

Final Report PoC



ÉCOLE
D'INGÉNIEURS
PARIS-LA DÉFENSE



Embedded Systems: Architecture and Programming
Proof of Concept – Design and implement an embedded system

Summary

1 – Introduction	3
2 – Devices & Circuits.....	4
3 – LoraWan	7
4 – The Thing Network.....	11
5 – Cayenne.....	15
6 – Database	17
7 – Deep Sleep Mode	20
8 – Development.....	23
9 – Conclusion	Erreur ! Signet non défini.



1– Introduction

The situation of bees is currently urgent. Indeed, this species is on the verge of extinction while its role in pollination of plants is essential. In addition, bees are very sensitive to their environment, which is increasingly polluted. Beekeepers have difficulty to find healthy environments to set up hives. Then it is about making the right observations and making the right decisions. This is also not easy when you are a beginner or amateur for example. So, one of the main constraints of this project is the fact that bees are sensitive to wave frequencies. That's why we need to develop a system that communicates with low frequencies. In addition, new, unknown constraints have been added, posing major problems such as climate change.

Here is all the constraints for the system we want to develop:



- Climate change poses very big problems for the health of bees and it's a new problem
- Continuous high frequencies are bad for the health of bees



- We have to develop a low-cost product to help beekeepers
- Our system will not be able to be connected to the mains, so we must have a very good autonomy



- In addition to having to use low frequencies, our system will have to have a fairly good range for the campaign.

That's why we developed a connected hive with LoRaWan. This hive will allow you to know in real time the temperature, humidity with the sensors connected in the hives. The collected data will be sent to a server so that it can be read remotely by the beekeeper on a mobile application. The first feature of our system will therefore be a real-time monitoring of the hive. After, these data will be retrieved from a database and analysed and processed through Deep Learning in order to combat the problem of climate change.

First, we will look at the different materials and components used for our system.

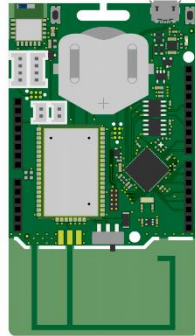
2 – Devices & Circuits

Material list:

Computer



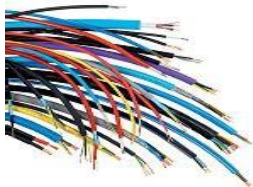
SODAQ Explorer



10k ohm resistor



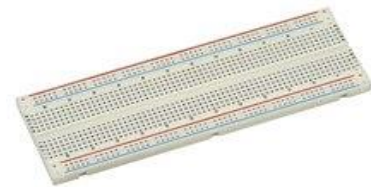
DHT22



Feeds



3V flat round battery



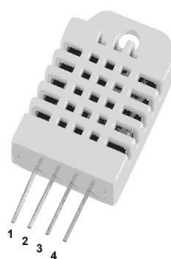
Breadboard

Here is the list of materials used to develop our system.

For the microcontroller, we first wanted to use the Arduino MKR 1300 LAN because it's a microcontroller compatible with LoraWan and the IDE of Arduino. But we saw that the school already have another microcontroller for LoraWan wich is the SODAQ Explorer. In addition to being compatible with lorawan, this microcontroller was also compatible on Arduino IDE so we decided to use this microcontroller.

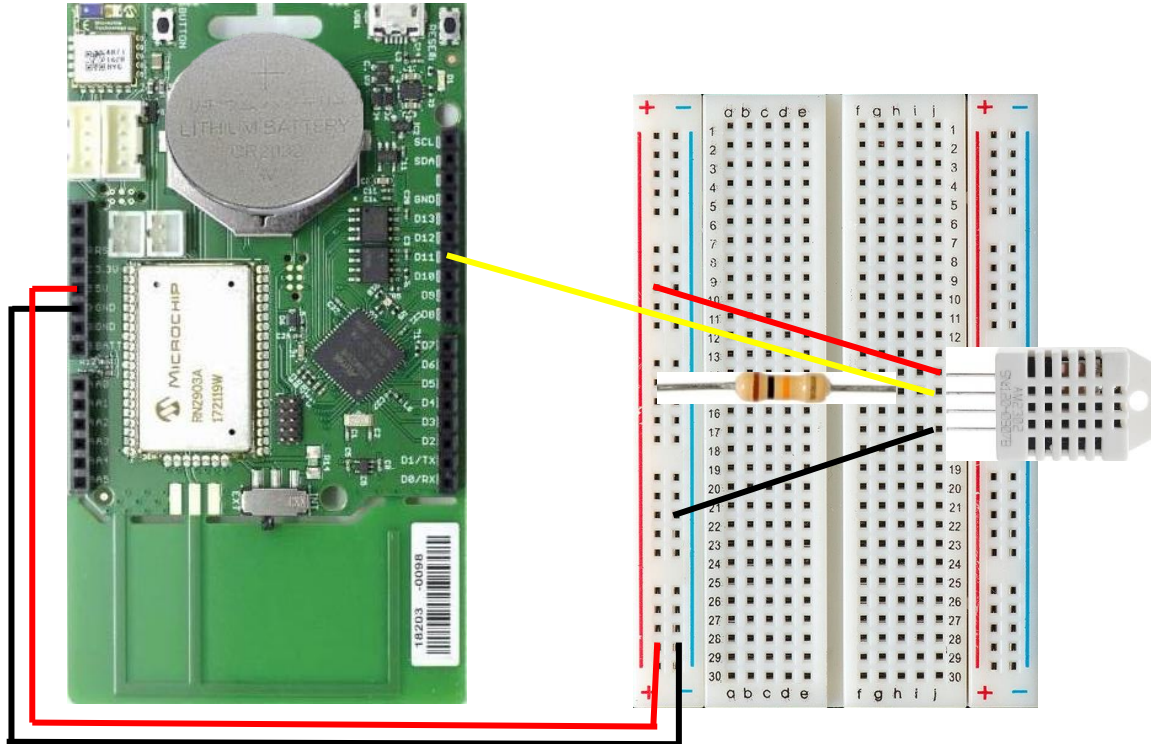
For the sensor, we wanted to catch the temperature and the humidity. The DHT22 can have both of them with a high precision. This sensor catches the temperature a little bit slowly but it's not a constrain for our case. Finally, this sensor is not expensive, so we chose to use it.

