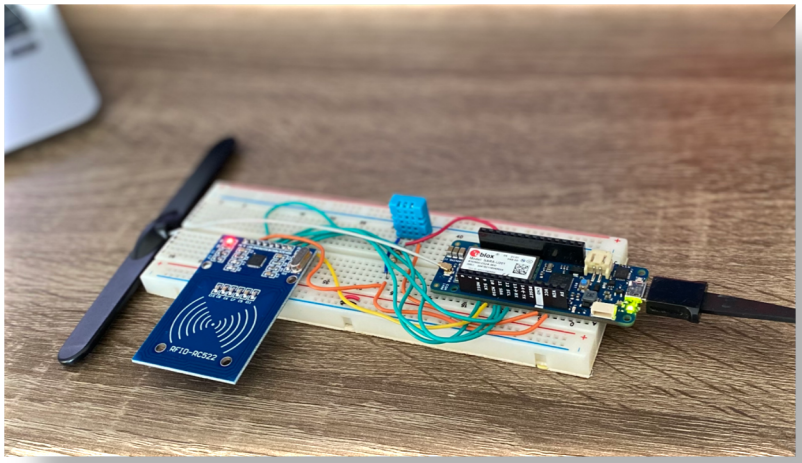
SMART CLASSROOM

Automatic Attendance and Classroom Environmental Monitoring



by

Ana Paula Cardoso Ferraz

EEM 602 – Internet of Things

Professor: Dr. Mohab Mangoud

University of Bahrain

College of Engineering

Bahrain

January 2022

ABSTRACT

The objective of this project was to create a prototype of an application of the Internet of Things in education. The traditional attendance roll call is a time-consuming task and if automated can give the instructor time to focus on other teaching activities. An RFID RC522 module was used to implement an automatic attendance system. To maintain the right physical environment and ensure effective learning a relative humidity and temperature sensor DHT11 was used to monitor the classroom environment and inform if the values were out of range. Both sensors were connected to an Arduino MKR GSM 1400 and data was sent to Arduino IoT cloud for monitoring. The Arduino sketch is explained, IoT cloud setup steps and solutions to challenges faced are explained in this report.

CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Motivation

The Smart classroom aims to bring students and instructors closer together, it helps the instructor to teach more efficiently and makes the environment more convenient for teaching and learning [1].

Taking attendance traditionally (roll call) is a time-consuming task, if automated the instructor can focus on other teaching activities. There are many technologies being used for automatic attendance: RFID, NFC, Smartphone with Bluetooth, Biometric (fingerprint or face recognition) [2]. In this project RFID reader module and RFID cards will be used to implement an automatic attendance system.

The classes are conducted inside a confined space and is difficult to attend a lecture if the environment is too hot or too cold. To achieve effective learning, the right physical environment inside the classroom should be maintained. There are many types of sensors being used in the Smart Classroom to measure air quality, light, temperature, humidity, acoustics, and radiation [3]. The Relative Humidity and Temperature sensor (DHT11) will be used in this project to monitor the temperature and humidity and inform if the values are out of range.

1.2 Objectives

- Simulation of an automatic attendance using RFID tags
- RFID readings to be recorded in an online attendance registry
- Acquire temperature and relative humidity data from classroom
- IoT sensors data to be sent to a cloud to be displayed in a graph

1.3 Project Structure

The RFID module scans the cards/tags and sends the information to the Arduino MKR GSM 1400 board using SPI bus communication. The DHT11 sensor acquires the calibrated temperature and relative humidity data and sends to the board through pin 2. More detailed information about each sensor and the Arduino MKR GSM 1400 board are mentioned in Chapter 2. The Arduino board sends data to cloud via GSM 850 MHz and connects to the MQTT broker (mqtts-sa.iot.arduino.cc:8883).

The project data flow can be seen at Figure 1-1 and the final project prototype at Figure 1-2.

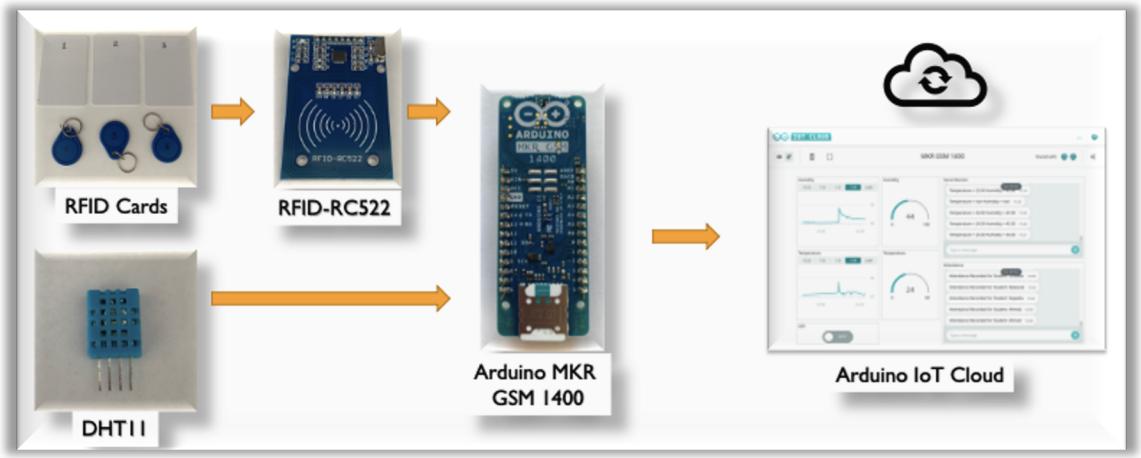


Figure 1-1 - Project Flowchart

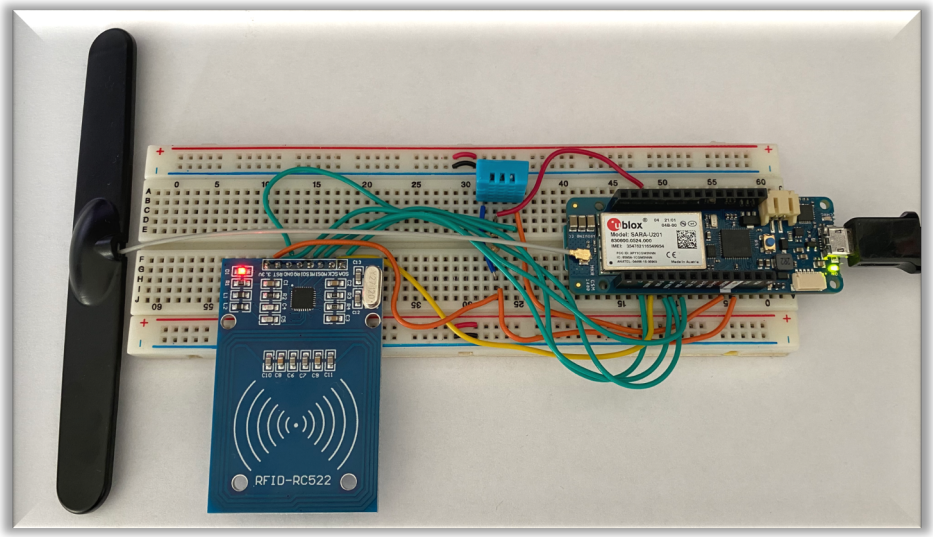


Figure 1-2 - Final project prototype

CHAPTER 2
Board and Sensors Information

For this project the following components were used:

- Arduino MKR GSM 1400
- DHT11 - Digital Relative Humidity and Temperature Sensor
- RFID-RC522 - Radio Frequency Identification Module
- Arduino IoT Cloud platform

2.1 Arduino MKR GSM 1400

The MKR GSM 1400 is a great option for GSM / 3G connectivity projects.
It uses Arm® Cortex®-M0 32-bit SAMD21 processor, it also features the powerful u-blox SARA-U201 module and the ECC508 crypto-chip for security. [4]

- u-blox SARA U201 – Enables GSM/3G connectivity for the MRK GSM 1400 board – Carrier frequency: GSM 850 MHz, E-GSM 1900 MHz, DCS 1800 MHz, PCS 1900 MHz
- Cortex-M0 32-bit SAMD21 – powerful low-power processor
- ATECC508 crypto chip – makes sure data remains secure and private, can store up to 16 keys in an EEPROM array

Figure 2-1 - Arduino MKR GSM 1400

Figure 2-2 shows the pinout for Arduino MKR GSM 1400. In this project we used the following pins:

