



## Assembly of circuit with RFID technology for car and sending IoT data

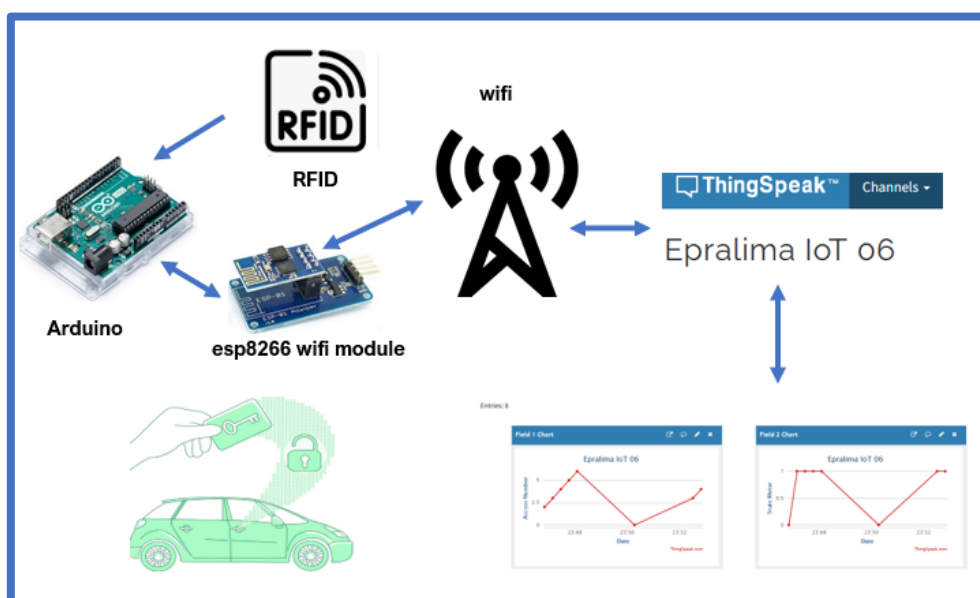
**Difficulty level:** Difficult

### 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 simulates RFID access control. In motorsport, it becomes safer to code contactless accesses. In this exercise it is possible for students to check and control the number of accesses, as well as to know if a car engine is in use at a given moment. 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 RFID technology in a car and communicating with IoT.



**Image 1:** application of RFID technology 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.


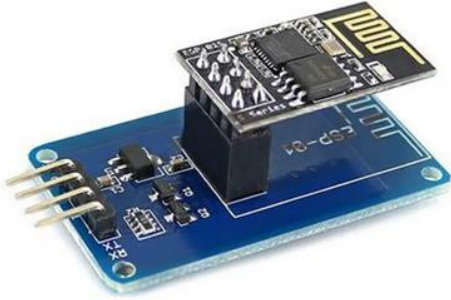
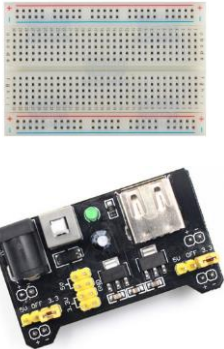





In this exercise we intend to learn how to draw diagrams (circuits), connect all the components correctly, develop software based on C language (Arduino), connect to the wifi network, communicate with an IoT server, ThingSpeak and read server-generated graphics.

Quantity	Component
1	Arduino Uno R3
1	ESP01-8266
1	Power Supply (braedBoard)
1	BreadBoard
2	Push button
2	Led Green and Red
1	LCD display 2 x 16 (I2C)
2	Resistor 1KOhm
2	Resistor 330Ohm
1	DC motor
1	Module bridge L298N

Table 1 - Components List



## Materials table

 <p>Arduino</p>	 <p>ESP01 - 8266</p>
 <p>Bread Board + Power Supply</p>	 <p>L298N Bridge Motor Module</p>
 <p>DC Motor</p>	 <p>LCD display 2 x 16 (I2C)</p>
 <p>Jumper wire</p>	 <p>330Ω      1KΩ      Leds</p>