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND



For the DHT22, we need to use a resistor between 4,7 and 10 k ohm connected to the second pin. This resistor is protecting the data pin of the sensor. We have chosen this resistance following various tests we have carried out on the sensor. We concluded that the resistance of 10k ohm allowed us to have very precise results while protecting our equipment to the maximum.

Circuits



So, this is the circuit to connect the DHT22 to the SODAQ Explorer. The DHT22 is connected to the + and - of the sodaq on pin 1 and 4. Pin 2 is used to collect data and we can see that we add a resistor connected to the power to protect the sensor. The third pin is not connected to anything, it is used to test and calibrate the sensor.

To finish we put the flat round battery inside the microcontroller to use it when it's not connected to the computer and test the deep sleep mode after.

DHT22

To read the DHT22 with the SODAQ Explorer we develop a code on Arduino IDE. We need 2 libraries for this code. The first one is DHT.h for the DHT22, it allows us to start and stop this device and to get the value of the temperature and the humidity. The second library is Arduino.h, this one is for the microcontroller to translate the Arduino command for the SODAQ Explorer.

For the code, we need to declare the DHT we use with two arguments: the pin connected to the data and the type of the DHT (22 or 11 for exemple). Here we used the pin 11 and we have the DHT22.

In the setup we have to start the sensor with the command `dht.begin()`. and to declare the baud rate with `serial.begin(baud rate we want)`.

In the loop, we are going to read all the values:

- the temperature in C° with `dht.readTemperature()`;
- the relative humidity in percent with `dht.readHumidity()`;
- the temperature felt in C° with `dht.computeHeatIndex()`;

It's also possible to read the temperature in Fahrenheit with the same function. We just need to declare the argument `isFahrenheit` to `true` in the `dht.readTemperature()` function.

In the loop we also add the function `if` to check if the data have been correctly read. We used `isnan(value)` function to check if the value is a number. If it's not the case, we send a message and restart the code.

To finish, we display all the values on the computer with the function `Serial.print(value)`.

You can see all our code just below:

```
#include <DHT.h>
#include <DHT_U.h>
#include <Arduino.h>
|
#define DHTPIN 11 // broche ou l'on a branche le capteur
// de-commenter le capteur utilise
#define DHTTYPE DHT22 // DHT 22 (AM2302)
DHT dht(DHTPIN, DHTTYPE); //déclaration du capteur

void setup()
{
    Serial.begin(9600);
    Serial.println("DHT22 test!");
    dht.begin();
}
void loop()
{
    delay(2000);

    // La lecture du capteur prend 250ms
    // Les valeurs lues peuvent etre vieilles de jusqu'a 2 secondes (le capteur est lent)
    float h = dht.readHumidity(); //on lit l'hygrometrie
    float t = dht.readTemperature(); //on lit la temperature en celsius (par default)
    // pour lire en fahrenheit, il faut le parametre (isFahrenheit = true) :
    float f = dht.readTemperature(true);

    //On verifie si la lecture a echoue, si oui on quitte la boucle pour recommencer.
    if (isnan(h) || isnan(t) || isnan(f))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }
}
```

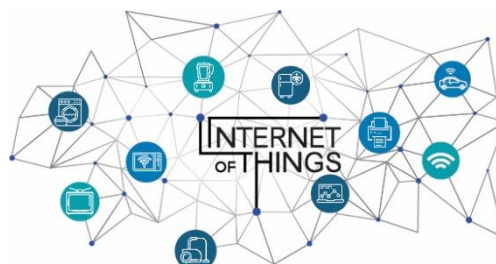
```
'  
// Calcul de l'indice de temperature en Farenheit  
float hif = dht.computeHeatIndex(f, h);  
// Calcul de l'indice de temperature en Celsius  
float hic = dht.computeHeatIndex(t, h, false);  
  
//Affichages :  
Serial.print("Humidite: ");  
Serial.print(h);  
Serial.print(" %\t");  
Serial.print("Temperature: ");  
Serial.print(t);  
Serial.print(" *C ");  
Serial.print(f);  
Serial.print(" *F\t");  
Serial.print("Indice de temperature: ");  
Serial.print(hic);  
Serial.print(" *C ");  
Serial.print(hif);  
Serial.println(" *F");  
}
```

3– LoraWan

Lora Wan is a communication protocol like the WIFI or the Bluetooth. It was developed by Cycleo of Grenoble, France and acquired by Semtech the founding member of the LoRa Alliance. This communication protocol is working with Low Radio Rate. This is why we chose to use this technology to preserve bee's health.