- VCC (3.3V) – to power up RFID module and DHT11 sensor
- GND – ground for RFID and DHT11 sensor
- D2 – digital pin 2 – for DHT11 data
- D8 – digital pin 8 – MOSI
- D9 – digital pin 9 – SCK
- D10 – digital pin 10 - MISO
- D11 – digital pin 11 – SDA (SS)
- D11 – digital pin 12 – SCL (RST)

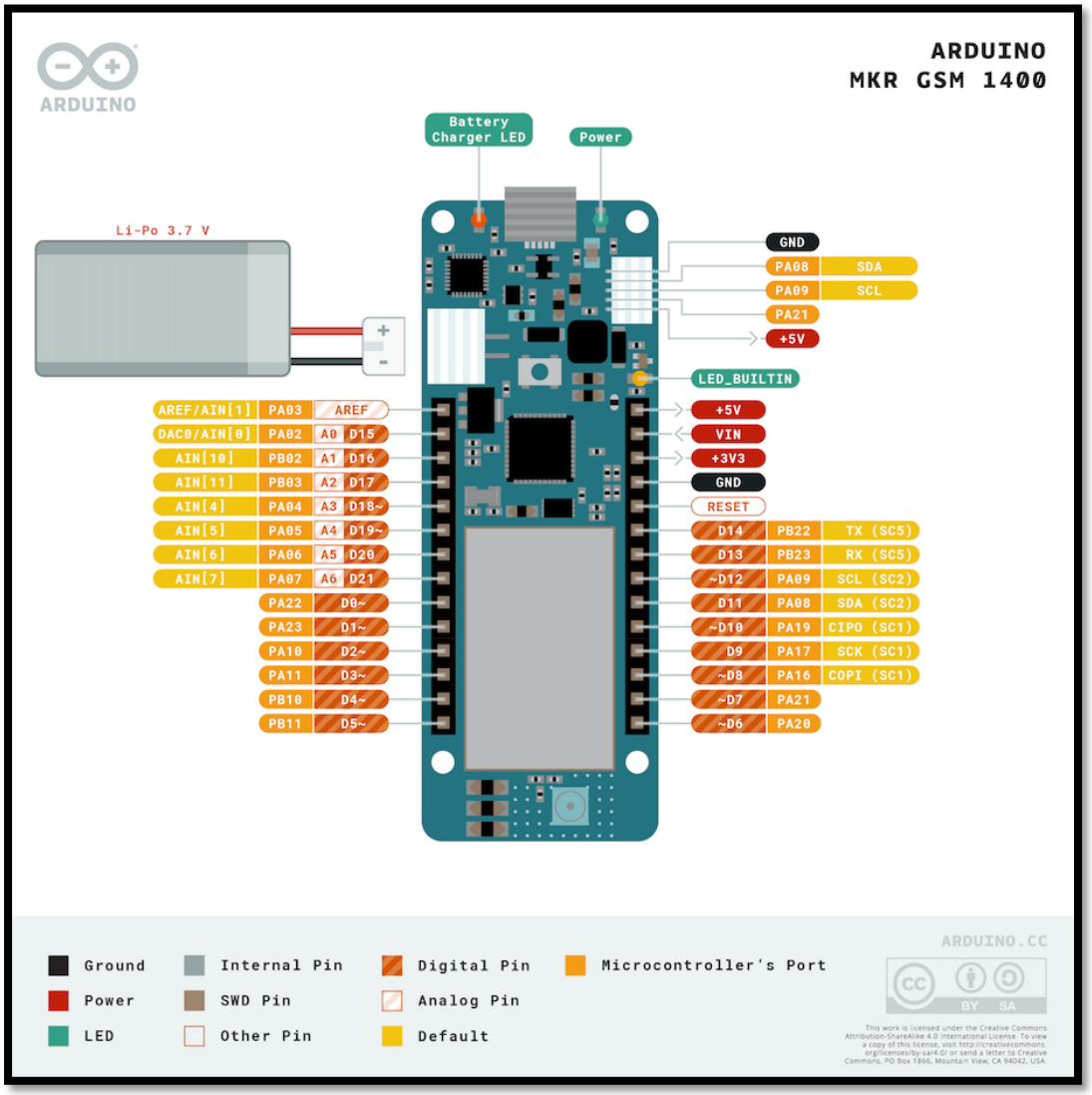


Figure 2-2 - Arduino MKR GSM 1400 Pinout

2.2 DHT11 – Digital Relative Humidity and Temperature Sensor

DHT11 outputs a calibrated digital signal. It applies exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability. Its sensing elements are connected with 8-bit single-chip computer. Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory. Small size & low consumption & long transmission distance (100m) enables DHT11 to be suited in all kinds of harsh application occasions. Single row packaged with four pins, making the connection very convenient. [4]

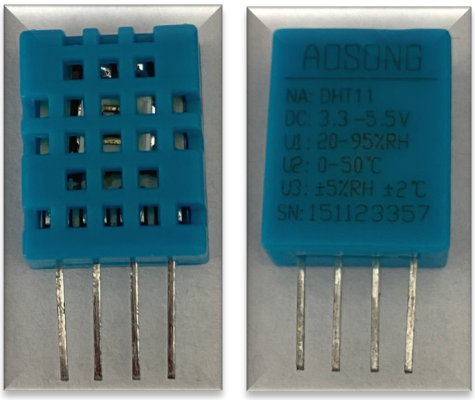


Figure 2-3 - DHT11 Digital relative humidity and temperature sensor

DATA=16 bits RH data+16 bits Temperature data+8 bits check-sum

Example: MCU has received 40 bits data from DHT11 as

0010 0001	0000 0000	0001 1010	0000 0000	0011 1011
Integral part of RH	Decimal part of RH	Integral part of T	Decimal part of T	check sum

Remarks: The decimal part of RH and T is always 0000 0000.

Here we convert integral part of RH from binary system to decimal system,

0010 0001 → 33

Binary system Decimal system, **RH=33%RH**

Here we convert integral part of T from binary system to decimal system,

0001 1010 → 26

Binary system Decimal system, **T=26 Celsius**

Sum=0010 0001+0000 0000+0001 1010+0000 0000=0011 1011

Check-sum=the last 8 bits of Sum=0011 1011

Figure 2-4 - Illustration of humidity and temperature data transmission [4]

2.3 RFID-RC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56 MHz [5]. The module can support I2C, SPI (used in this project), UART and has an RFID card and key fob. It is commonly used in attendance systems and other person/object identification applications.

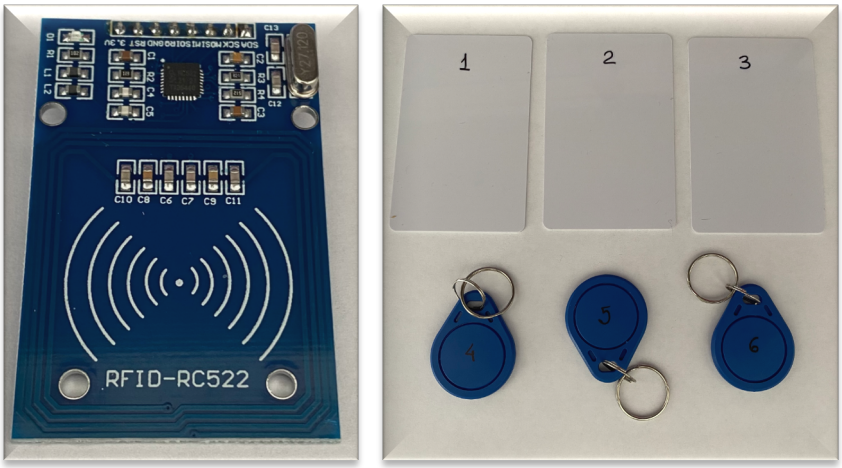


Figure 2-5 - RFID Module, cards and key fobs

The SPI (Serial Peripheral Interface) communication was used for the RFID module, it uses 4 pins from the Arduino board:

- SCK – Serial Clock (output from board) pin 9
- MISO – Master In Slave Out (data output from receiver) – pin 10
- MOSI – Master Out Slave In (data output from board) – pin 8
- SS – Slave Select (output from board – indicate data is being sent) – pin 11

Figure 2-6 - SPI communication connections

CHAPTER 3
Arduino Code

This session is dedicated to explaining the main parts of the Arduino Code. The full code is available at the Appendix.