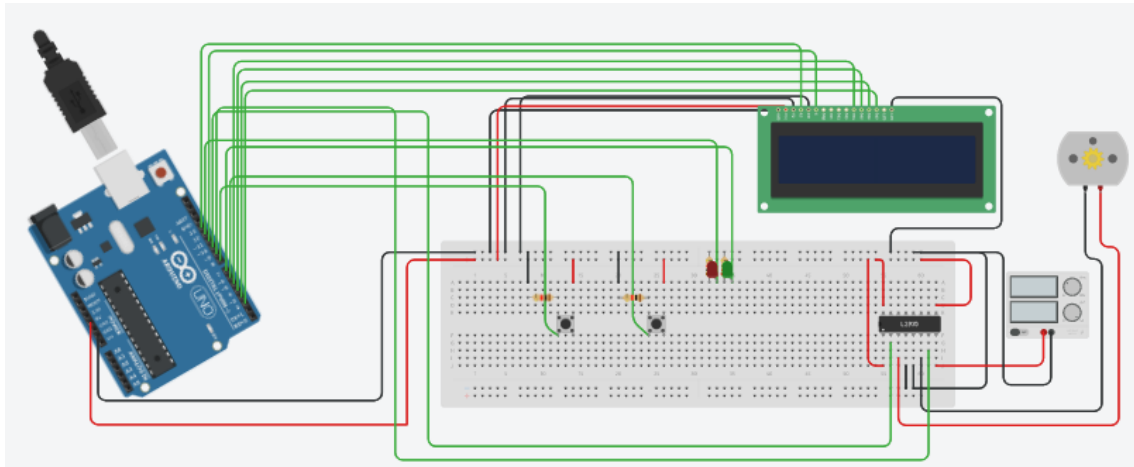


Image 2 – Diagram circuit

## 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.

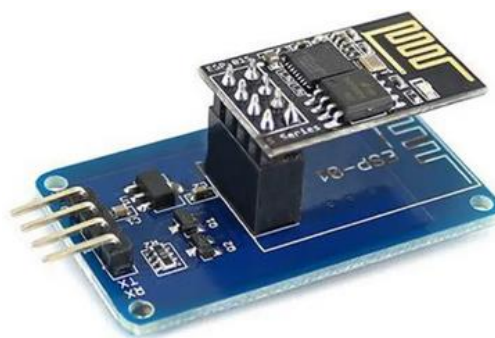


Image 3 - ESP01 – 8266

LM35 Linear Temperature Sensor is based on semiconductor LM35 temperature sensor. It can be used to detect ambient air temperature.