There is another communication protocol called SigFox very similar to LoraWan. We chose LoraWan because the flow and quantity of data sent is much greater than SigFox. Moreover, this technology allows to secure the connections between the connected objects and the server unlike SigFox which has less possibility to secure its connections. LoraWan is thus a technology in full expansion it is for all these reasons that we preferred LoraWan to SigFox.



To finish, LoraWan is also a long-range radio communication and it's a low-cost technology because it's open source and there is a lot of free Cloud to get back all the data. We can see that LoraWan is corresponding to all our constrains. Now we're going to see how it works:



To start, the microcontroller read the temperature from the DHT22. After, the Lora Module on the SODAQ Explorer send via Low Radio Rate a signal to a LoraWan Gateway. This signal can use 2 modes of activation. For our case we used the OTAA activation that work with an address called DevEUI and a password called the AppKey. These 2 informations will be send with the Gateway with a Secure IP connection to a LoraWan Cloud. The Cloud will check is the DevEUI and the AppKey are correct and saved on their site. For our case, we chose to use The Thing Network as a Cloud because it's a free one. Now we can get the value on internet via a computer by going on the site or sending a mail for example. Now we're going to see the code to send all the data via LoraWan.



Libraries and definition of variables:

```
#include <Arduino.h>
#include "TheThingsNetwork.h"
#include "CayenneLPP.h"
#include <Adafruit_Sensor.h>
// Board Definitions
#define bleSerial Serial1
#define loraSerial Serial2
#define debugSerial SerialUSB
#define SERIAL_TIMEOUT 10000

#define ADC_AREF 3.3f
#define BATVOLT_R1 4.7f
#define BATVOLT_R2 10.0f
#define BATVOLT_PIN BAT_VOLT

#include <DHT.h>
#include <DHT_U.h>

#define debugSerial SerialUSB
#define DHTPIN 11
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
// LoRa Definitions and constants
// For OTAA. Set your AppEUI and AppKey. DevEUI will be serialized by using HwEUI (in the RN module)
const char *appEui = "70B3D57ED00272B6";
const char *appKey = "7FEAF9F85F2F0D45C45D37276707F373";

const bool CNF = true;
const bool UNCNF = false;
const byte MyPort = 3;
byte Payload[51];
byte CNT = 0; // Counter for the main loop, to track packets while prototyping
#define freqPlan TTN_FP_EU868 // Replace with TTN_FP_EU868 or TTN_FP_US915
#define FSB 0 // FSB 0 = enable all channels, 1-8 for private networks
#define SF 7 // Initial SF
```

Here we can see all the libraries and the constant and name we defined for all the code. Like before, we used the Arduino.h library to translate the command from Arduino. The Adafruit_Sensor.h is a library that allow us to send data from the sensor. We also include TheThingNetwork.h because we're going to use this cloud for our case. The library CayenneLPP is here to provide a convenient and easy way to send data over LPWAN networks such as LoRaWAN. The Cayenne LPP is compliant with the payload size restriction, which can be lowered down to 11 bytes, and allows the device to send multiple sensor data at one time. We can also easily do a mobile application after with Cayenne therefore we chose to use it.

Otherwise, we define different names for the Serial connection because it will be better after to manage them easily. Just below these, we define some parameters to calculate the percentage of battery on the microcontroller to send it after with LoRaWAN. The DHT library, the pin and the type are defined like we explained it before. Just below, we can see the declaration of the appEui and the appKey to secure the connection with the Cloud. To finish, we declare the frequency and the parameters of the LoRa connection.

The setup:

```

loraSerial.begin(57600);
debugSerial.begin(57600);
// Wait a maximum of 10s for Serial Monitor
while (!debugSerial && millis() < SERIAL_TIMEOUT);

// Set callback for incoming messages
ExpLoRer.onMessage(message);

//Set up LoRa communications
debugSerial.println("-- STATUS");
ExpLoRer.showStatus();

delay(1000);

debugSerial.println("-- JOIN");

// For OTAA. Use the 'join' function
if (ExpLoRer.join(appEui, appKey, 6, 5000)) // 6 Re-tries, 3000ms delay in between
    LED(GREEN); // Switch to GREEN if OTAA join is successful
else
    LED(RED); // Switch to RED if OTAA join fails

// For ABP. Use the 'personalize' function instead of 'join' above
//ExpLoRer.personalize(devAddr, nwksKey, appSKey);

//Hard-coded GPS position, just to create a map display on Cayenne
CayenneRecord.reset();
CayenneRecord.addGPS(1, 51.4141391, -0.9412872, 10); // Thames Valley Science Park, Shinfield, UK
// Copy out the formatted record
byte PayloadSize = CayenneRecord.copy(Payload);
dht.begin();
ExpLoRer.sendBytes(Payload, PayloadSize, MyPort, UNCNF);
CayenneRecord.reset();
} // End of the setup() function

