MCU Board (e.g. MyMkrWiFi1010) to connect.

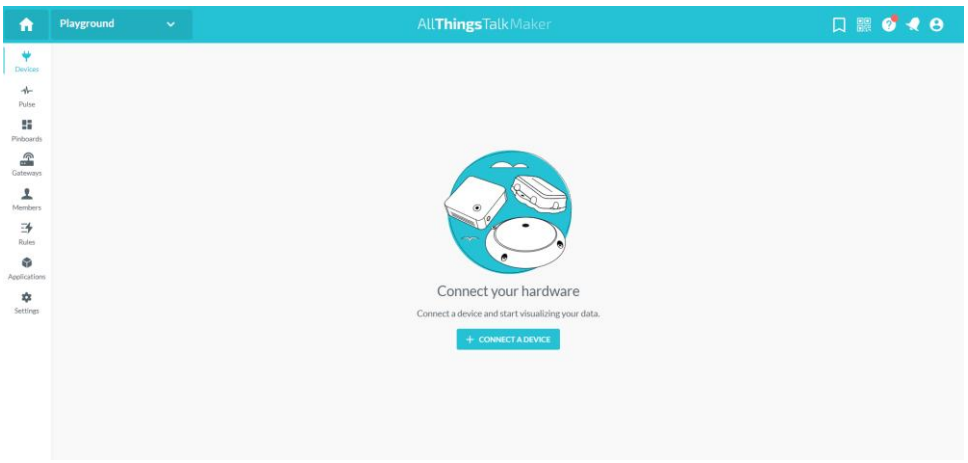

Figure 4A – Connect a Device

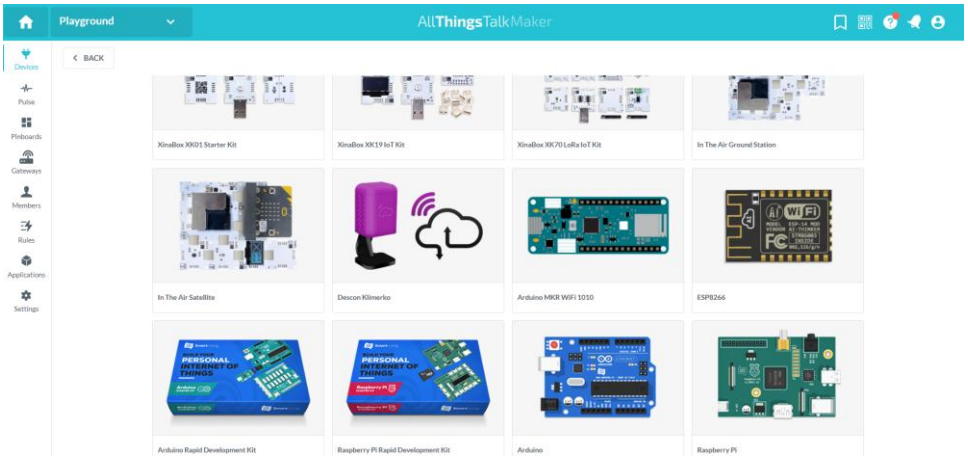

Figure 4B – MCU board selection

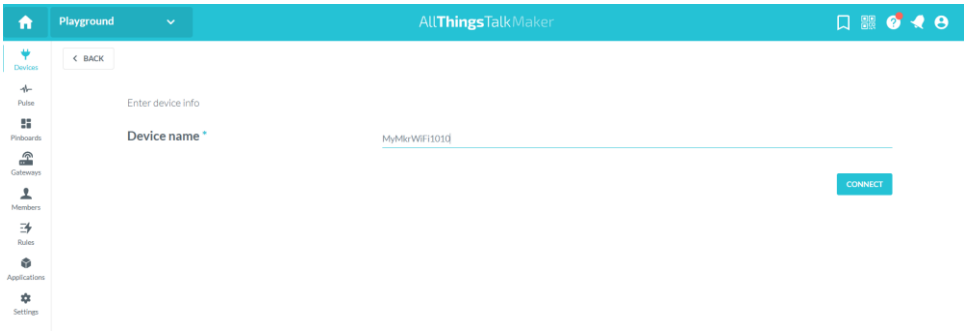

Figure 4C - MCU board name

Adding Sensors or Data Sources

Next step is to configure device assets (Figure 4D), where sensors or data sources are added. Click **CREATE ASSET** to start adding the required sensors. In this lesson we will be adding CO2 Sensor, Temperature Sensor and Humidity Sensor. Refers to Figure 4E – 4G on how to add the sensors. Figure 4H shows a list of sensors successfully added.

