



Assembly of circuit with Keypad for car and sending IoT data

Difficulty level: Difficult

Goals

Automotive IoT is the integration of gadgets, sensors, cloud computing, applications, and other such components into vehicles to function as a complex system for the connection of cars, predictive maintenance, fleet management, OEMs, insurance, and more.

The integration of the Internet of Things in the automotive industry allows manufacturers to implement sought-after innovations that can ultimately transform cars into near-artificial intelligence. At a didactic level, we are now going to develop some exercises using sensors for data acquisition, processed by the Arduino microcontroller.

This exercise intends to apply a numerical keyboard guided by a microcontroller to block or unblock an electromechanical system. This exercise is aided by a red led that indicates system blocked and a green led that indicates system unlocked.

For the possible sending of data, it will be necessary to apply, for example, the ESP8266 ESP-01 module that allows the connection of several devices to the internet (or local network), and consequent sending of data from the sensors applied to the autonomous system.

Image-1: Understanding the application of Keypad in a car and communicating with IoT.

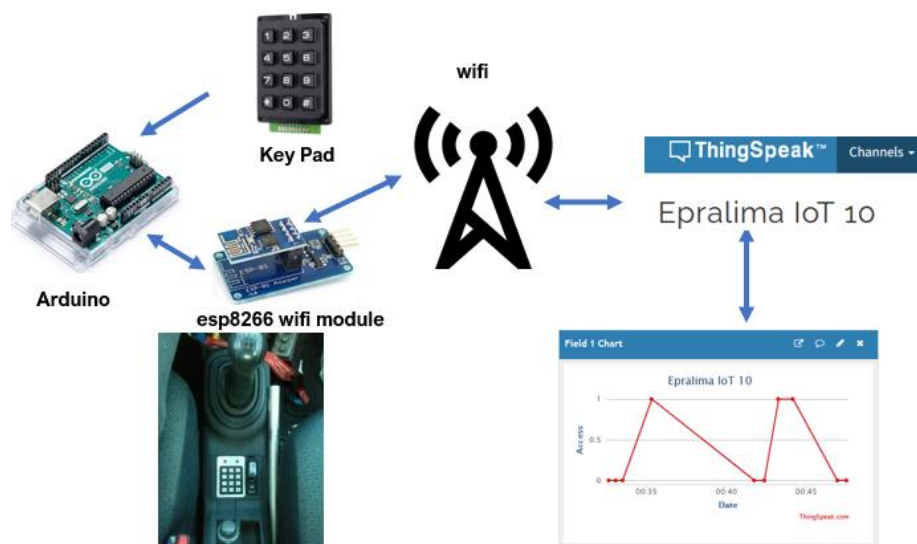


Image 1: application of Keypad in a car and communicating with IoT



Skills

- The skills our students will gain are:
- Students' ability to build circuits will be developed.
- The ability to program the Arduino board and use the ESP8266 Module for Internet access will develop.
- The ability to receive data from the brightness sensor and send the received data to Thing Speak will be gained.
- Data analytics will improve their ability to connect with the Internet of Things.

Required materials and circuit diagram.


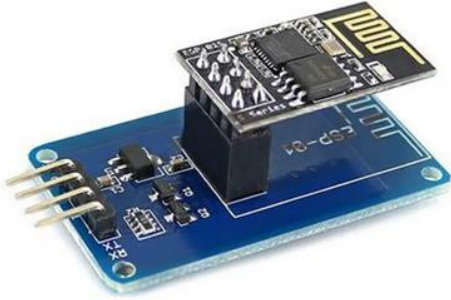
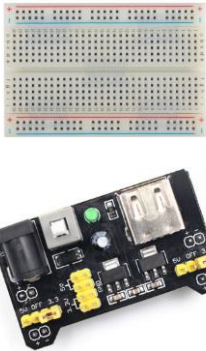

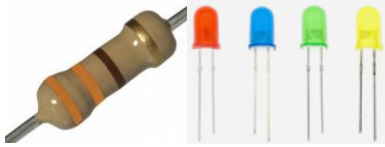

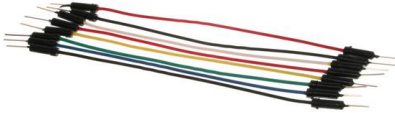
In this exercise we intend to learn how to draw diagrams (circuits), connect all the components correctly, develop software based on C language (Arduino), connect to the wifi network, communicate with an IoT server, ThingSpeak and read server-generated graphics.