```

In the setup we initialise the same baud rate for the Lora module and the USB module and we start them. After we wait that the Serial Monitor is working with the function while(). After, we set call back for messages that are coming on the microcontroller. We also set up the Lora communication by using the function showStatus that return this:

```

-- STATUS
EUI: 0004A30B002049BA
Battery: 3283
AppEUI: 0000000000000000
DevEUI: 0004A30B002049BA
Data Rate: 5
RX Delay*1: 1000
RX Delay 2: 2000
-- JOIN

```

We need the DevEUI to set up the connection with The Thing Network. So, with the showStatus function we can get this number.

To join the connection, we use the function Explorer.join(appEui, appKey, n° of retries, delay between) and we turn the led into green if the connection successful and red if not.

Now with the function CayenneReccord we can reset the connection with Cayenne on The Thing Network. So we reset before to send the first message. In our case we send the GPS address because we know it's possible to get it easily with Lora by

triangulation with the Gateways. But we didn't have the time to code the function now, so we send a hard-coded position to use the library CayenneLPP with the GPS.

We also wanted to send this position just first time the hive is install so we send this message in the setup. This is why we use the function CayenneReccord.copy to copy the last message added to the CayenneReccord function. The message of the position will be send the position with the function Explorer.sendbytes().

The Loop:

```
void loop()
{
  CNT++;
  debugSerial.println(CNT, DEC);

  float MyTemp = getTemperature(); // Onboard temperature sensor
  CayenneRecord.addTemperature(3, MyTemp);
  float RelativeHumidity = getHumidity(); // Grove sound sensor on pin A8
  CayenneRecord.addRelativeHumidity(4, RelativeHumidity);
  CayenneRecord.addAnalogInput(2, getBatteryVoltage());
  SerialUSB.println(getBatteryVoltage());
  // When all measurements are done and the complete Cayenne record created, send it off via LoRa
  LED(BLUE); // LED on while transmitting. Green for energy-efficient LoRa
  byte PayloadSize = CayenneRecord.copy(Payload);
  byte response = ExpLoRer.sendBytes(Payload, PayloadSize, MyPort, UNCNF);

  LED(response + 1); // Change LED colour depending on module response to uplink success
  delay(100);

  CayenneRecord.reset(); // Clear the record buffer for the next loop
  LED(OFF);

  // Delay to limit the number of transmissions
  for (byte i=0; i<8; i++) // 8+2second delay. Additional ~2sec delay during Tx
  {
    // LED(i % 8); // Light show, to attract attention at tradeshow!
    delay(500);
    LED(OFF);
    delay(500);
  }
} // End of loop() function
```

The first thing we do in the loop is to start the counter. After we use the function CayenneReccord to send the temperature, the humidity and the level of battery. The first argument is these functions is the channel of the value. To finish the loop we delay the code to manage the number of transmissions.

4 – The Thing Network

TheThingNetwork is a website, it provides a set of open tools and a global, open network to build IoT application at low cost, featuring maximum security and ready to scale. The Things Network is a proud contributor member of the LoRa Alliance and that's why we used their services to send data by LoraWan network. LoraWan is used thanks to its particularity to emit electromagnetic waves at lower frequencies than known technologies

**95694****MEMBERS****9944****GATEWAYS****147****COUNTRIES**

Above, this is a map showing the extent of the network in the world (147 countries), they have 9944 gateways, this guarantees us to have a means of communicating our data

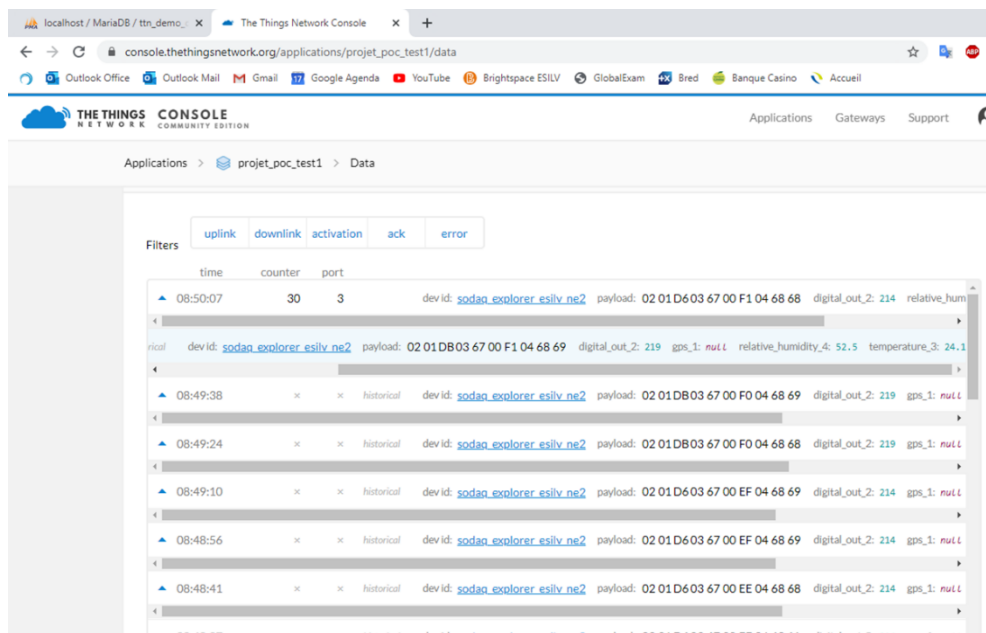
We must configure a connection between the device and a gateway. We follow a tutorial for program the SODAQ Explorer board and connecting it to the The Things Network (TTN).