3.1 Libraries and Definitions

Before the start of the code, the libraries need to be included in the program. The pin numbers that are physically connected to the Arduino board are defined. The instances of the RFID receiver (mfrc522) and the DHT sensor (dht) that will be used for the data acquisition are created. A variable called student is created, which is a two-dimensional array of strings that stores the student's name and the associated RFID tag ID.

```
#include <SPI.h>
#include <MFRC522.h>
#include "thingProperties.h"
#include "DHT.h"
#define DHTpin 2
#define DHTTYPE DHT11
#define SS_PIN 11
#define RST_PIN 12
MFRC522 mfrc522(SS_PIN, RST_PIN);

String student[6][2] = {
  {"90 17 34 37", "Ahmed" },
  {"D3 AA D0 45", "Mubarak" },
  {"52 98 81 39", "Najeeba" },
  {"71 E4 AE 20", "Ghadeer" },
  {"E3 72 52 1A", "Fatema" },
  {"91 39 45 20", "Jawaher" }
};
DHT dht(DHTpin, DHTTYPE);
```

Figure 3-1 - Libraries and Definitions

3.2 Setup

The **setup ()** function is executed only once when the board is powered up or when the reset button is pressed. The serial connection is initialized with a baud rate of 9600. The DHT sensor acquisition calls the function **begin ()** to start the acquisition.
Some Arduino IoT Cloud functions are called to initialize properties, begin the connection, set the debug message levels and to print any debug information.

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  //while (!Serial);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  dht.begin();
  // Defined in thingProperties.h
  initProperties();
  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  delay(4);
}
```

Figure 3-2 - Setup function

3.3 Main function - Loop

The **loop ()** function in Arduino includes the begin of the SPI communication and the mfrc522 initialization. At first, these two initializations were done at the setup phase, but were moved to the loop due to a bug when the Arduino Cloud is updated [7]. It changes pin 10 to input, breaking the SPI communication. The solution was to initialize the SPI communication on every loop to reset pin 10 to output mode.

```
void loop() {
  SPI.begin();
  mfrc522.PCD_Init();
  if ( mfrc522.PICC_IsNewCardPresent()) {
    if (mfrc522.PICC_ReadCardSerial()) {
      String content = "";
      byte letter;
      for (byte i = 0; i < mfrc522.uid.size; i++)
      {
        content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        content.concat(String(mfrc522.uid.uidByte[i], HEX));
      }
      //Serial.println(content.substring(1));
      content.toUpperCase();
      for (int i = 0; i < 6; i++) {
        if (content.substring(1) == student[i][0])
        {
          msg_Attendance = "Attendance Recorded for Student: " + student[i][1] ;
        }
      }
      Serial.print(msg_Attendance);
    }
  }
  dht_sensor_getdata();
  delay(500);
  ArduinoCloud.update();
}
```

Figure 3-3 - Main function - loop

After the initialization of the mfrc522, the function **PICC_IsNewCardPresent ()** is called. In case a new card is detected, the function **PICC_ReadCardSerial ()** reads the ID information from the RFID card. The ID is a variable of type string called content. It is converted to upper case to be compared with the array of **student** IDs stored in the definition part of the program. When the ID read from the card matches a student from the record, the attendance is recorded and stored in the variable **msg_Attendance.**
The **dht_sensor_getdata ()** function is called, more detailed information in the section 3.4. A short delay of 500 ms is introduced before calling the **ArduinoCloud.update()** function. This function sends all four variables to the cloud: **temperature, humidity, msg_Attendance** and **msgTempHum**.

3.4 DHT sensor function

The function **dht_sensor_getdata ()** was created to read the humidity and temperature data from the DHT11 sensor. The dht.h library imports the functions **readHumidity ()** and **readTemperature ().**

The if-else conditional checks for the thresholds of low temperature (20ºC) and high temperature (27ºC) and then sends a message to the dashboard. This could be implemented with an actuator (to turn on or off the air conditioner) or an alarm in the real classroom.