Quantity	Component
1	Arduino Uno R3
1	ESP01-8266
1	Power Supply (braedBoard)
1	BreadBoard
1	Key Pad
1	Servo Motor
2	Led Green, Red and Blue
2	Resistor 330Ohm

Table 1 - Components List



Materials table

	
Arduino	ESP01 - 8266
	
Bread Board + Power Supply	Key PAD
	
330 Ω Leds	Servo Motor
	
Jumper wire	

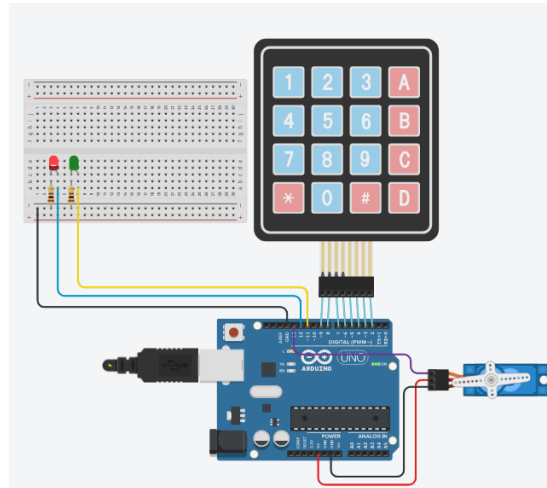


Image 2 – Diagram circuit

Implementation

Development of communication of microcontroller systems, and sensors, with the ThingSpeak IoT cloud.

The ESP8266 WiFi module (image 3) is a small shield with integrated TCP/IP protocol that can give any microcontroller access to the WiFi network. The ESP8266 is capable of both hosting an application and offloading all WiFi network functions from another application processor. Each ESP8266 module is pre-programmed with an AT command making its firmware settings, meaning that we can simply connect this module to the Arduino working as any other WiFi shield would. This module has a great cost/benefit ratio and has a very large and constantly growing user community.

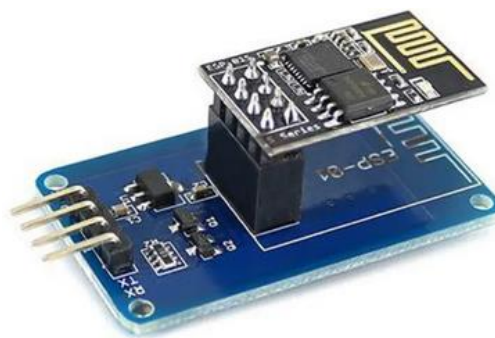


Image 3 - ESP01 – 8266



A basic 12 button keypad for user input. The buttons are setup in a matrix format. This allows a microcontroller to 'scan' the 7 output pins to see which of the 12 buttons is being pressed.