First, we install sodaq explorer librarie on Arduino IDE via Board Manager window, then we instal The Things Network Arduino Libraries and Cayenne Low Power Payload (LPP) Libraries which is not currently necessary but helpful in another use. These libraries bring new way to programming and sending informations to the nearby gateway.

Secondly, we setup the Arduino Serial Port in order to interact, log & debug applications on connected devices.

Finally, we create a The Things Network Account to register our device and to have a login and a password. After the device is registered, we have created an application (display to see datas) and select the appropriate Handler Europe: ttn-handler-eu we can establish a connection.

Below, an example of display of our data on the site. We can't sell iot devices with thethingnetwork database.

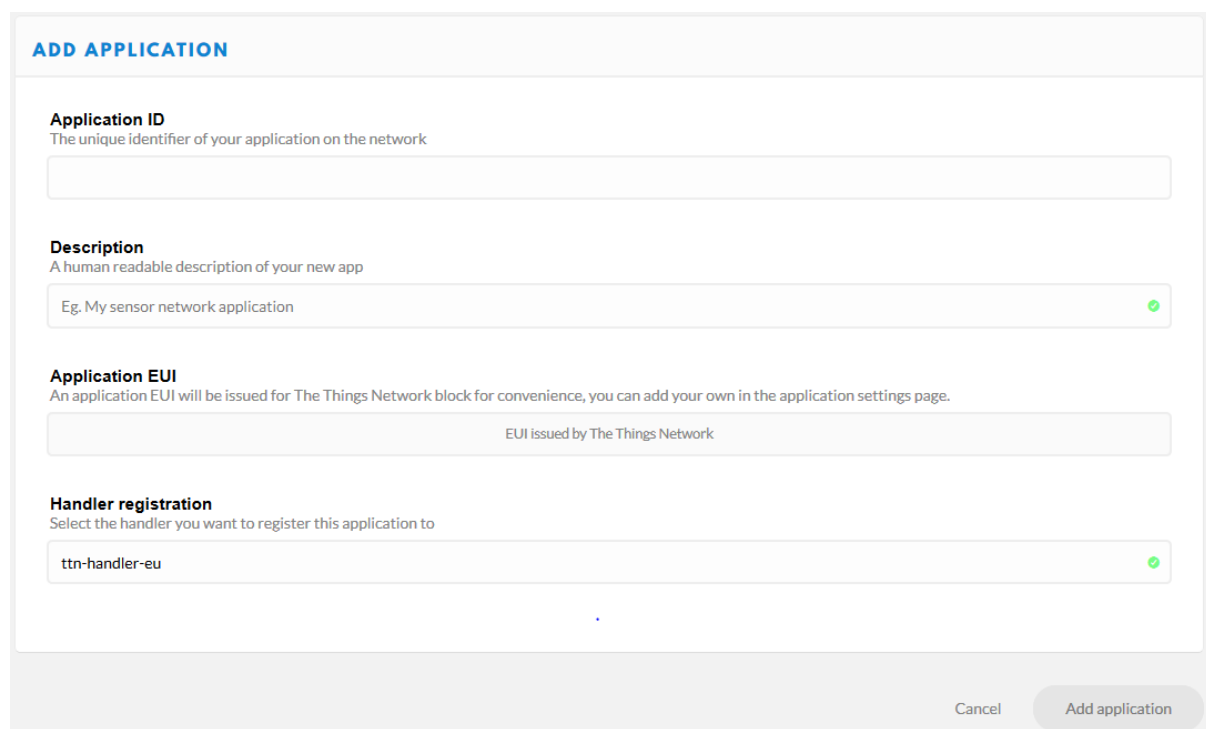


The screenshot shows the 'Data' tab in the The Things Network Console for the application 'projet_poc_test1'. It displays a list of data points with filters for 'uplink', 'downlink', 'activation', 'ack', and 'error'. The data points are sorted by time, counter, and port. Each entry includes a timestamp, counter, port, device ID, payload, and various sensor readings like digital_out_2, gps_1, relative_humidity_4, and temperature_3.