```
void dht_sensor_getdata()
{
  float hm = dht.readHumidity();
  Serial.print(F("Humidity "));
  Serial.println(hm);
  float temp = dht.readTemperature();
  Serial.print(F("Temperature "));
  Serial.println(temp);
  humidity = hm;
  temperature = temp;
  if (temp > 27) {
    msgTempHum = "Temperature = " + String (temperature) + "  Humidity = " + String(humidity) + " -> Temperature High ";
  }
  else if (temp < 20) {
    msgTempHum = "Temperature = " + String (temperature) + "  Humidity = " + String(humidity) + " -> Temperature Low ";
  }
  else {
    msgTempHum = "Temperature = " + String (temperature) + "  Humidity = " + String(humidity) + " -> All ok ";
  }
}
```

Figure 3-4 - Temperature and Humidity function

CHAPTER 4
IoT Cloud

This chapter focus on the IoT Cloud platform used to acquire the data from the Arduino board.

4.1 Setting up the Cloud

To connect to the Arduino IoT Cloud, first we need to create an account or sign in as shown in Figure 4-1.

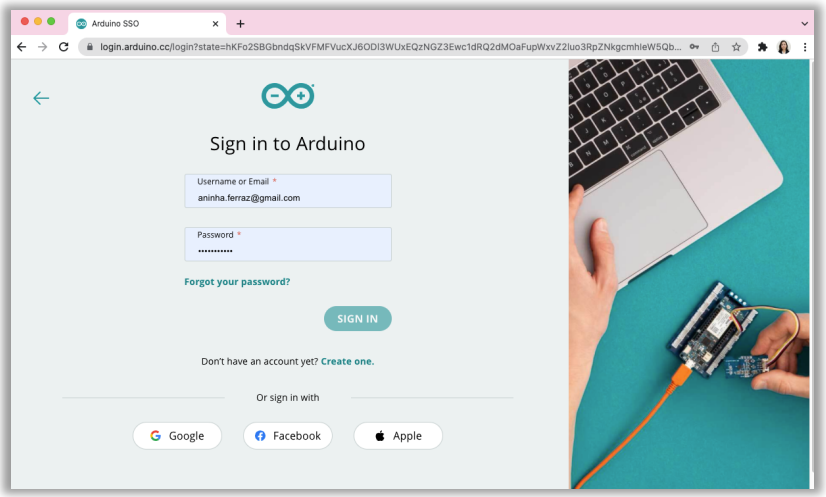


Figure 4-1 - Sign in to Arduino

To use the Web-based editor, that has all cores and libraries already installed, we need to install the Create Agent Plugin as shown in Figure 4-2. This agent will recognize the boards connected to the computer via USB. [4]

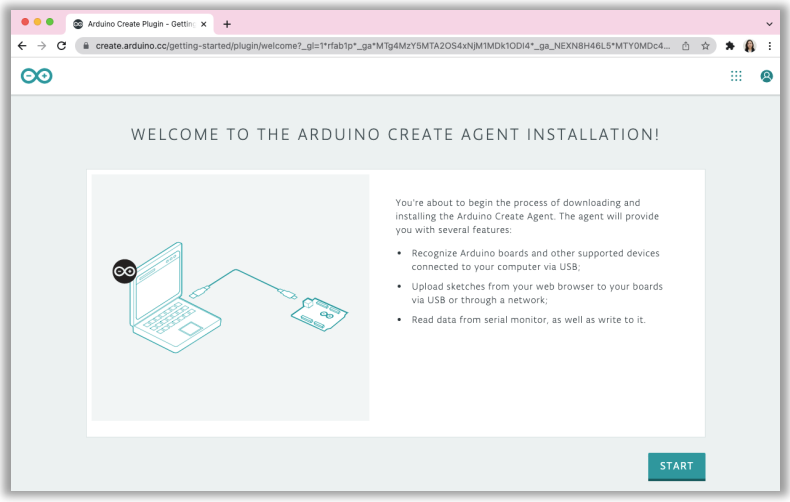


Figure 4-2 - Arduino Create Agent Installation

Once in the Web Editor page (Figure 4-3) we can see the board connected/disconnected, edit our code and upload when done.

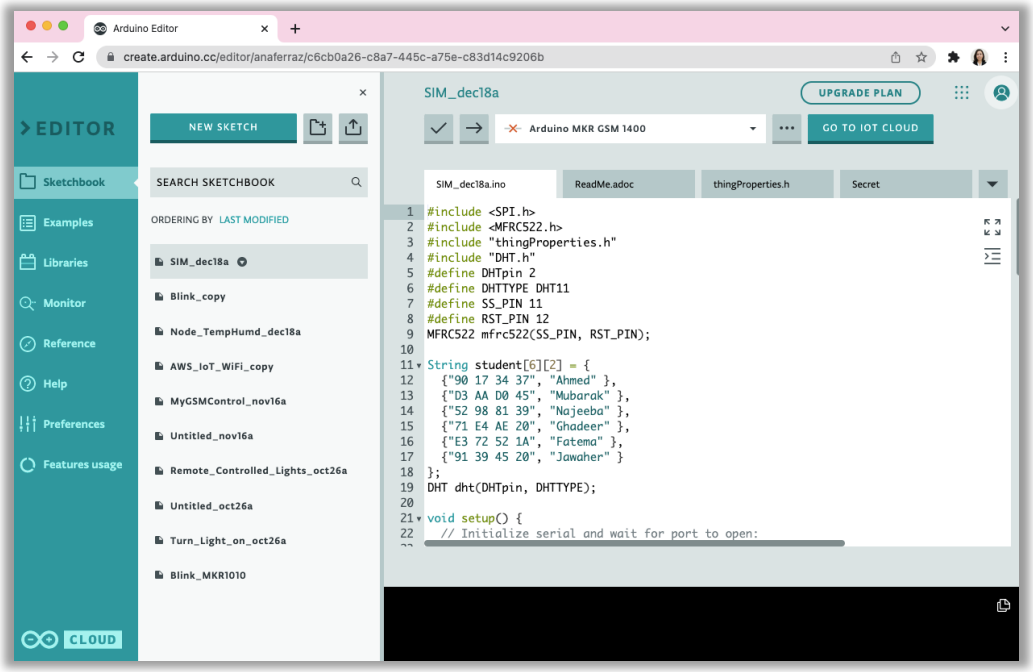


Figure 4-3 - Arduino Web Editor

However, there are other two options available:

- Classic offline Arduino IDE 1.8.13 (Integrated Development Environment) as in Figure 4-4– This was used for local sensor troubleshooting
- New Arduino IDE 2.0 – has new features like debugging, code highlighting and auto complete, which is currently in beta stage.

Figure 4-4 – Classic offline Arduino 1.8.18 installed on laptop

To select the IoT Cloud menu or Web Editor we can click at the top right button near the profile picture.

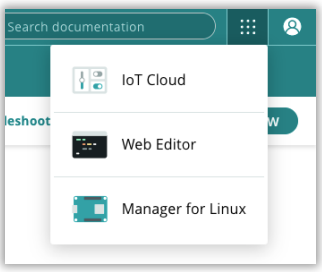


Figure 4-5 - IoT Cloud and Web Editor

Once we select the IoT cloud Menu, there are a few options available, but in this project, we will focus on:

- Things
- Dashboards
- Devices

Once we select Things on top right (Figure 4-5) the following menu shown on Figure 4-6 appears. You can click to Create Thing from scratch or select an existing Thing.