Image Key Pad

Implementation in practice

1. Assemble the circuit in the image 2;
2. Connect correctly ESP01-8266 image 5

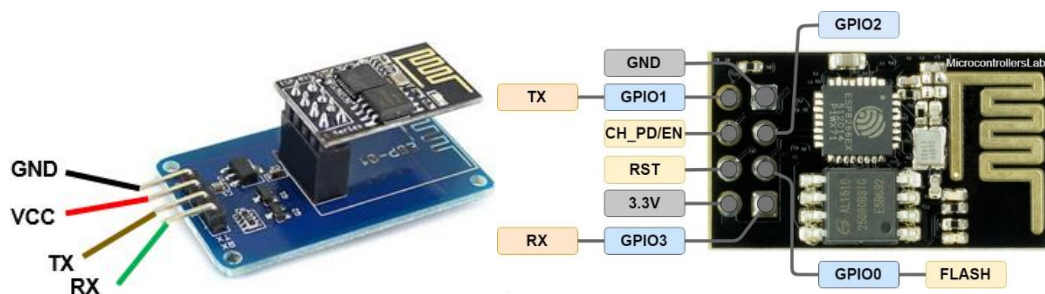


Image 4 ESP-01 Connections



3. Real assembled circuit image 6

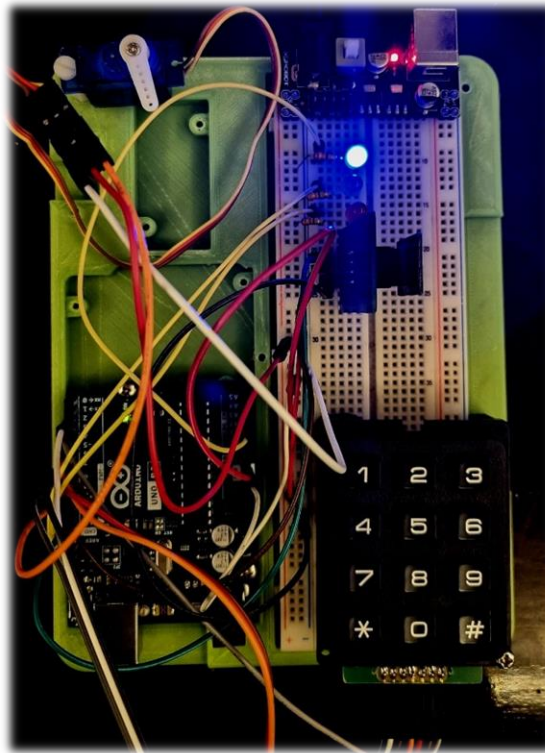


Image 5 Real circuit in breadboard

4. Create a ThingSpeak account image 7

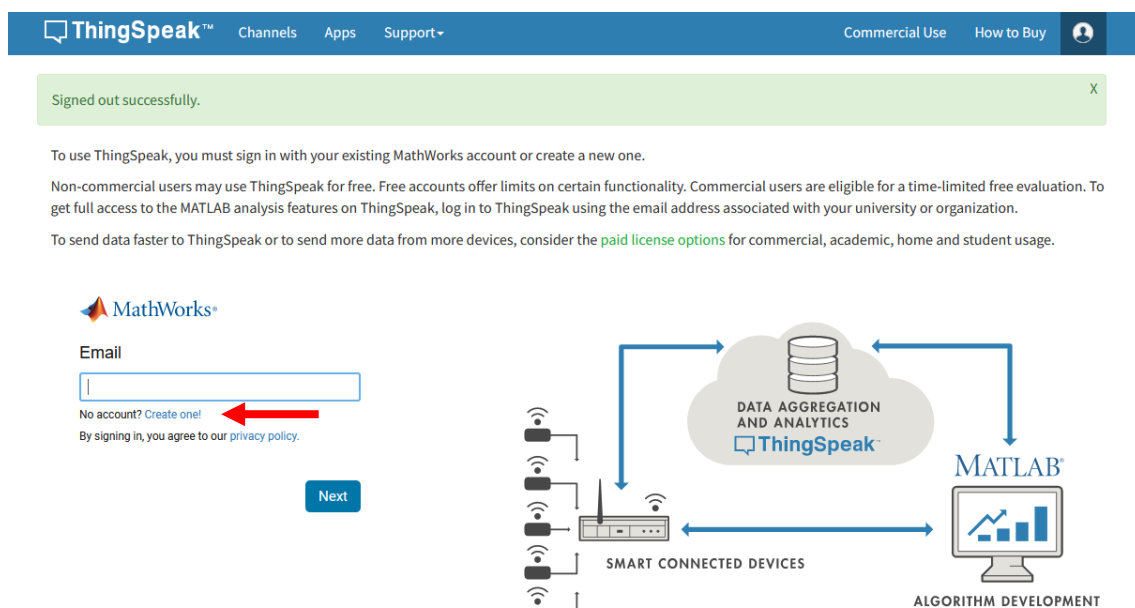


Image 6 - Thing Speak



5. Create a new channel image 8

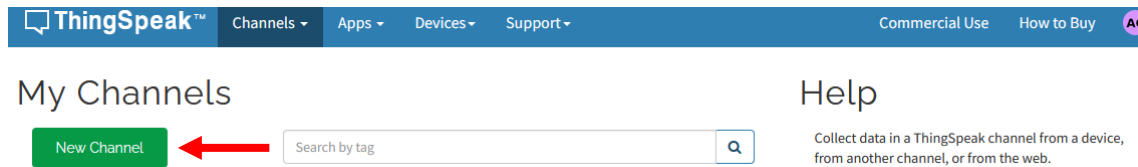


Image 7 Interface ThingSpeak

6. Configure channel, with name, description and fields. Image 9.

Note: The fields refer to data processed by the microcontroller and data from the sensors under study. Each field will generate a graph.