time	counter	port	dev id	payload	digital_out_2	relative_hum
08:50:07	30	3	sodan_explorer_esilv_ne2	02 01D603 67 00 F1 04 68 68	214	
08:49:38	x	x	sodan_explorer_esilv_ne2	02 01DB03 67 00 F0 04 68 69	219	gps_1: null relative_humidity_4: 52.5 temperature_3: 24.1
08:49:24	x	x	sodan_explorer_esilv_ne2	02 01DB03 67 00 F0 04 68 68	219	gps_1: null
08:49:10	x	x	sodan_explorer_esilv_ne2	02 01D603 67 00 EF 04 68 69	214	gps_1: null
08:48:56	x	x	sodan_explorer_esilv_ne2	02 01D603 67 00 EF 04 68 69	214	gps_1: null
08:48:41	x	x	sodan_explorer_esilv_ne2	02 01D603 67 00 EE 04 68 68	214	gps_1: null
08:48:27	x	x	sodan_explorer_esilv_ne2	02 01D603 67 00 EF 04 68 68	214	gps_1: null

The solution is to create a mysql database to create mobile applications and real-time monitoring. We can also use TTN solution, myDevices Cayenne allows us to quickly design, prototype, and visualize IoT solutions. It can be a tool to visualize real-time and historical data, sent over The Things Network, but there is a risk of user rights issues, some services impose non-commercial licenses, and this could be problematic in the future.

So just below we can see how to add an application on The Thing Network:



The screenshot shows the 'ADD APPLICATION' form in the The Things Network Console. It includes fields for Application ID, Description, Application EUI, and Handler registration. The form is partially filled out with example data.

ADD APPLICATION

Application ID
The unique identifier of your application on the network

Description
A human readable description of your new app

Eg. My sensor network application

Application EUI
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.

EUI issued by The Things Network

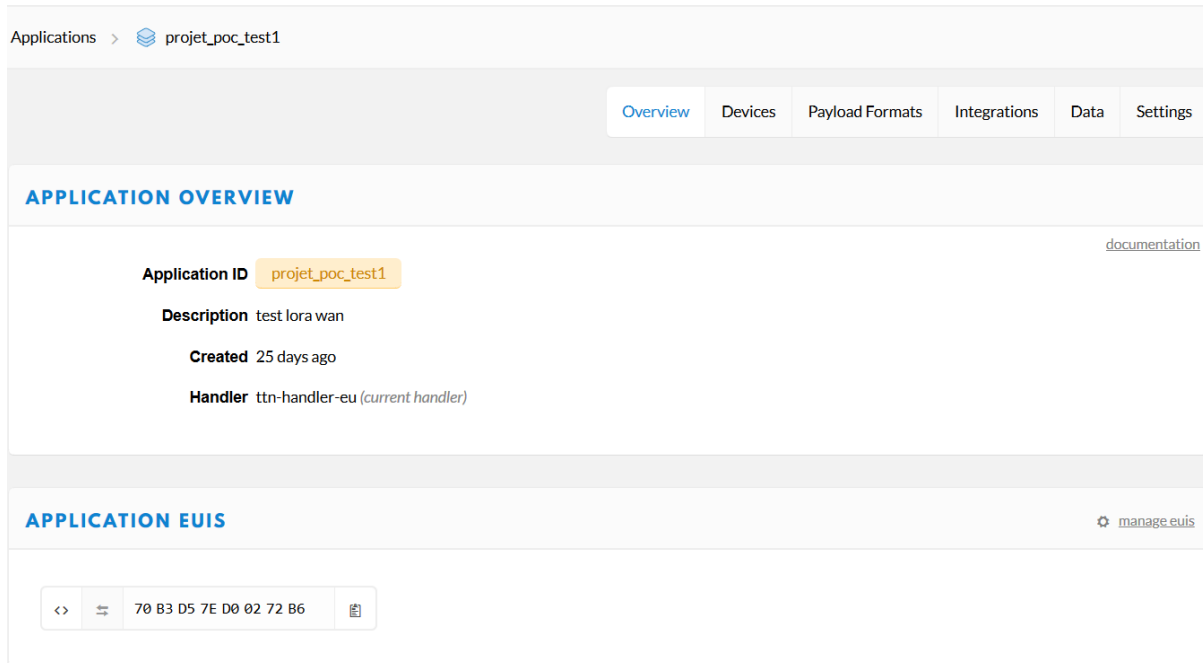
Handler registration
Select the handler you want to register this application to

ttn-handler-eu

Cancel Add application

After you registered on TTN, you have to create an application to connect the device.

You have to choose a name that you will put in the “Application ID” and you click to add application after. After you have to click on “Payload Formats” like you can see below and chose Cayenne LPP.