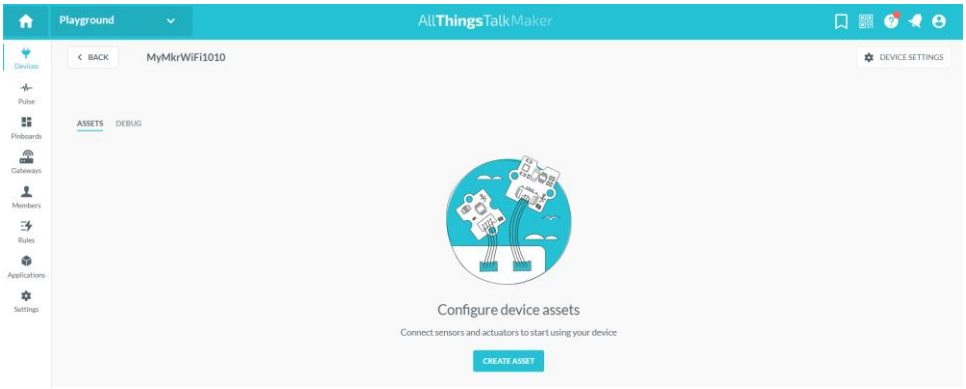

Figure 4D – Adding Sensors or Data Sources

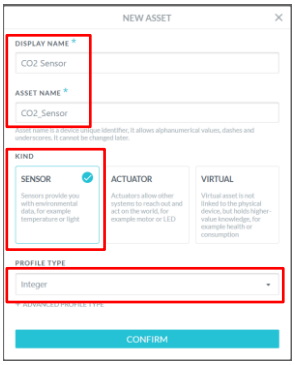

Figure 4E – Add CO2 Sensor

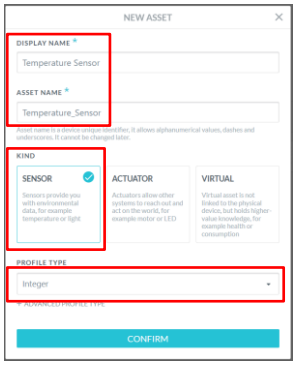

Figure 4F – Add Temperature Sensor

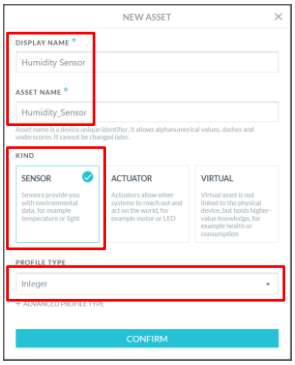

Figure 4G – Add Humidity Sensor

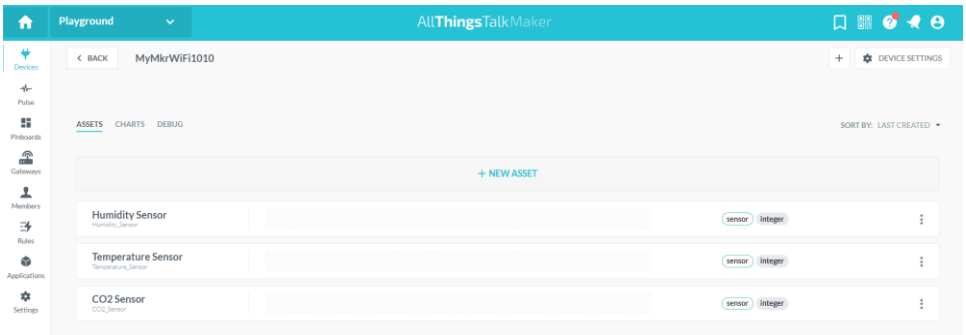

Figure 4H – Sensors added

Dashboard Design

Pinboard is a term used by AllThingsTalk referring to a Dashboard. Clicking on the Pinboards icon shows the available user created Pinboards. Figure 4I indicates that no Pinboard is created. Click on **+ CREATE NEW PINBOARD** to create a new Pinboard for displaying sensors data (CO2, Temperature and Humidity). Figure 4J shows an empty Pinboard, ready to be user customized.

To create a Pinboard that displays CO2 concentration in PPM using a Chart widget, start by navigating to the WIDGETS LIBRARY on the AllThingsTalk content page. Search for the Chart widget, drag it to the desired location on the Pinboard, position it and resize it, as shown in Figure 4K. Assign an appropriate WIDGET NAME, such as "CO2 Concentration (PPM)," and enter a relevant

keyword (e.g. CO) in the DATA SOURCE search field to locate and select the CO2 Sensor as data source for the MCU board defined earlier (MyMkrWiFi1010), and set the CHART TYPE preference to LINE CHART for optimal visualization of the sensor data. Repeat this process to create similar Line Charts for displaying Temperature and Humidity readings, using the corresponding sensors as data sources. Once all widgets are configured, your completed Pinboard design, as shown in Figure 4L, will display real-time data from the CO2, Temperature, and Humidity sensors, offering a comprehensive overview of environmental conditions.