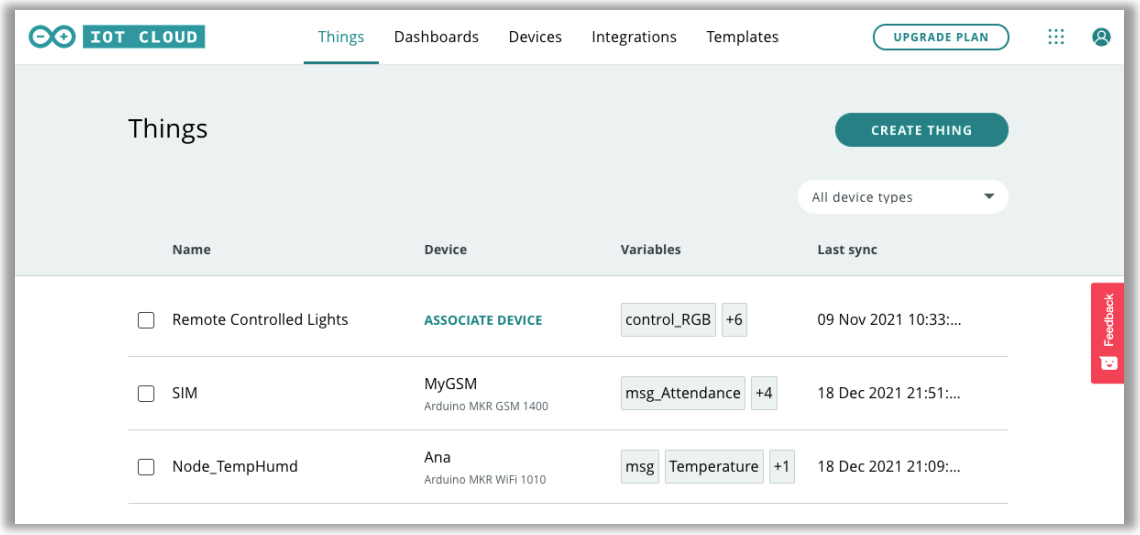


Figure 4-6 - Things Menu

This project's Thing is called SIM and we can see all components on Figure 4-7:

- Setup
  - Step 1 - Device - we can associate a device to our Thing
  - Step 2 - Add variables
  - Step 3 - Change the network settings
- Sketch - write the sketch or open the full editor, and connect to the Serial Monitor.
- Serial Monitor – Can be used to communicate with our device and troubleshoot our code

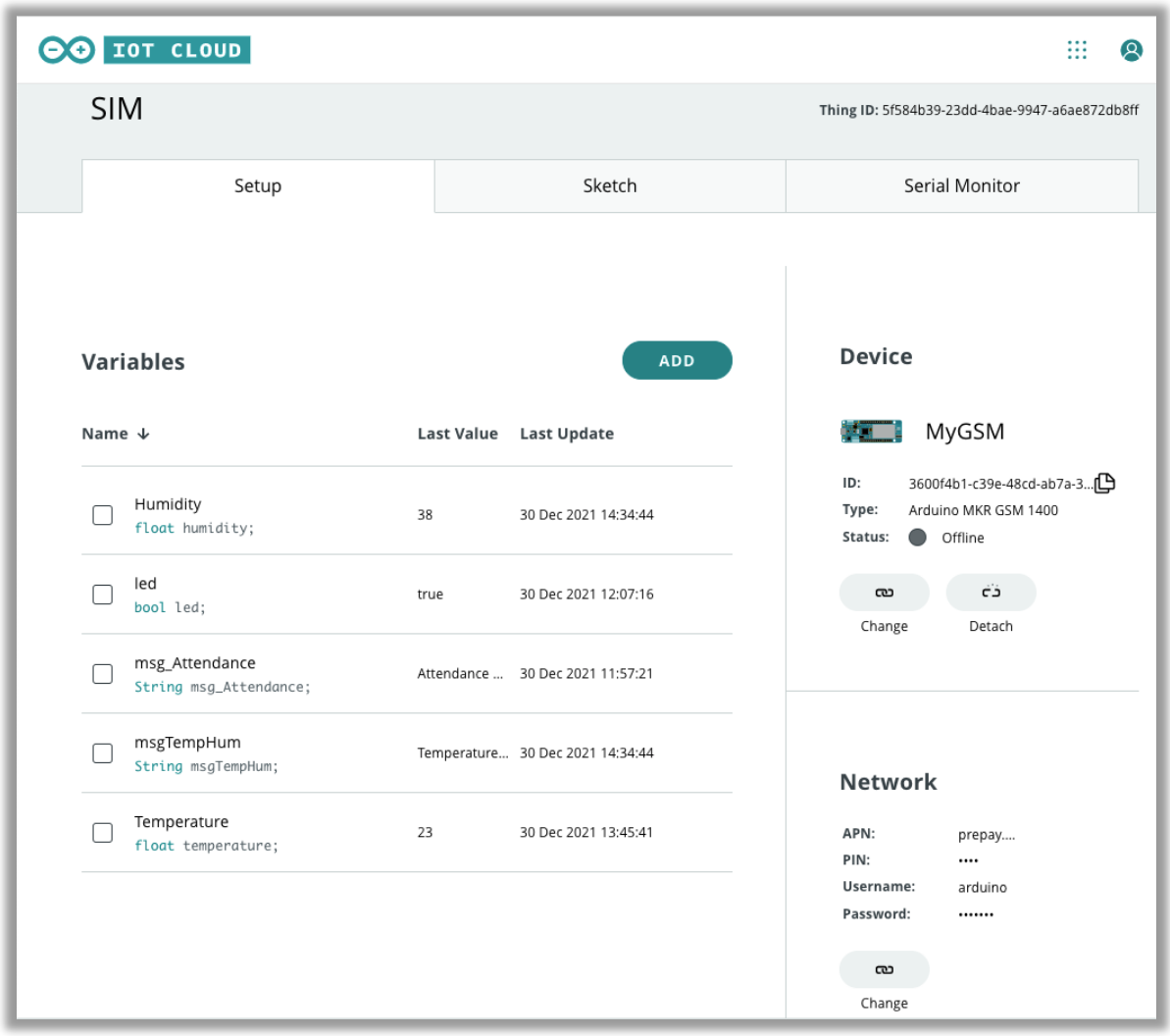


Figure 4-7 - My thing – SIM

A new Thing won't have a device configured, so the first step is to click and select device as displayed on Figure 4-8. Any device previously used in the IoT cloud can be associated or a new device can be set up as shown in Figure 4-9.

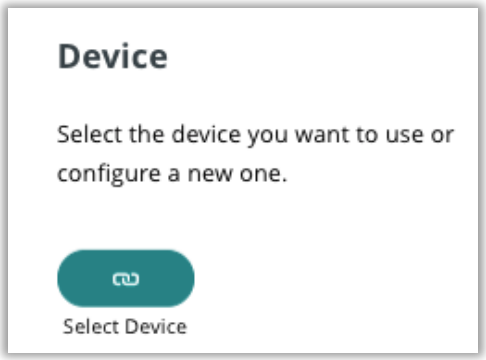


Figure 4-8 - Configure a new Device

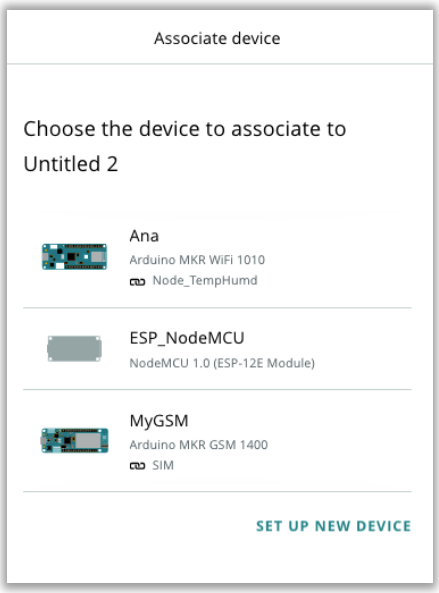


Figure 4-9 - Associate or set up a new device

For this project we associated the device Arduino MKR GSM 1400 to the Thing. The second step is to add the Variables as shown in Figure 4-10.

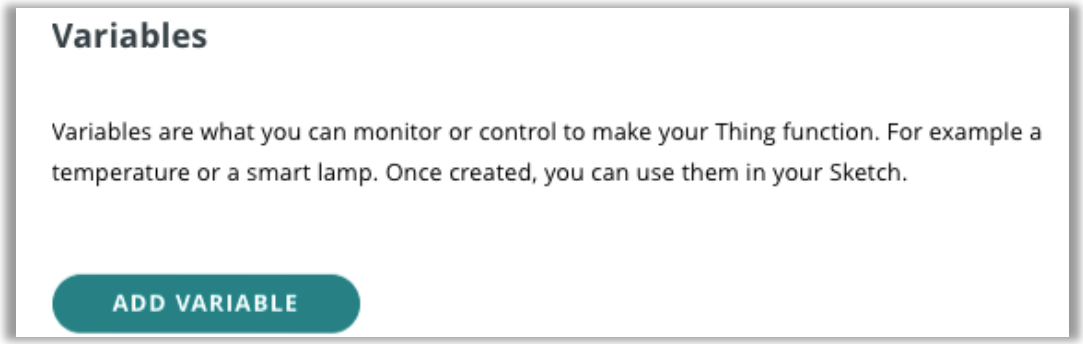


Figure 4-10 - Add a new variable

Once you click to add a variable, you need to select the variable name, type, permission, update policy and threshold. In this project 5 variables were created:

- Humidity – to store and display the relative humidity value on the dashboard
- Temperature – to store and display the room temperature on the dashboard
- msg_Attendance – to display the student attendance, name and time
- msgTempHum – to display the temperature and humidity and any warnings
- led – this LED was used for quick troubleshooting to check board/cloud connection

The variable permission can be:

- Read & Write – variable can work both as input and output, the data can be sent from the device to the cloud and vice versa
- Read only – variable can work only as output, the data can be sent only from the device to the cloud

The variable update policy can be:

- On Change: the variable will be updated to the cloud whenever the change in value is greater than or equal to the set threshold
- Periodically: the variable will be updated to the cloud each time the number of seconds set is elapsed

The basic variable types used in this project were:

- Boolean – true or false (LED)
- Floating point number – Numbers with decimals (temperature and humidity)
- Character String - words and sentences (msg_Attendance and msgTempHum)