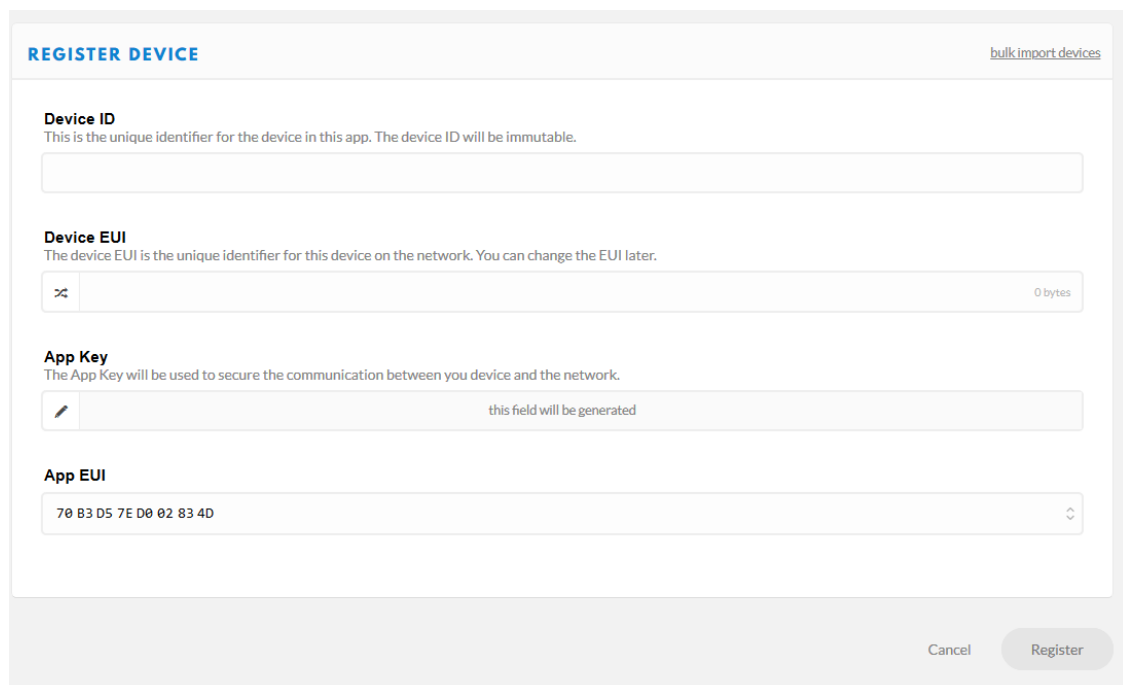


The screenshot shows the 'APPLICATION OVERVIEW' page for an application named 'projet_poc_test1'. The page has a breadcrumb 'Applications > projet_poc_test1' and a navigation bar with tabs: Overview (selected), Devices, Payload Formats, Integrations, Data, and Settings. The overview section displays the following information:

- Application ID:** projet_poc_test1
- Description:** test lora wan
- Created:** 25 days ago
- Handler:** ttn-handler-eu (current handler)

Below the overview, there is a section titled 'APPLICATION EUIs' with a 'manage euis' link. It shows a list of EUIs with a table containing one entry: '70 B3 D5 7E D0 02 72 B6'.

Now we are in the application and we need to add our device. For this we have to click on “Register a device” and after you arrive in this window that you can see below. You have to choose a name and put it in “Device ID”. After, you need to fill the DevEUI display on the monitor when you use the function Explorer.showstatus() in our code. You will see “-STATUS-” on the monitor, it will be just below. To finish you have to click on “Register”.



The screenshot shows the 'REGISTER DEVICE' form. It has a breadcrumb 'bulk import devices' and a title 'REGISTER DEVICE'. The form contains the following fields:

- Device ID:** A text input field with a description: 'This is the unique identifier for the device in this app. The device ID will be immutable.'
- Device EUI:** A text input field with a description: 'The device EUI is the unique identifier for this device on the network. You can change the EUI later.'
- App Key:** A text input field with a description: 'The App Key will be used to secure the communication between you device and the network.'
- App EUI:** A text input field with a description: 'The App EUI will be used to secure the communication between you device and the network.'

At the bottom of the form, there are two buttons: 'Cancel' and 'Register'.

To finalize your connection, you have to go in the Device Overview and copy the "Exemple Code" with the DevEUI and the AppKey in your code and to run it.

Applications > projet_poc_test1 > Devices > sodaq_explorer_esilv_ne2

DEVICE OVERVIEW

Application ID **projet_poc_test1**

Device ID **sodaq_explorer_esilv_ne2**

Activation Method **OTAA**

Device EUI **<> 00 04 A3 0B 00 20 21 BD**

Application EUI **<> 70 B3 D5 7E D0 02 72 B6**

App Key **<>**

Device Address **<> 26 01 25 7F**

Network Session Key **<>**

App Session Key **<>**

Status ● 18 days ago

Frames up 42 [reset frame counters](#)

Frames down 2

You will see this in the part "Data" of your application. We can see the counter of the number of transmissions. We can also see from which device the transmission comes from and the data are sorted by channel number.