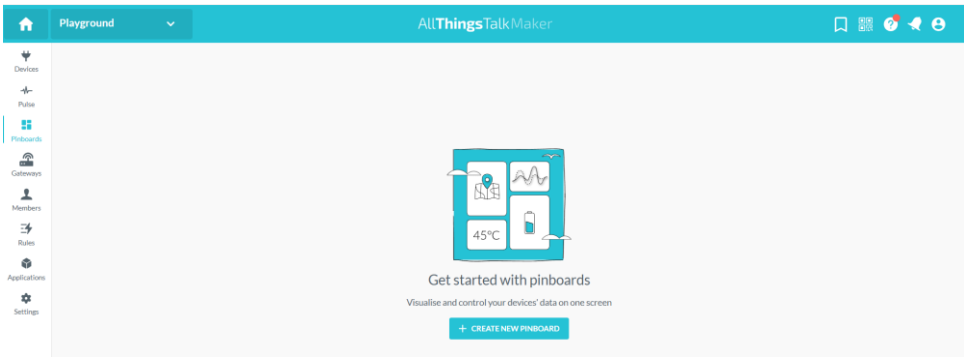

Figure 4I – No Pinboard available

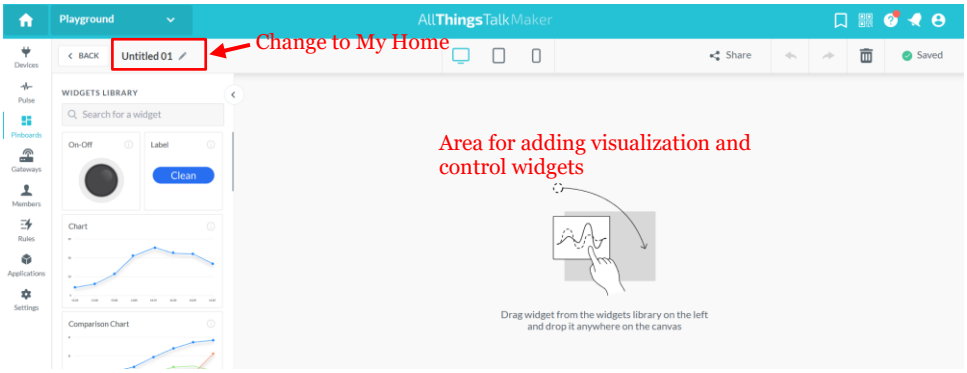

Figure 4J – Empty Pinboard

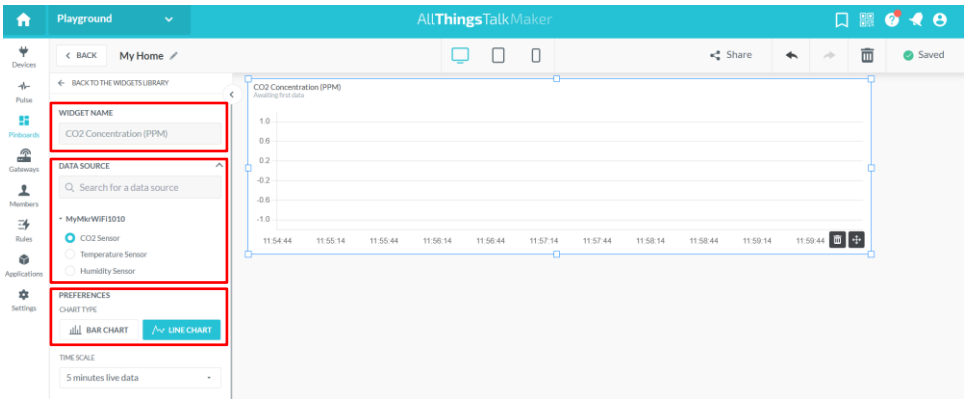

Figure 4K – Pinboard with a Chart widget for displaying CO2 PPM data

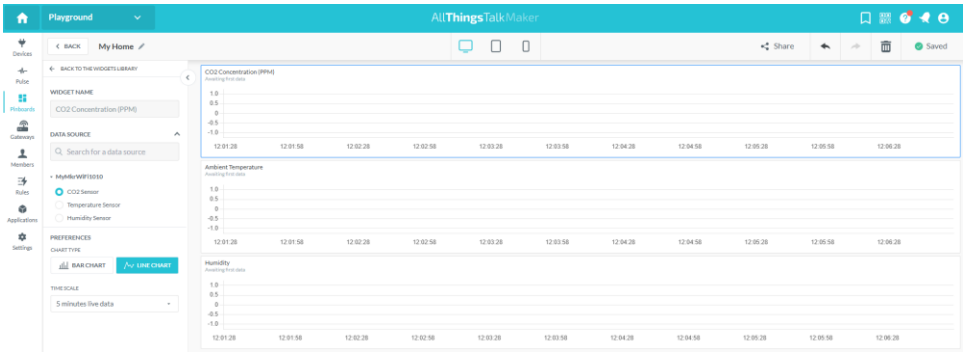

Figure 4L – Pinboard for displaying CO2, Temperature and Humidity data

Arduino Board and AllThingsTalk Integration

The *CO2THSensorDashboard.ino* sketch (Figure 4M) demonstrates how to integrate the **CO2, Temperature, and Humidity Sensor (SCD30)** with the AllThingsTalk Maker platform for real-time data collection and visualization. It uses the **Arduino MKR WiFi 1010** board to read sensor data (details in Lesson 2), transmits the data to AllThingsTalk over WiFi, and displays it on the platform.

The system leverages on several libraries, including *AllThingsTalk_WiFi.h* for AllThingsTalk integration, *WiFiNINA.h* for WiFi management, and *SCD30.h* for interfacing with the SCD30 sensor. The sketch also requires a valid *WiFiSSID*, *WiFiPassword*, *AllThingsTalk DeviceID* and *AllThingsTalk DeviceToken*.

In the *setup()* function, the sketch initializes the I2C communication for the SCD30 sensor, sets up serial communication for debugging, and connects the Arduino board to the AllThingsTalk platform using WiFi credentials and device authentication details. The platform assets are pre-defined for CO2 concentration, temperature, and humidity, allowing for seamless transmission of these sensor readings.

In the *loop()* function, the code periodically checks for new data from the SCD30 sensor. If data is available, it reads the CO2 concentration (in PPM), temperature (in °C), and humidity (in %), storing these values in an array. The readings are printed to the Serial Monitor in a comma-separated format for local debugging and monitoring. The sensor data is then sent to the respective assets on AllThingsTalk using the *device.send()* method, with delays introduced between transmissions to ensure stable communication. The *device.loop()* function maintains the connection with the AllThingsTalk platform.

This sketch offers a practical framework for building an environmental monitoring system, combining Arduino hardware with AllThingsTalk's cloud capabilities for data visualization. It provides real-time updates on CO2 levels, temperature, and humidity, making it suitable for smart home automation, agricultural monitoring, or IoT-based educational projects. The addition of debug information and structured communication ensures reliability and ease of use, making it a robust solution for IoT enthusiasts and developers.