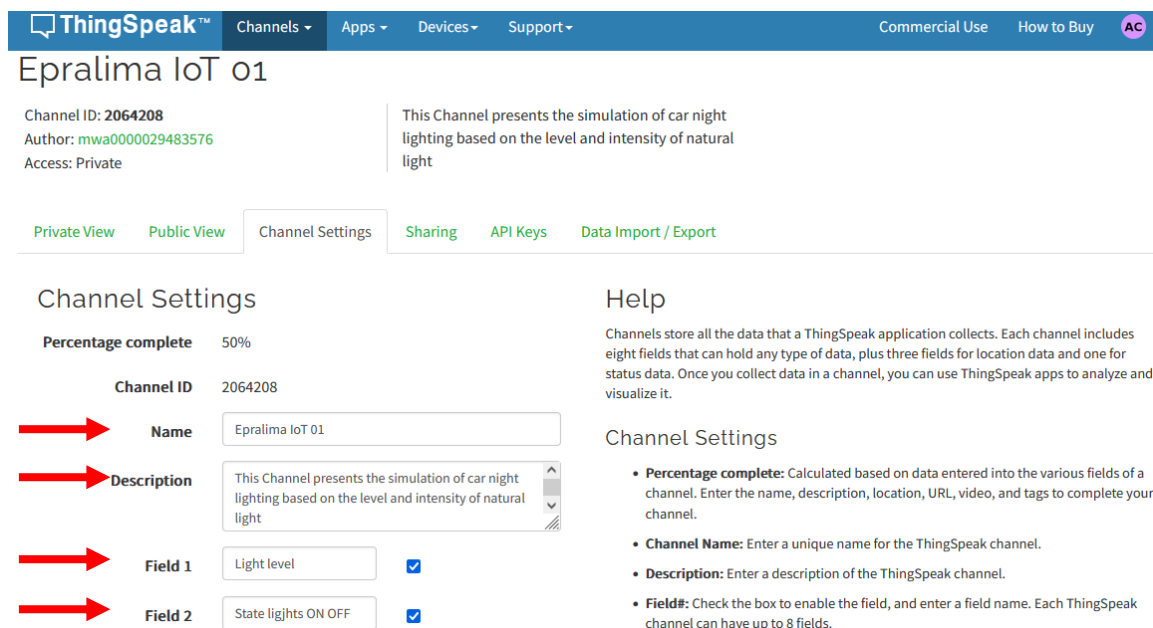


Image 8 Configure Channel



7. Save settings channel Image 10

ThingSpeak™ Channels Apps Devices Support Commercial Use How to Buy AC

station that acquires data from an Arduino® device. [Learn More](#)

Link to GitHub

Elevation

Show Channel Location ☐

Latitude

Longitude

Show Video ☐

☒ YouTube ☐ Vimeo

Video URL

Show Status ☐

[Save Channel](#)

Image 9 Save settings channel.

8. In this step, we will pay special attention to the api keys, as they are the ones that, through the string key, will allow access to the IoT repository in arduino programming. Also very important are the API requests.

ThingSpeak™ Channels Apps Devices Support Commercial Use How to Buy AC

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Write API Key

[Key](#)

[Generate New Write API Key](#)

Read API Keys

[Key](#)

Note

[Save Note](#) [Delete API Key](#)

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- **Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

API Requests

Write a Channel Feed

```
GET https://api.thingspeak.com/update?api_key=UC[redacted]CP&field1
```

Read a Channel Feed

```
GET https://api.thingspeak.com/channels/[redacted]feeds.json?api_key=D8
```

Image 10 - API Keys



9. Programming Arduino

Inclusion of the necessary libraries and declaration of variables and constants inherent to the program's operation.

```
1 #include <SoftwareSerial.h>
2 #include <Keypad.h>
3 #include<Servo.h>
4 SoftwareSerial ESP_Serial(10, 11); //PINOS QUE EMULAM A SER
5                                     // RX_AUX  --liga--> TX dc
6                                     // TX_AUX  --liga--> RX dc
7 //Port 2 Ard pin 2 keypad --> Line
8 //Port 3 Ard pin 7 keypad --> Line
9 //Port 4 Ard pin 6 keypad --> Line
10 //Port 5 Ard pin 4 keypad --> Line
11 //Port 6 Ard pin 3 keypad --> Column
12 //Port 7 Ard pin 1 keypad --> Column
13 //Port 8 Ard pin 5 keypad --> Column
14
15 #define serialcomSpeed 115200
16 #define DEBUG true
17 #define redLed 12
18 #define greenLed 13
19 #define blueLed A1
20 #define serv 9
21
22 byte pinosLines[] = {2,3,4,5}; //Port 2 ard pin 1 keypad
23 byte pinosColumns[] = {6,7,8};
24
25 char teclas[4][3] = {{ '1', '2', '3'},
26                      { '4', '5', '6'},
27                      { '7', '8', '9'},
28                      { '*', '0', '#' }};
29
30 Keypad tecladol = Keypad( makeKeymap(teclas), pinosLines, pinosColumns, 4, 3);
31 Servo servomotor1;
32 String pin="";
33 String accessPin = "12345";
34 unsigned long agora = 0;
35 int access = 0;
36
37 long writingTimer = 17;
38 long startTime = 0;
39 long waitTime = 0;
40
41 boolean error;
42 String APIKey = "4          I";
43 void readKeyPad(void);
44 void readPinCode(char);
```