### Implementation in practice

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

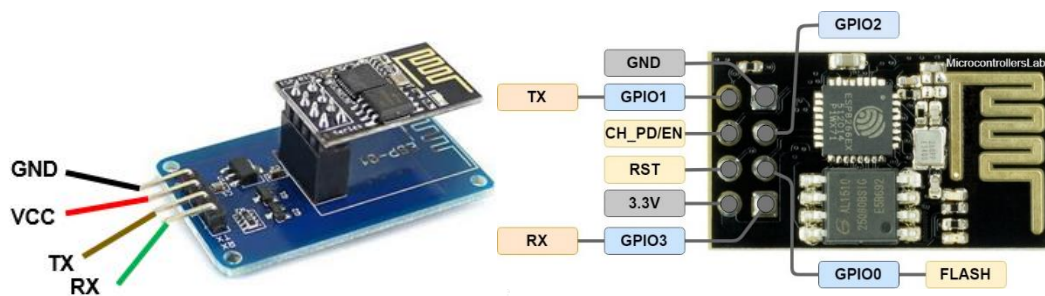


Image 4 ESP-01 Connections

3. Real assembled circuit image 6

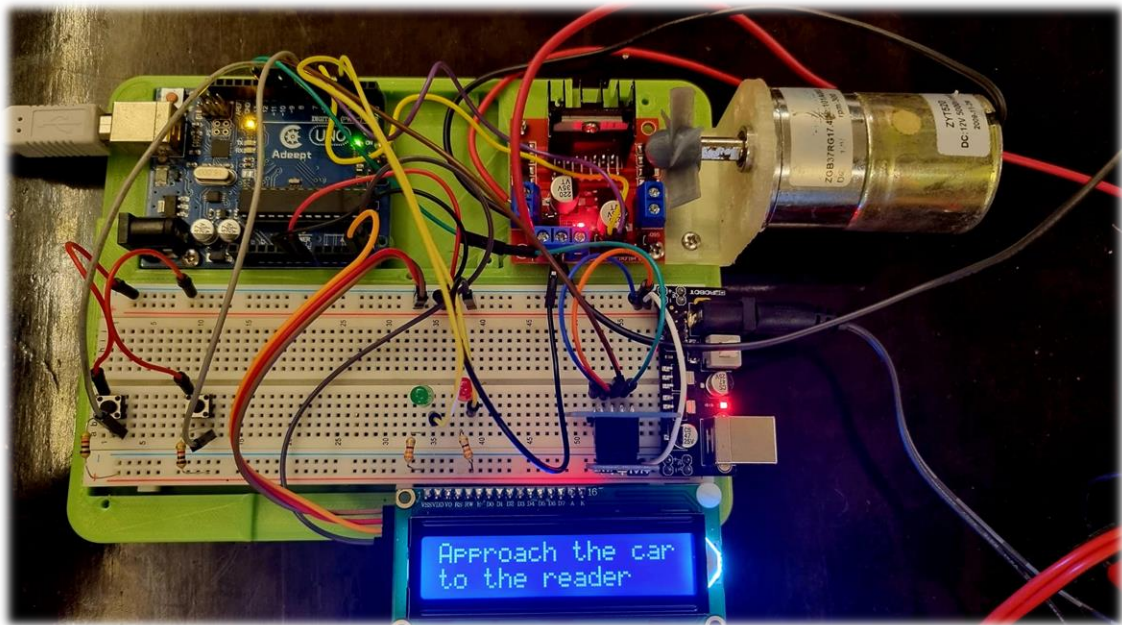


Image 5 Real circuit in breadboard



#### 4. Create a ThingSpeak account image 7

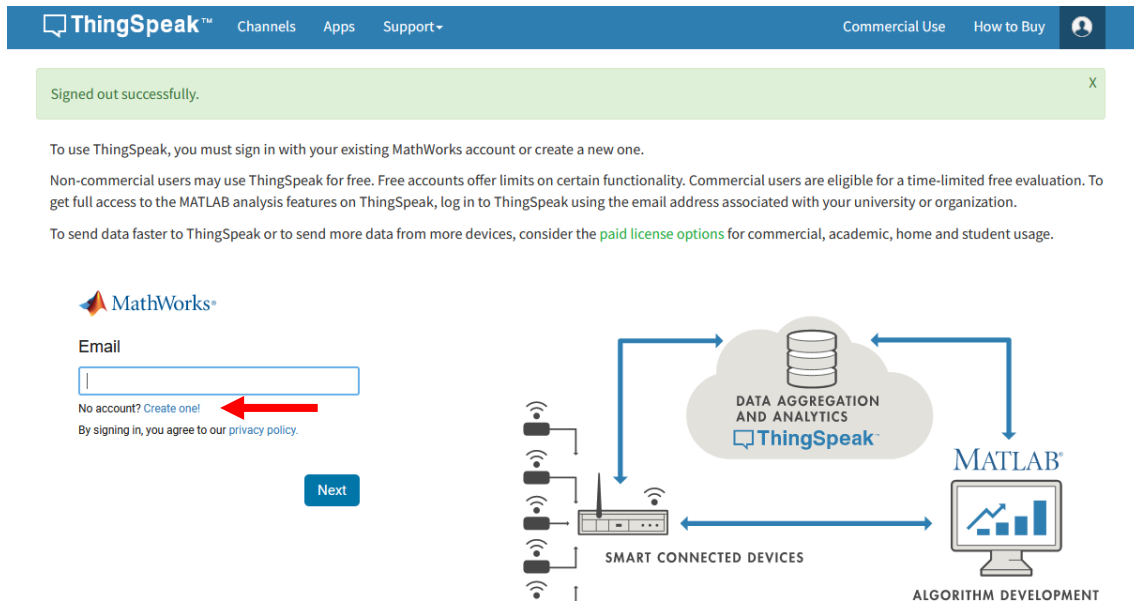


Image 6 - Thing Speak

#### 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.





## Epralima IoT 01

Channel ID: 2064208  
Author: mwa0000029483576  
Access: Private

This Channel presents the simulation of car night lighting based on the level and intensity of natural light

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

### Channel Settings

Percentage complete 50%

Channel ID 2064208

→ Name Epralima IoT 01

→ Description This Channel presents the simulation of car night lighting based on the level and intensity of natural light

→ Field 1 Light level ☒

→ Field 2 State lights ON OFF ☒

### Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

### Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.

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.

**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<Wire.h>
3 #include<LiquidCrystal_I2C.h>
4 #define serialcomSpeed 115200
5 #define DEBUG true
6 SoftwareSerial ESP_Serial(10, 11); //PINOS QUE EMULAM A SERIAL, ONDE
7                                     // RX_AUX  --liga--> TX do ESP01
8                                     // TX_AUX  --liga--> RX do ESP01
9 #define sentidolMotor 5
10 #define sentido2Motor 6
11 #define failedButton 7
12 #define sucessButton 8
13 #define ledGreen 2
14 #define ledRed 3
15 LiquidCrystal_I2C lcd(0x27, 16, 2);
16 void stopMotor(void);
17 void runMotorFoward(void);
18 void runMotorBackward(void);
19 void initialMessage(void);
20 void AccessAllowed(void);
21 void AccessDenied(void);
22 int valor_btn1, _
23 int access_count = 0;
24 int motor_ON_OFF = 0;
25 long writingTimer = 17;
26 long startTime = 0;
27 long waitTime = 0;|
28 boolean error;
29 String APIKey = "4962WQBV1S5V3T1I";
```

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