CO2THSensorDashboard.ino

```
#include <AllThingsTalk_WiFi.h> // AllThingsTalk Library
#include <WiFiNINA.h>           // WiFi Library (Arduino MKR WiFi 1010).
#include <utility/wifi_drv.h>   // Exposes MKR1010's underlying functions to control the RGB LED
#include "ArduinoJson.h"        // Arduino JSON Library
#include "Scheduler.h"          // Arduino Scheduler Library
#include "SCD30.h"              // CO2, Temperature and Humidity Sensor Library
#define SERIAL Serial

auto wifiCreds   = WifiCredentials("WiFiSSID", "WiFiPassword");   // Your WiFi Network Name and Password
auto deviceCreds = DeviceConfig("DeviceID", "DeviceToken");       // Go to AllThingsTalk Maker > Devices > Your Device > Settings > Authentication to get your Device ID and Token
auto device      = Device(wifiCreds, deviceCreds);                // Create "device" object
char* CO2_SENSOR  = "CO2_Sensor";                                 // (ATT Asset) CO2_Sensor
char* TEMP_SENSOR = "Temperature_Sensor";                         // (ATT Asset) Temperature_Sensor
char* HUMI_SENSOR = "Humidity_Sensor";                            // (ATT Asset) Humidity_Sensor

void setup() {
    Wire.begin();               // Initialize I2C Communication
    SERIAL.begin(115200);       // Initialize Serial Port
    scd30.initialize();         // Initialize Grove – CO2, Temperature, and Humidity Sensor
    device.debugPort(Serial);   // Set AllThingsTalk library to output its debug to "Serial"
    device.init();              // Initialize WiFi and AllThingsTalk
}

void loop() {
    float result[3] = {0};
    if (scd30.isAvailable()) {                          // Check for Sensor availability
        scd30.getCarbonDioxideConcentration(result);    // Get CO2, Temp and Humidity data from sensor
        SERIAL.print(result[0]);                        // CO2 Concentration on PPM
        SERIAL.print(",");                              // Delimiter (comma)
        SERIAL.print(result[1]);                        // Ambient Temperature in Degree C
        SERIAL.print(",");                              // Delimiter (comma)
        SERIAL.println(result[2]);                      // Humidity in %
        device.send(CO2_SENSOR,  (int) result[0]);      // Send CO2 data to AllThingsTalk
        delay(1000);
        device.send(TEMP_SENSOR, (int) result[1]);      // Send Temperature data to AllThingsTalk
        delay(1000);
        device.send(HUMI_SENSOR, (int) result[2]);      // Send Humidity data to AllThingsTalk
        delay(1000);

    }
    delay(2000);
    device.loop();                                      // Keep AllThingsTalk connection alive
}
```