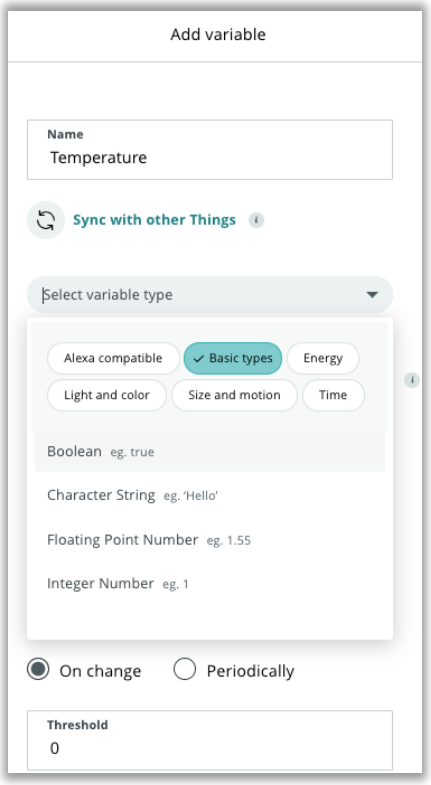


Figure 4-11 - Name and select variable type

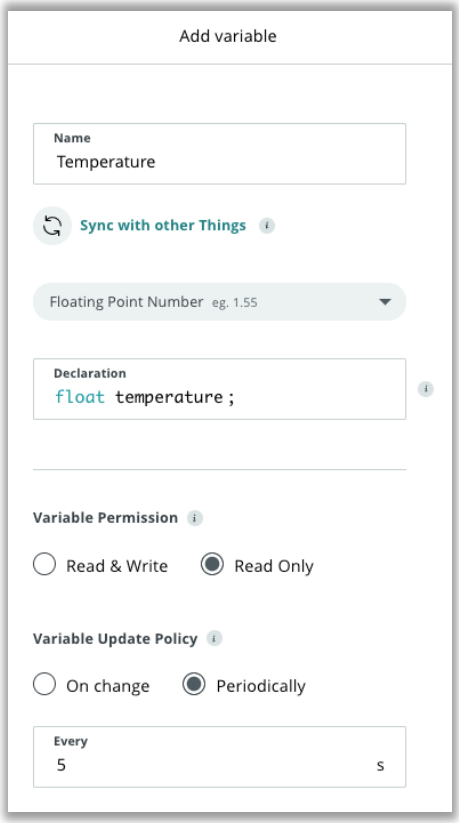


Figure 4-12 - Example of temperature variable of floating point number type

After setting up all variables, the third step is to add network credentials in the configure Network menu as shown in Figure 4-13.

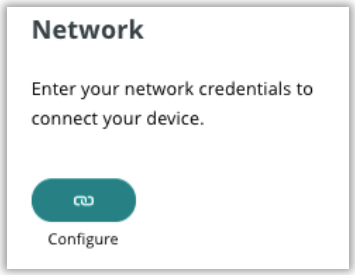


Figure 4-13 - Network configuration

In this project we used the Arduino SIM card that has credentials as shown in Figure 4-15. In the project proposal the board suggested was the Arduino MRK Wi-Fi 1010. In case that device is selected the network configuration will be different as shown in Figure 4-14. The local Wi-Fi name must not include spaces.

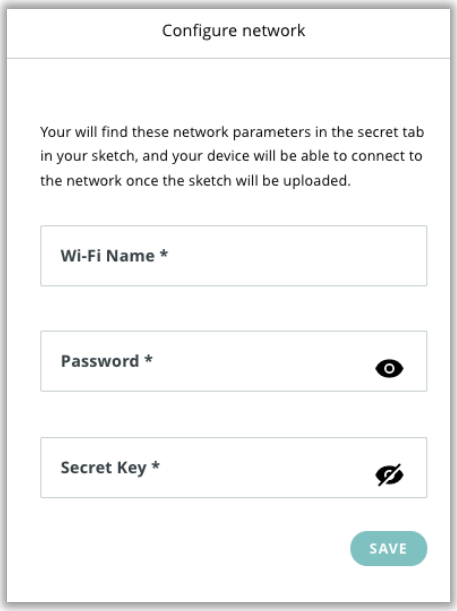


Figure 4-14 – Wifi Network configuration

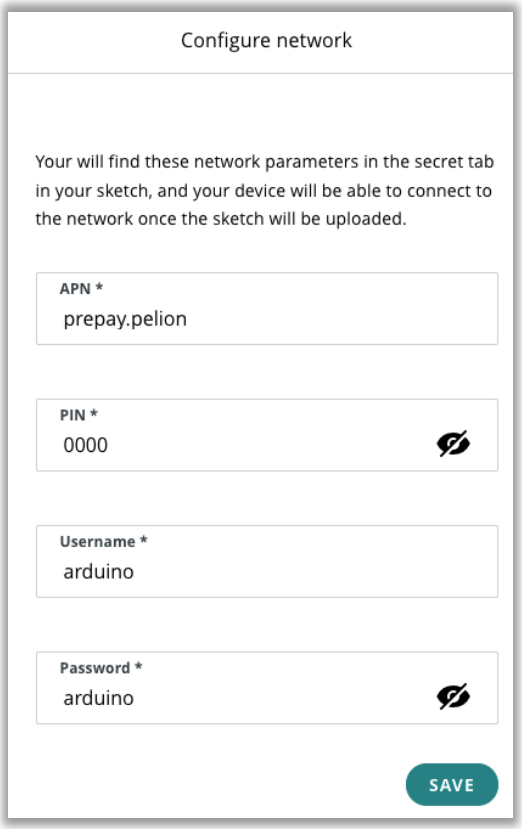


Figure 4-15 – GSM Network Configuration

After setting up device, adding variables and configuring the network, the sketch is added as shown in Figure 4-16. The sketch can be edited at this tab but the full editor has more functionalities and was used in this project. A small part of the code is automatically updated by Arduino IoT Cloud based on the information added in the first three steps.

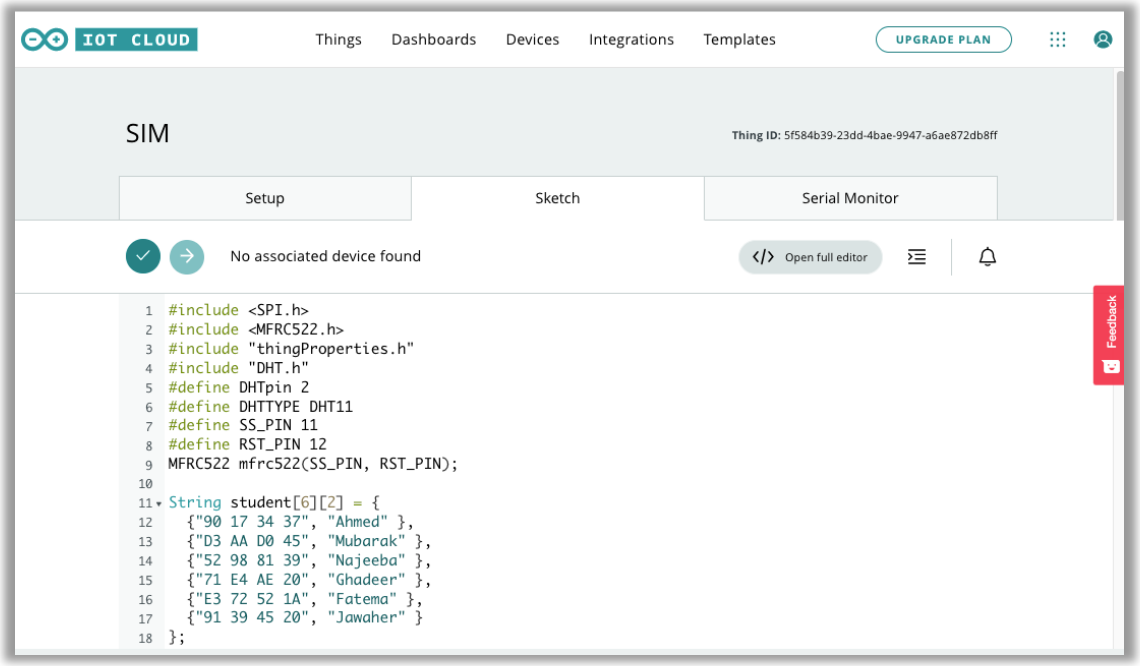


Figure 4-16 - Editing sketch tab or open full editor

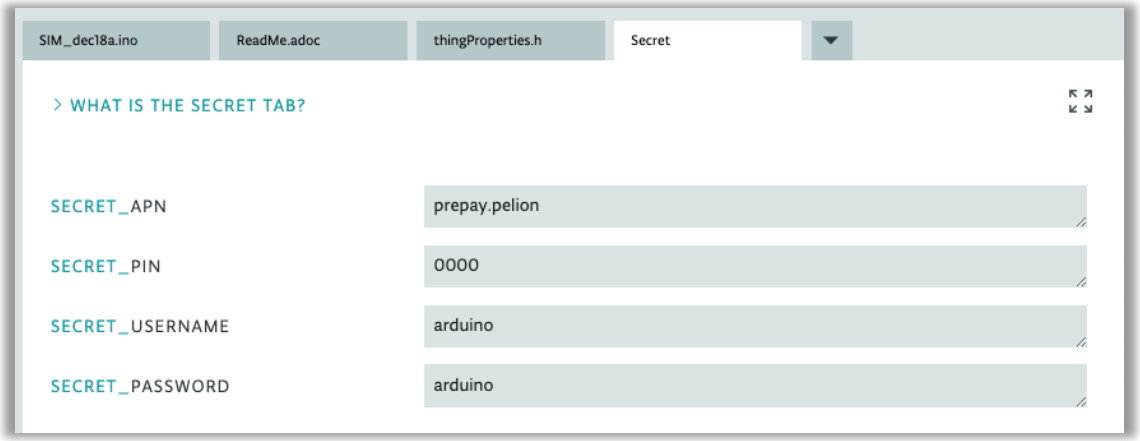


Figure 4-17 - SIM (Subscriber Identity Module) information

Figure 4-18 - Serial Monitor Test Messages

4.2 Dashboard

The dashboard is the last part of the IoT Cloud set up and we can click to build a dashboard as shown in Figure 4-19:

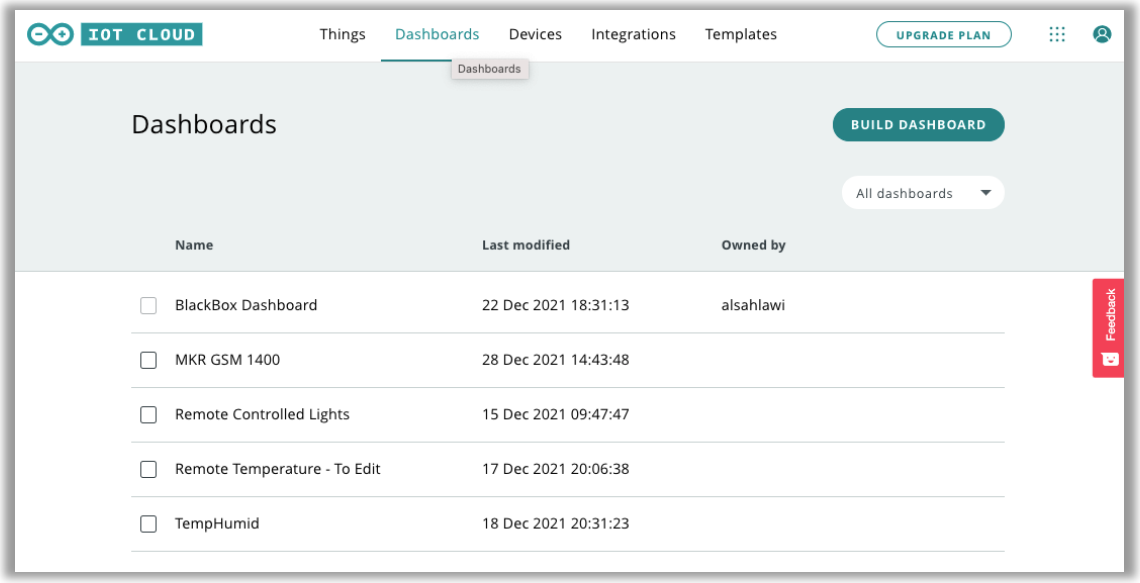


Figure 4-19 – Dashboards' Menu

To populate our dashboard, we need to add widgets as shown in Figure 4-20 or add things and then select the variables as shown in Figure 4-21:

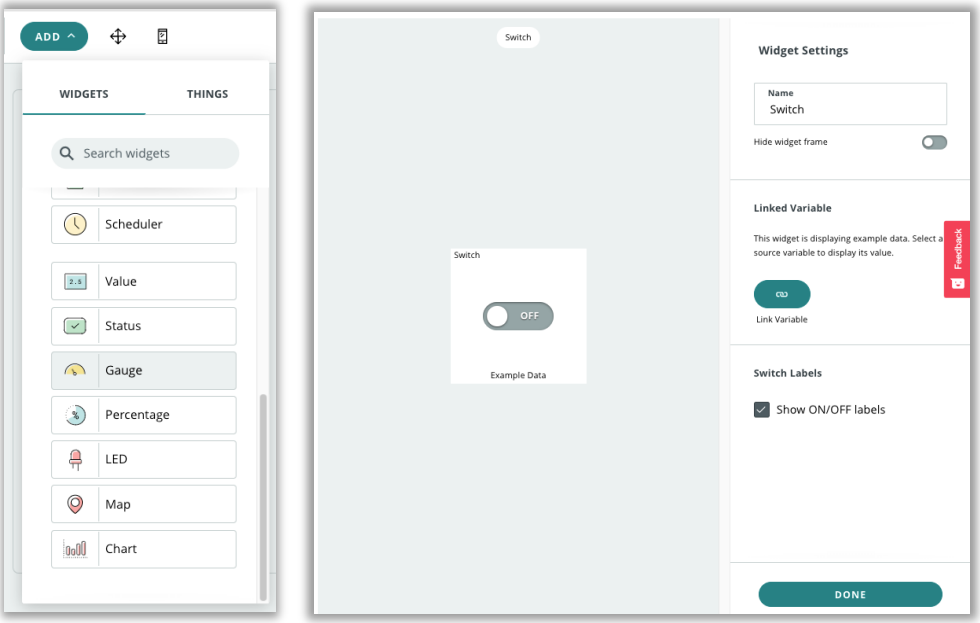


Figure 4-20 - Adding widets to dashboard

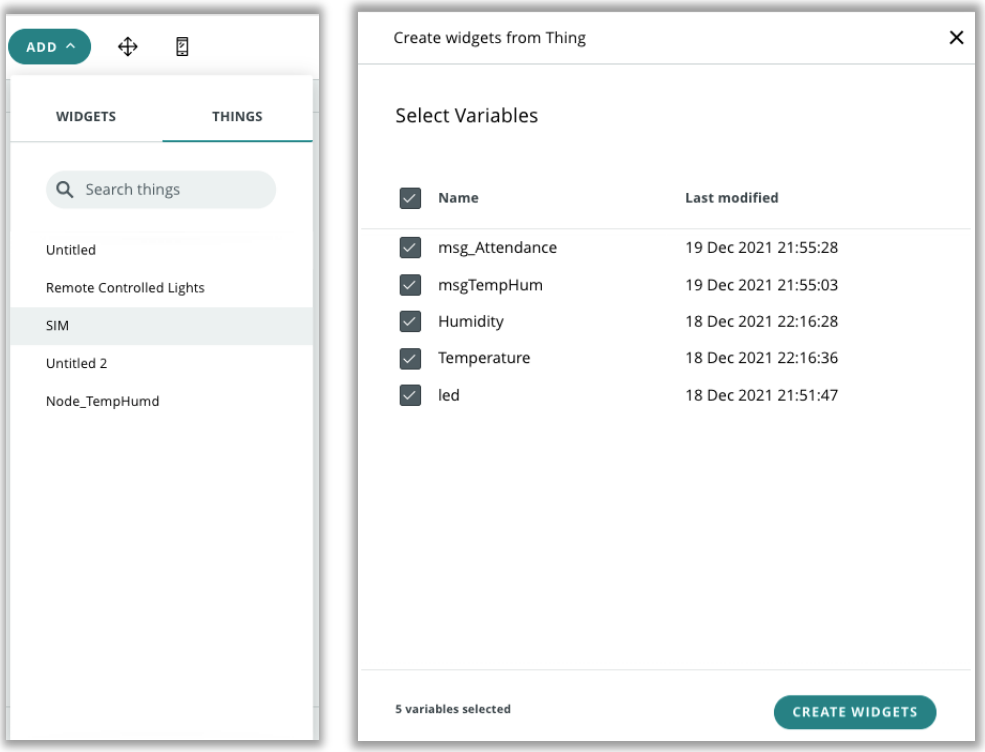


Figure 4-21 - Creating widgets from variables

In this project we added seven widgets, LED button, Humidity and Temperature gauges, Humidity and Temperature charts, temperature and humidity message and attendance message. There are two type of dashboard views: the mobile view (Figure 4-22) and desktop view (Figure 4-23).

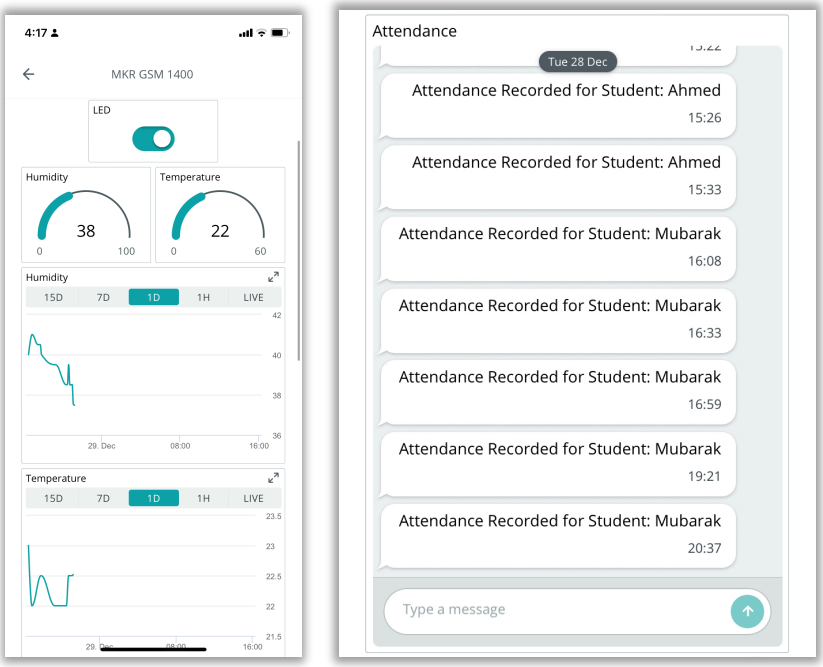


Figure 4-22 - Dashboard for Mobile View

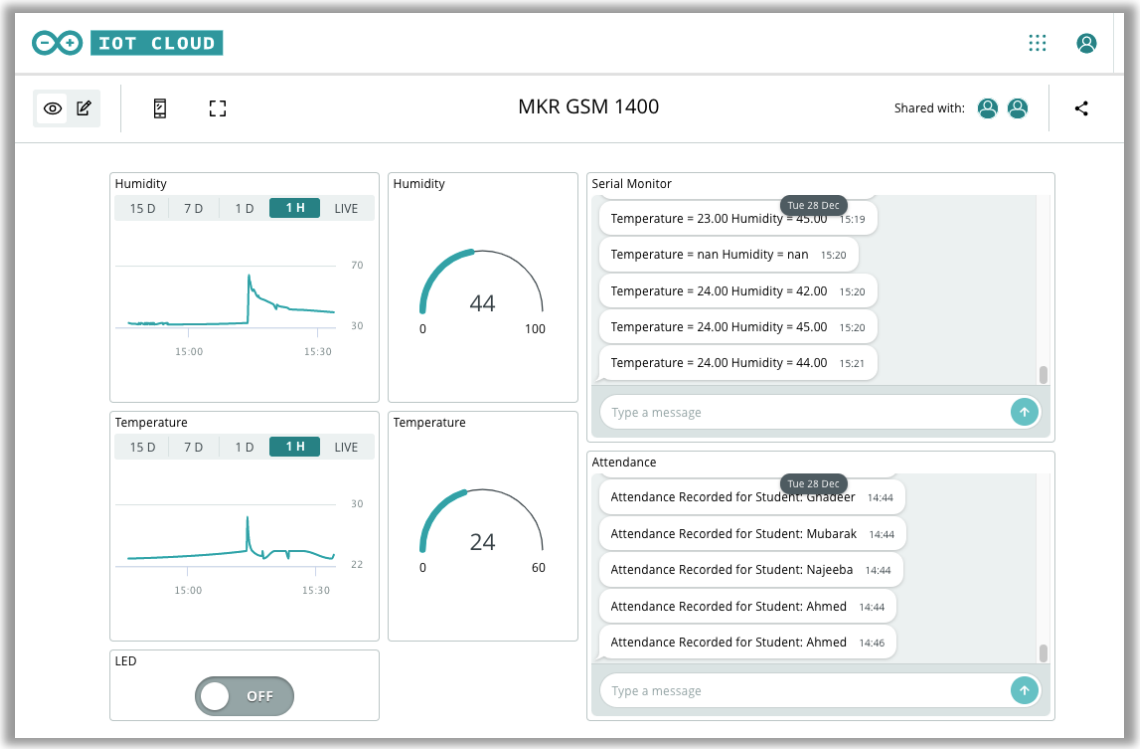