Void setup() function for initializing parameters for starting the program.

```
48 void setup() {
49   Serial.begin(serialcomSpeed);
50   ESP_Serial.begin(serialcomSpeed);
51   startTime = millis();
52   InitWifiModuleESP();
53   Serial.println("Teclado 4x4 com Biblioteca Keypad");
54   Serial.println("Aguardando acionamento das teclas...");
55   Serial.println();
56   pinMode(2, INPUT);
57   pinMode(3, INPUT);
58   pinMode(4, INPUT);
59   pinMode(5, INPUT);
60   pinMode(6, INPUT);
61   pinMode(7, INPUT);
62   pinMode(8, INPUT);
63   pinMode(redLed, OUTPUT);
64   pinMode(greenLed, OUTPUT);
65   pinMode(blueLed, OUTPUT);
66   analogWrite(blueLed, 255);
67   servomotor1.attach(serv);
68   delay(500);
69   servomotor1.write(10);
70   // -----
71 }
```

AT commands

AT commands are the basic way to configure and trigger the ESP8266 when it is under control of an external device (like an Arduino, for example).

Current AT commands are direct descendants of the so-called "Hayes Standard" from 1981, used to allow personal computers to interact with telephone connections by directly controlling a mode.

The **InitWifiModule()** function initializes the ESP8266 through AT commands.

```
46 void InitWifiModuleESP() {
47   //Este procedimento envia os COMANDOS AT para o ESP 01
48   envioDadosESP_AT("AT+RST\r\n", 2000, DEBUG); //faz reset ao modulo;
49   envioDadosESP_AT("AT+CWMODE=1\r\n", 1500, DEBUG);
50   delay(100);
51   envioDadosESP_AT("AT+CWJAP=\"Epralima\", \"*****\r\n\", 2000, DEBUG);
52   delay(500);
53   envioDadosESP_AT("AT+CIFSR\r\n", 1500, DEBUG);
54   delay(100);
55   envioDadosESP_AT("AT+CIPMUX=0\r\n", 1500, DEBUG);
56   delay(100);
57 }
```



The **envioDadosESP_AT(str,int,boolean)** function is responsible for sending AT commands to the ESP8266

```
59 String envioDadosESP_AT(String comando, const int timeout, boolean debug)
60 {
61     String resposta = "";
62     ESP_Serial.println(comando);
63     long int tempo = millis();
64     while((tempo+timeout) > millis()){
65         while(ESP_Serial.available()){
66             char c = ESP_Serial.read();
67             resposta+=c;
68         }
69     }
70     if(debug){
71         Serial.print(resposta);
72     }
73     return resposta;
74 }
```

The **startThingSpeakCmd(str,int,boolean)** function opens connection to ThingSpeak IoT analytics platform. The IP address of the ThingSpeak platform is: 184.106.153.149 with connection on port 80. The AT command to start ThingSpeak communication is AT+CIPSTART=PROTOCOL, IP_ADRESS, PORT.

```
106 void startThingSpeakCmd(void) {
107     ESP_Serial.flush();
108     String cmd="";
109     cmd = "AT+CIPSTART=\"TCP\", \"";
110     cmd+="184.106.153.149"; //Endereco IP thingerSpeak
111     cmd+="\", 80";
112     ESP_Serial.println(cmd);
113     Serial.print("Start Commands: ");
114     Serial.println(cmd);
115     if(ESP_Serial.find("Error"))
116     {
117         Serial.println("AT+CIPSTART error");
118         return;
119     }
120 }
```



The **EscreverParaThingSpeak** function generates a string to build an API Request.