Please enter the correct:
WiFiSSID,
WiFiPassword,
DeviceID,
DeviceToken

Figure 4M – Grove – CO2, Temperature and Humidity Sensor data transmitted to AllThingsTalk for visualization on a cloud dashboard.

4.2 ThingSpeak (Advanced Content)

ThingSpeak is a cloud-based IoT analytics platform designed to collect, store, analyze, and visualize data from connected devices and sensors. Developed by MathWorks, it enables seamless integration with MATLAB, making it a powerful tool for advanced data analysis and predictive modeling. ThingSpeak is used in IoT projects for real-time monitoring and control, offering customizable dashboards and graphical tools like line charts, bar graphs, and gauges to provide actionable insights.

ThingSpeak supports data collection from various devices, such as Arduino, and Raspberry Pi, and ESP8266, through HTTP or MQTT protocols. Each device can have its own data channel with up to eight fields for storing different types of data, such as temperature, humidity, or pressure. This data can be stored for historical analysis, making it suitable for long-term monitoring projects. With ThingSpeak's visualization features, users can create and customize widgets for real-time updates, allowing easy identification of trends and anomalies in the data.

A standout feature of ThingSpeak is its integration with MATLAB, which enables users to perform advanced data processing, predictive analytics, and implement custom algorithms directly within the platform. For example, MATLAB integration can be used to analyze weather patterns, predict equipment failures, or optimize energy usage. Additionally, ThingSpeak supports triggers and alerts, allowing users to configure notifications based on predefined conditions. These alerts can be sent via email or other communication channels to prompt proactive responses to critical events.

ThingSpeak's RESTful APIs make it easy to integrate with a variety of devices and third-party applications, enhancing its flexibility for different IoT applications. This capability supports a wide range of use cases, such as environmental monitoring, home automation, predictive maintenance, and educational projects. In environmental monitoring, ThingSpeak can track air quality, temperature, and soil moisture levels, while in home automation, it can control lighting, heating, or security systems based on real-time data. Its predictive maintenance capabilities are particularly useful for industries, as it allows users to analyze sensor data and predict equipment failures, reducing downtime and costs.

To enable Arduino integration with ThingSpeak, the following library is to be downloaded and installed:

https://github.com/mathworks/thingspeak-arduino/archive/refs/heads/master.zip

User Account Creation

To create and access a user account on ThingSpeak, start by navigating to https://thingspeak.com/ using a web browser. On the homepage, click on the **Sign In** button located at the top-right corner, next to "How to Buy". If you do not have an account, proceed to create one by providing a valid email address and setting a secure password. Complete the required steps, such as agreeing to the terms of service. After submitting your information, check your email for a confirmation message from ThingSpeak, and click the verification link to activate your account. Once the account is successfully created, return to the ThingSpeak website and log in using your email and password, as shown in Figure 4N. You are now ready to create channels, configure devices, and utilize the platform's features for IoT data visualization and analysis.

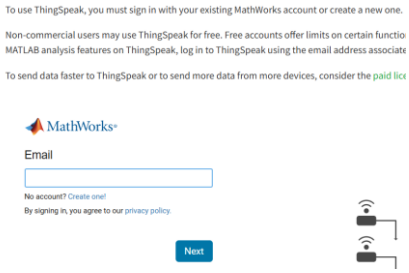

Figure 4N – ThingSpeak login page

Dashboard Design

In ThingSpeak, a **Channel** serves as the equivalent of a dashboard, allowing users to store, visualize, and analyze data. To create a new Channel, click on the **New Channel** button. As depicted in Figure 4O, a form will appear where you can configure your Channel settings. For this lesson, fill in the **Name**, **Description**, and fields for **Field1**, **Field2**, and **Field3** using the provided information, ensuring that each field corresponds to a specific data type or sensor reading.