Figure 4-23 – Final Arduino IoT Cloud Dashboard

4.3 Exporting data

The data can be exported from the cloud using the Download Historic Data option (Figure 4-24) at the dashboard.

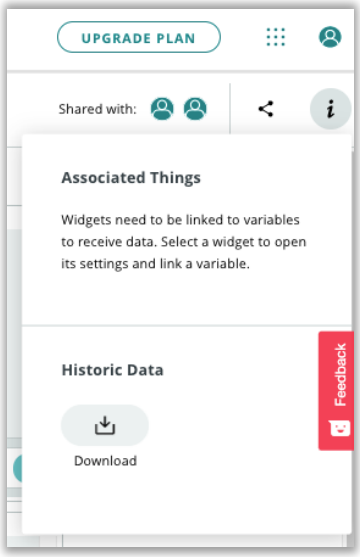


Figure 4-24 - Download Historic Data

The download historic data has the option of selecting which variables we want to download, and the period as shown in Figure 4-25. This is useful if we want to use the data for further analysis.

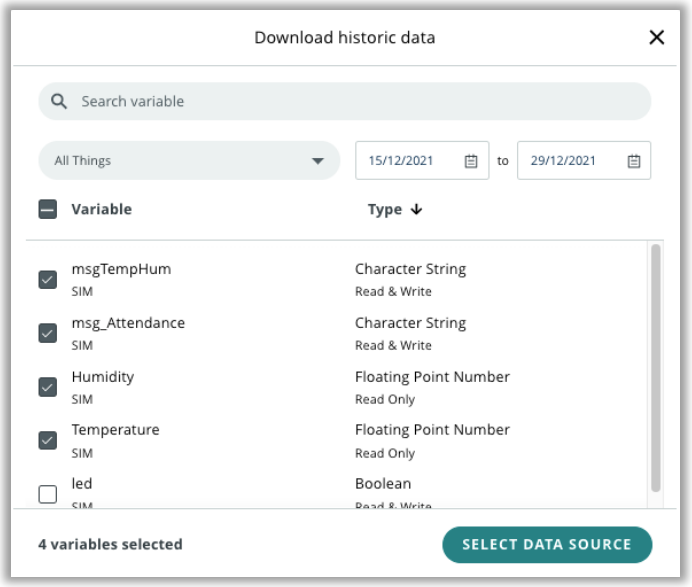


Figure 4-25 - Selecting variables to download history

In Figure 4-26 you can see an example of the attendance data that was received by email and downloaded as CSV file. The readme.txt file contains the variable names, period requested and a message wishing us to “Have fun!”.

	1	2	3	4	5
126	2022-01-09T16:07:22.433Z	Attendance Recorded for Student: Ahmed			
127	2022-01-09T16:07:23.225Z	Attendance Recorded for Student: Ahmed			
128	2022-01-09T16:26:03.140Z	Attendance Recorded for Student: Mubarak			
129	2022-01-09T16:26:10.696Z	Attendance Recorded for Student: Najeeba			
130	2022-01-09T16:26:26.186Z	Attendance Recorded for Student: Jawaher			
131	2022-01-09T16:26:28.228Z	Attendance Recorded for Student: Fatema			
132	2022-01-09T16:26:31.281Z	Attendance Recorded for Student: Ghadeer			
133	2022-01-09T16:27:06.421Z	Test			
134	2022-01-09T16:27:09.769Z	Test			
135	2022-01-09T16:27:45.005Z	Attendance Recorded for Student: Fatema			
136	2022-01-09T16:27:54.252Z	Attendance Recorded for Student: Jawaher			
137	2022-01-09T16:27:59.020Z	Attendance Recorded for Student: Ghadeer			
138	2022-01-09T16:28:04.421Z	Attendance Recorded for Student: Ahmed			
139	2022-01-09T16:28:07.102Z	Attendance Recorded for Student: Mubarak			
140	2022-01-09T16:29:26.657Z				
141	2022-01-09T16:29:29.322Z				
142	2022-01-09T16:49:51.380Z	Attendance Recorded for Student: Ahmed			
143	2022-01-09T16:51:19.286Z	test			
144	2022-01-09T16:51:21.786Z	Attendance Recorded for Student: Ahmed			
145	2022-01-09T16:51:22.617Z	Attendance Recorded for Student: Ahmed			
146	2022-01-09T16:51:32.956Z	Attendance Recorded for Student: Mubarak			
147	2022-01-09T16:51:36.992Z	Attendance Recorded for Student: Ahmed			
148	2022-01-09T16:51:46.873Z	Attendance Recorded for Student: Ghadeer			
149	2022-01-09T16:51:51.442Z	Attendance Recorded for Student: Fatema			
150	2022-01-09T16:51:55.988Z	Attendance Recorded for Student: Jawaher			

readme.txt