Example:

GET /update?api_key=U.....P&field1= 0&field2= 0

```
140 void EscreverParaThingSpeak(void) {
141
142     startThingSpeakCmd();
143     String getStr = "";
144     getStr = "GET /update?api_key=";
145     getStr += APIKey;
146     getStr += "&field1=";
147     getStr += String(access);
148     getStr += "\r\n\r\n";
149     GetThingSpeakcmd(getStr);
150 }
```

The **GetThingSpeak(str)** function, is responsible for determining and sending an API Request through the AT+CIPSEND command to write to the ThingSpeak channel, returning the message received by the response from the ThingSpeak data platform. The communication will be closed if the response is not favourable.

```
122 String GetThingSpeakcmd(String getStr) {
123
124     String command="";
125     command = "AT+CIPSEND=";
126     command += String(getStr.length());
127     ESP_Serial.println(command);
128     Serial.println(command);
129     int result = ESP_Serial.find(">");
130     if(result==1)
131     {
132         Serial.print("String GET--> ");
133         Serial.println(getStr);
134         ESP_Serial.print(getStr);
135         Serial.println(getStr);
136         delay(500);
137         String messageBody = "";
138         String linha="";
139         while(ESP_Serial.available()){
140             linha = ESP_Serial.readStringUntil('\n');
141             if(linha.length() == 1){
142                 messageBody = ESP_Serial.readStringUntil('\n');
143             }
144         }
145         Serial.print("MessageBody received: ");
146         Serial.print(messageBody);
147         return messageBody;
148     }
```



The **readKeypad ()** and **readPinCode()** These functions read the pin in keypad.

```
152 void readKeyPad(void) {
153   char tecla_pressionada;
154   String pwd;
155   pin="";
156   Serial.println("Your code: ");
157   do{
158     tecla_pressionada = teclado1.getKey();
159     readPinCode(tecla_pressionada);
160   }while(tecla_pressionada!='#');
161 }
162
163 void readPinCode(char c) {
164   switch(c) {
165     case '1':
166     case '2':
167     case '3':
168     case '4':
169     case '5':
170     case '6':
171     case '7':
172     case '8':
173     case '9':
174     pin+=c;
175     break;
176   }
177 }
```

Results

Analysing the graphics, it is possible to observe the access a electro mechanic system.

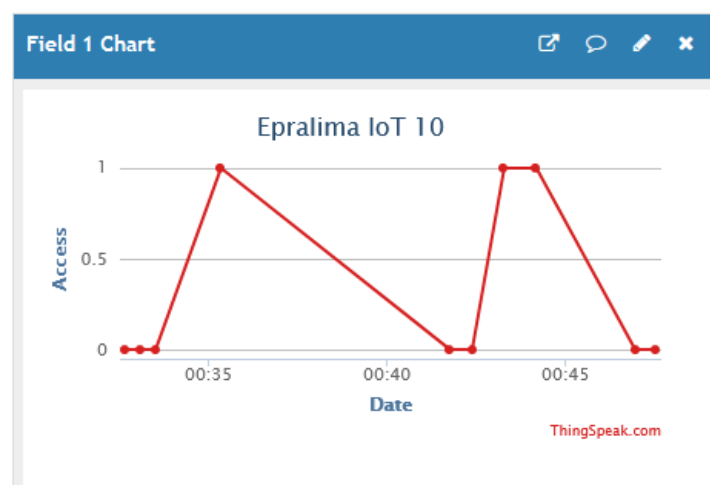


Image 11 – Results IoT ThingSpeak



The data acquired by the ThingSpeak IoT platform can also be exported to CSV files and consequently imported into datasheets as shown in Table 2

created_at	entry_id	field1	field2
24/04/2023 00:32	1	0	
24/04/2023 00:33	2	0	
24/04/2023 00:33	3	0	
24/04/2023 00:35	4	1	
24/04/2023 00:41	5	0	
24/04/2023 00:42	6	0	
24/04/2023 00:43	7	1	
24/04/2023 00:44	8	1	
24/04/2023 00:46	9	0	
24/04/2023 00:47	10	0	

Tabela 2 - DataSheet

In short

This exercise intends to apply a numerical keyboard guided by a microcontroller to block or unblock an electromechanical system. This exercise is aided by a red led that indicates system blocked and a green led that indicates system unlocked.

For possible sending of data, it will be necessary to apply, for example, the ESP8266 ESP-01 module that allows the connection of several devices to the internet (or local network), and consequent sending of data from the sensors applied to the autonomous system.