After completing the form, scroll to the bottom of the page and click the **Save Channel** button to finalize the setup. This action creates a new Channel, which acts as your **Smart Home Dashboard**, as shown in Figure 4P. The Channel layout is designed for effective data visualization and analysis, with built-in compatibility for MATLAB integration.

The Channel layout can be customized to suit specific visualization and data analysis needs by interacting with various buttons and exploring the tabs. These customization options allow users to modify the arrangement of widgets, add or remove fields, and adjust data display formats to create a more user-friendly and informative dashboard.

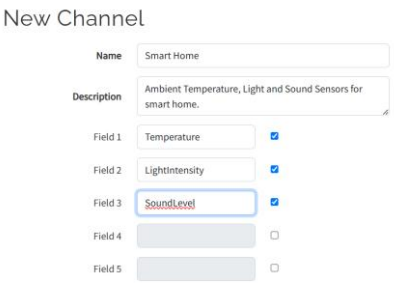

Figure 4O – ThingSpeak New Channel setup for Smart Home

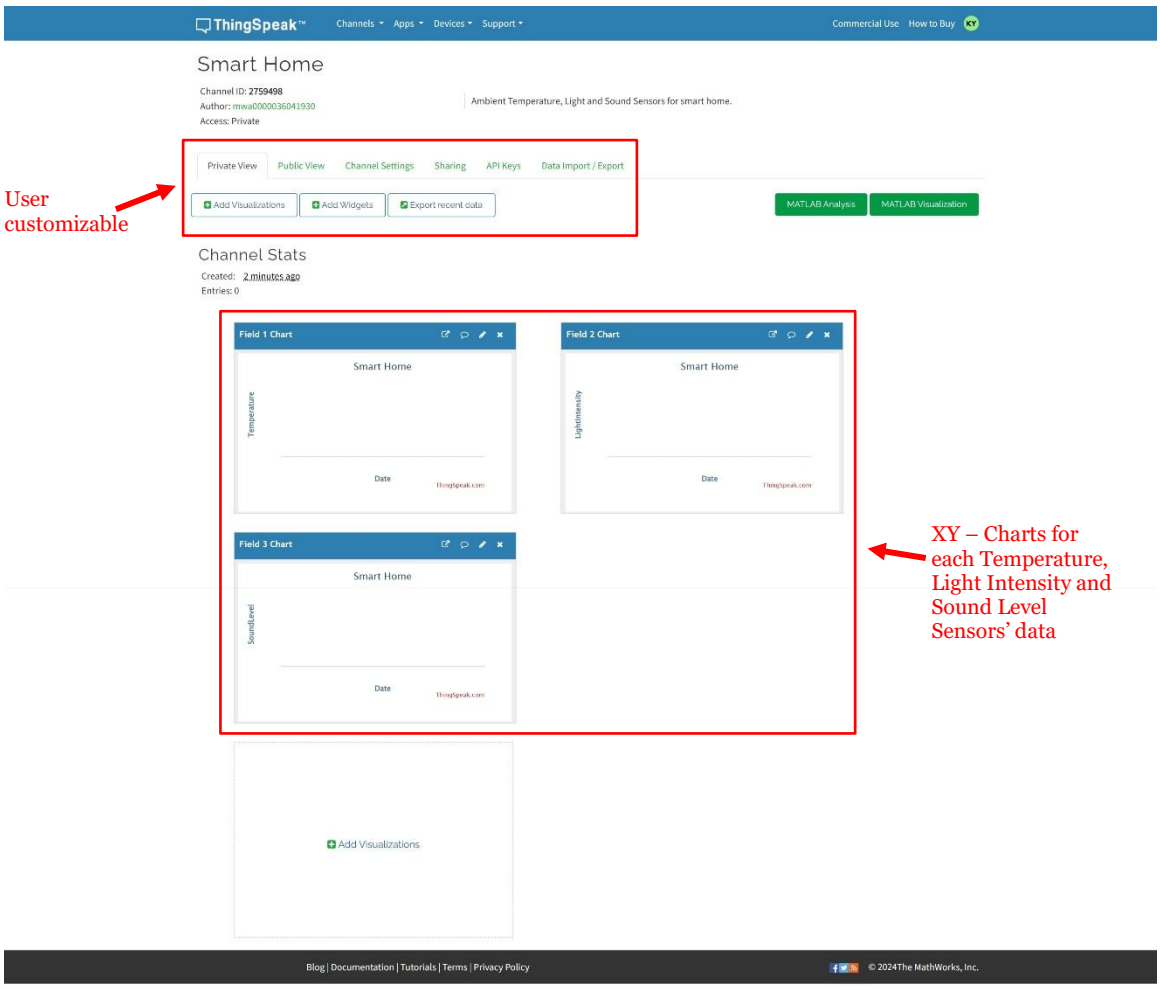

Figure 4P – Smart Home Channel (Dashboard)

Arduino Board and ThingSpeak Integration

In this lesson, the **Temperature Sensor, Sound Sensor, and Digital Light Sensor** are utilized as part of a Smart Home monitoring system. These sensors will provide data for temperature, sound intensity, and ambient light levels, respectively. To set up the system, connect the sensors to the **Arduino MKR WiFi 1010 board** following the procedures outlined in Lesson 2.