```
Arduino IoT Cloud historic data

variables:
  - id: 378ad338-abcc-45d7-aac8-fd1f16bd6557
    name: msg_Attendance
    thingName: SIM
from: 2021-12-26T00:00:00Z
to: 2022-01-09T23:59:59Z

Have fun! :)
```

Figure 4-26 – Example of CSV historical data received by email

CHAPTER 5
Conclusion

In this project, a prototype of an automatic attendance system and classroom environmental monitoring was created. This IoT application in education can save valuable time to the teacher and make sure the classroom environment is optimum for the teaching and learning experience. The RFID RC522 module was used to implement the automatic attendance and the DHT11 was used to monitor the relative humidity and temperature of classroom. In case the values were out of range, it informed the user with a message. This could be improved in the future by implementing an alarm or an actuator to turn on/off the air conditioner. Both sensors were connected to an Arduino MKR GSM 1400 and data was sent to Arduino IoT cloud.

The initial plan mentioned in the project proposal was to use an Arduino MKR Wi-Fi 1010, but it was damaged unfortunately while setting up the AWS cloud. Before the damage, there were some issues with the Wi-Fi connection because the network had spaces in the name. It was decided to replace the Wi-Fi to GSM and AWS to Arduino IoT Cloud. Some of the challenges faced included the RFID and DHT working separately but not together. After troubleshooting locally using the serial monitor, the prototype wouldn't connect the cloud. After investigating all possibilities, the issue was pin 10 used in SPI communication for the RFID module. The Arduino IoT Cloud update would break the SPI communication and the RFID module wouldn't work. The solution implemented was to move the SPI.begin() function to the loop to reset all the pins before the ArduinoCloud.update() function.

The project was published at the Arduino Project Hub [8] for easy visualisation and knowledge sharing. Many of the lessons learned in this Project will be used in my teaching practice. I have plans to introduce IoT applications in my embedded systems course and suggest final year project to students.

References

[1] J. Faritha Banu, R. Revathi, M. Suganya and N. R. Gladiss Merlin, "IoT based Cloud integrated smart classroom for smart and a sustainable campus," *Procedia Computer Science,* vol. 172, pp. 77-81, 2020.

[2] M. K. Saini and N. Goel, "How smart are smart classrooms? A review of smart classroom technologies," *ACM Computing Surveys,* vol. 52, no. 6, 12 2019.

[3] S. Trilles, P. Juan, S. Chaudhuri and A. B. V. Fortea, "Data on CO2, temperature and air humidity records in Spanish classrooms during the reopening of schools in the COVID-19 pandemic," *Data in Brief,* vol. 39, 12 2021.

[4] Arduino, "MKR GSM 1400 Overview," [Online]. Available: https://docs.arduino.cc/hardware/mkr-gsm-1400. [Accessed Dec 2021].

[5] Aosong, "Datasheet DHT11," [Online]. Available: https://www.electronicoscaldas.com/datasheet/DHT11_Aosong.pdf. [Accessed Dec 2021].

[6] NXP, "MFRC522 datasheet," [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf. [Accessed Dec 2021].

[7] Arduino, "Forum," [Online]. Available: https://forum.arduino.cc/t/arduinocloud-update-doesnt-work-for-me/633852/4. [Accessed Dec 2021].

[8] A. Ferraz, "Automatic Attendance and Classroom Environmental Monitoring," [Online]. Available: https://create.arduino.cc/projecthub/anaferraz/automatic-attendance-and-classroom-environmental-monitoring-2496bd. [Accessed Jan 2022].