```
33 void setup() {
34   Serial.begin(serialcomSpeed);
35   ESP_Serial.begin(serialcomSpeed);
36   startTime = millis();
37   InitWifiModuleESP();
38   lcd.init();
39   pinMode(failedButton, INPUT);
40   pinMode(sucessButton, INPUT);
41   pinMode(ledGreen, OUTPUT);
42   pinMode(ledRed, OUTPUT);
43   pinMode(sentidolMotor, OUTPUT);
44   pinMode(sentido2Motor, OUTPUT);
45   lcd.setBacklight(HIGH);
46   initialMessage();|
47 }
```



## 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**

```
102 void EscreverParaThingSpeak(void) {
103
104     startThingSpeakCmd();
105     String getStr = "";
106     getStr = "GET /update?api_key=";
107     getStr += APIKey;
108     getStr += "&field1=";
109     getStr += String(access_count);
110     getStr += "&field2=";
111     getStr += String(motor_ON_OFF);
112     getStr += "\r\n\r\n";
113     GetThingSpeakcmd(getStr);
114 }
```

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 **AccessAllowed()** and **AccessDenied()** These procedures provide instructions to the various components (Display, Leds and DC Motor) depending on whether the access is valid or not.

```
142 void AccessAllowed(void) {
143     digitalWrite(ledGreen,HIGH);
144     digitalWrite(ledRed,LOW);
145     runMotorFoward();
146     lcd.clear();
147     lcd.setCursor(0,0);
148     lcd.print("Access allowed");
149     lcd.setCursor(0,1);
150     lcd.print("Motor ON");
151     lcd.clear();
152     lcd.print("Access allowed");
153     access_count++;
154 }
155
156 void AccessDenied(void) {
157     lcd.clear();
158     digitalWrite(ledRed,HIGH);
159     digitalWrite(ledGreen,LOW);
160     lcd.print("Invalid Card");
161     delay(1000);
162     initialMessage();
163     digitalWrite(ledRed,LOW);
164     stopMotor();
165 }
```



Various procedures for controlling the motor and writing to the LCD.

```
116 void initialMessage(void)
117 {
118   lcd.clear();
119   lcd.print("Approach the card");
120   lcd.setCursor(0,1);
121   lcd.print("to the reader");
122 }
123
124 void stopMotor(void){
125   digitalWrite(sentidolMotor, LOW);
126   digitalWrite(sentido2Motor, LOW);
127   motor_ON_OFF = 0;
128 }
129
130 void runMotorFoward(void){
131   digitalWrite(sentidolMotor, HIGH);
132   digitalWrite(sentido2Motor, LOW);
133   motor_ON_OFF = 1;
134 }
135
136 void runMotorBackward(void){
137   digitalWrite(sentidolMotor, LOW);
138   digitalWrite(sentido2Motor, HIGH);
139   motor_ON_OFF = 1;
140 }
```

## Results

It is possible to control the number of accesses as well as to understand if an engine is in use at any given time.

Entries: 8

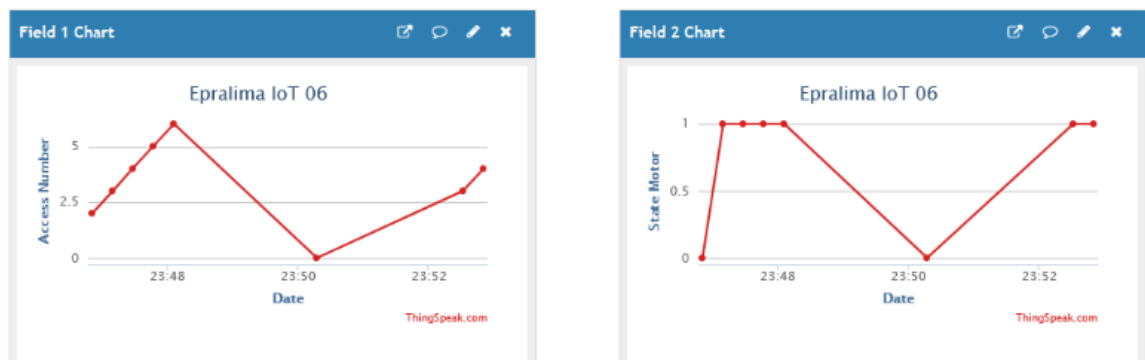


Image 11 – Results IoT ThingSpeak



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	field3
19/04/2023 23:46	1	2	0	1
19/04/2023 23:47	2	3	1	1
19/04/2023 23:47	3	4	1	1
19/04/2023 23:47	4	5	1	1
19/04/2023 23:48	5	6	1	1
19/04/2023 23:50	6	0	0	1
19/04/2023 23:52	7	3	1	1
19/04/2023 23:52	8	4	1	1

Tabela 2 - DataSheet

## In short

This exercise simulates RFID access control. In motorsport, it becomes safer to encode accesses through contact less.