- **Temperature Sensor**: Connect this sensor to an analog port (e.g., Port A1) on the MKR Connector Carrier Board.
- **Sound Sensor**: Attach this sensor to another analog port (e.g., Port A2) to monitor environmental noise levels.
- **Digital Light Sensor**: Connect this sensor to the I2C port on the MKR Connector Carrier Board, ensuring proper communication for light intensity readings.

Once connected, the Arduino MKR WiFi 1010 will serve as the data acquisition system, collecting readings from these sensors and transmitting the data to a cloud platform, such as ThingSpeak, for real-time visualization and analysis on the Smart Home Channel.

This Arduino sketch *smartHomeDashboard.ino* (Figure 4Q) integrates three sensors - Temperature Sensor, Sound Sensor, and Digital Light Sensor with the ThingSpeak platform to collect, process, and upload environmental data for real-time cloud-based visualization.

The sketch utilizes libraries such as *math.h* for mathematical operations, *Wire.h* for I2C communication, *WiFiNINA.h* for WiFi connectivity, *Digital_Light_TSL2561.h* for interfacing with the light sensor, and *ThingSpeak.h* for managing data uploads to the ThingSpeak IoT platform. The sketch also requires valid *WiFiSSID*, *WiFiPassword*, ThingSpeak *Channel Number* and ThingSpeak *Write API Key*.

In the *setup()* function, the system initializes the I2C communication for the light sensor, configures the analog read resolution to 12 bits for higher precision, and establishes a connection with the WiFi module. If the WiFi module is absent, the sketch halts further operations. The TSL2561 light sensor is initialized, and the ThingSpeak client is set up to handle data transmission. The serial communication is also initialized for debugging and status logging to the Serial Monitor.

The *loop()* function begins by checking the WiFi connection status. If the connection is lost, the sketch attempts to reconnect by repeatedly initiating the WiFi module with the provided network credentials. Sensor data acquisition follows, where the Temperature Sensor reads raw analog values, converts them to resistance using the Steinhart–Hart equation, and calculates the temperature in Celsius. The Sound Sensor captures sound intensity as a raw analog value, while the Digital Light Sensor measures visible light intensity in Lux.

After processing the sensor readings, data is sent to ThingSpeak by updating three fields in the platform's channel: Field 1 for temperature, Field 2 for light intensity, and Field 3 for sound level. A status message, *Sensor Values Update*, is also included. The success or failure of the data upload is logged to the Serial Monitor, providing real-time feedback for debugging. A 20-second delay is

introduced at the end of each loop to comply with ThingSpeak's update rate limits.

smartHomeDashboard.ino

```
#include <math.h>                    // Math Library
#include <Wire.h>                    // I2C Library
#include <WiFiNINA.h>                // WiFi Library
#include <Digital_Light_TSL2561.h>   // Digital Light Sensor Library
#include "ThingSpeak.h"              // Always include thingspeak header
#define TEMP_SENSOR  A1              // Temperature Sensor connects to An
#define SOUND_SENSOR A2              // Sound Sensor connects to Arduino
const int B  = 4275;                 // B value of the thermistor (from D
const int R0 = 100000;               // R0 = 100k
int Temperature = 0;                 // Temperature Sensor
int Sound       = 0;                 // Sound Sensor
int Light       = 0;                 // Light Intensity

unsigned long myChannelNumber = 0000000;          // Replace 0000000 with your channel number
const char * myWriteAPIKey    = "WRITE_APIKEY";   // Replace WRITE_APIKEY with your channel write API Key
char ssid[]  = "MYSSID";                          // Replace MYSSID with your WiFi network name
char pass[]  = "MYPASSWORD";                      // Replace MYPASSWORD with your WiFi password
WiFiClient  client;                               // Initialize WiFi

void setup() {···
}
void loop() {
    if(WiFi.status() != WL_CONNECTED)             // Connect or reconnect to WiFi
    {···
    }
    int   a = analogRead(TEMP_SENSOR);            // Read Temperature Sensor raw value
    float R = 4095.0/a - 1.0;
    R = R0 * R;
    // converts R to temperature using simplified Steinhart-Hart equation
    float temperature = 1.0/(log(R/R0)/B + 1/298.15) - 273.15;
    Temperature = (int) temperature;
    Sound = analogRead(SOUND_SENSOR);             // Read Sound Sensor raw value (Sound Level)
    Light = TSL2561.readVisibleLux();             // Read visible light intensity (Lux)
    ThingSpeak.setField(1, Temperature);          // Update ThingSpeak Field1 (Temperature)
    ThingSpeak.setField(2, Light);                // Update ThingSpeak Field2 (LightIntensity)
    ThingSpeak.setField(3, Sound);                // Update ThingSpeak Field3 (SoundLevel)
    ThingSpeak.setStatus("Sensor Values Update"); // Set ThingSpeak Status

    // write to the ThingSpeak channel
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if(x == 200){···
    }
    else{···
    }
    delay(20000); // Wait 20 seconds to update the channel again
}
```

Please enter the correct:
WiFiSSID,
WiFiPassword,
ThingSpeak Channel Number,
ThingSpeak Write API Key

Figure 4Q - Temperature, Sound and Digital Light Sensors data transmitted to ThingSpeak for visualization on a cloud dashboard.

Figure 4R illustrates the Smart Home Channel displaying live sensor data plots, where data from the sensors is transmitted at 20-second intervals. These visualizations provide basic insights into the data but come with limited customization and viewing options. To enhance the analysis and presentation of the data, MATLAB's online data analytics and visualization services can be integrated with ThingSpeak. MATLAB allows for advanced data processing, such as trend analysis, predictive modeling, and customized graphical representations, offering a more detailed and insightful view of the sensor data. By leveraging MATLAB's capabilities, users can create tailored visualizations and perform complex analytics, elevating the Smart Home Channel from basic monitoring to a comprehensive data-driven decision-making tool.

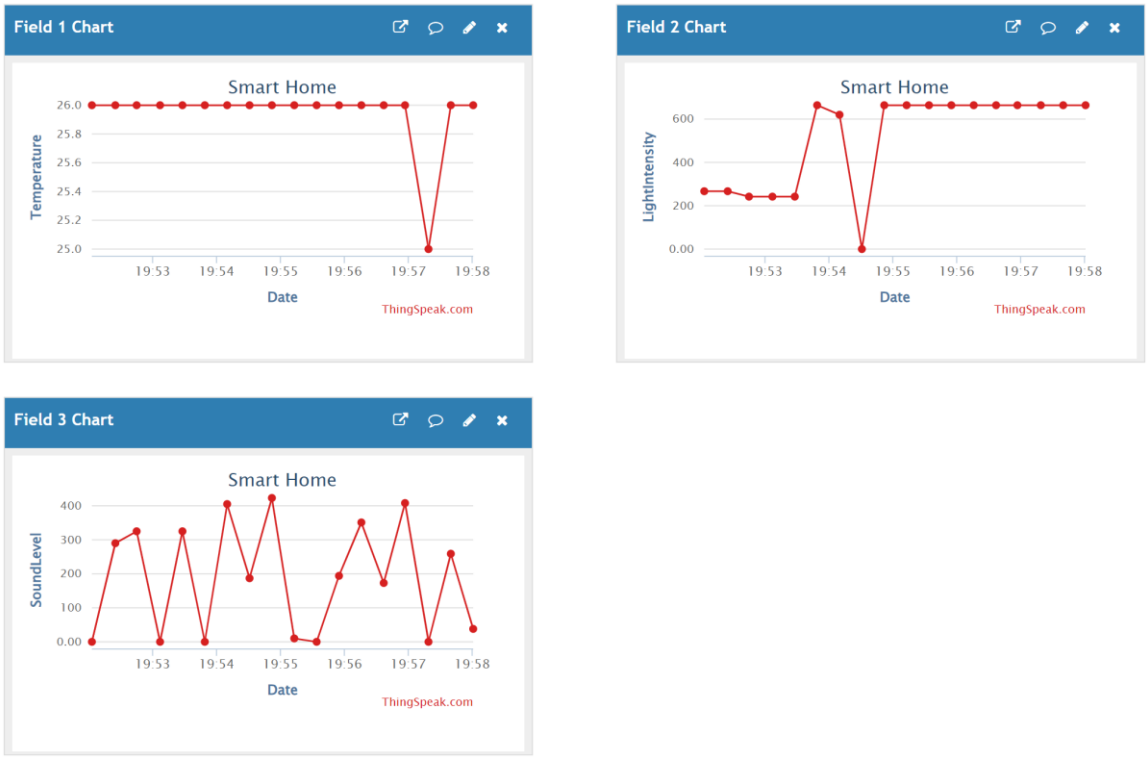

Figure 4R - Smart Home Channel with live data

- The End –