localhost / MariaDB / ttn_demo... x The Things Network Console x +

console.thethingsnetwork.org/applications/projet_poc_test1/data

THE THINGS NETWORK CONSOLE COMMUNITY EDITION Applications Gateways Support




Applications > projet_poc_test1 > Data

Filters **uplink** downlink activation ack error

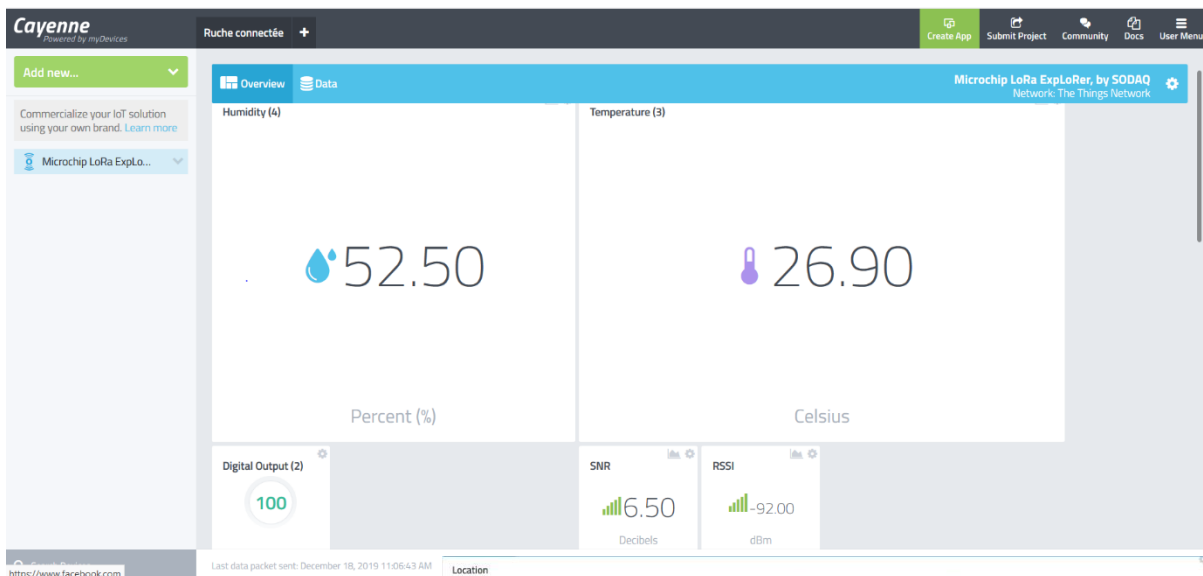
time	counter	port	dev id	payload	digital_out_2	relative_hum
08:50:07	30	3	dev id: sodaq_explorer_esilv_ne2	payload: 02 01 D6 03 67 00 F1 04 68 68	digital_out_2: 214	relative_hum
08:49:38	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 DB 03 67 00 F0 04 68 69	digital_out_2: 219	gps_1: null relative_humidity_4: 52.5 temperature_3: 24.1
08:49:24	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 DB 03 67 00 F0 04 68 68	digital_out_2: 219	gps_1: null
08:49:10	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 D6 03 67 00 EF 04 68 69	digital_out_2: 214	gps_1: null
08:48:56	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 D6 03 67 00 EF 04 68 69	digital_out_2: 214	gps_1: null
08:48:41	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 D6 03 67 00 EE 04 68 68	digital_out_2: 214	gps_1: null
08:48:27	x	x	historical dev id: sodaq_explorer_esilv_ne2	payload: 02 01 D6 03 67 00 EF 04 68 68	digital_out_2: 214	gps_1: null

5 – Cayenne

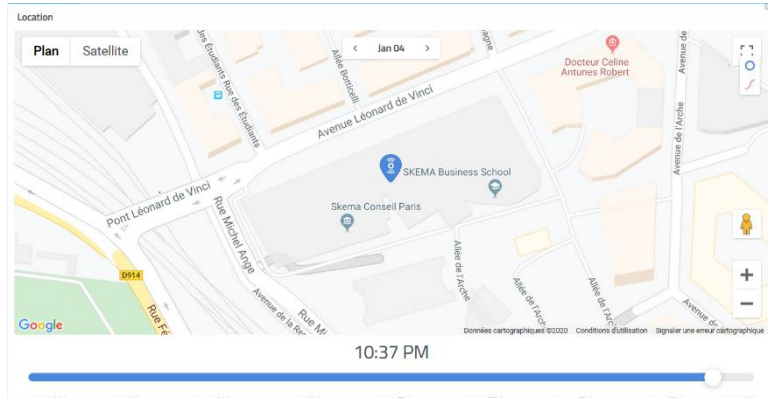
After creating your account on www.cayenne.mydevices.com you just have to configure your connexion with your device. For this you have to define the type of microcontroller and the type of connexion you use. You also have to copy the DevEUI used by your microcontroller.

Settings	Device Name	Microchip LoRa ExpLoRer, by SODAQ	
General	Device Icon	 LoRa	
	DevEUI	0004A30B002021BD	
	Activation Mode	Already registered	
	Remove Device	Remove Device	This action cannot be undone

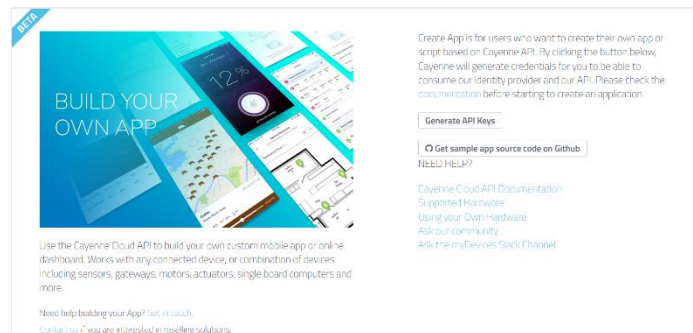
Then, you will have all your data display easily on your computer:



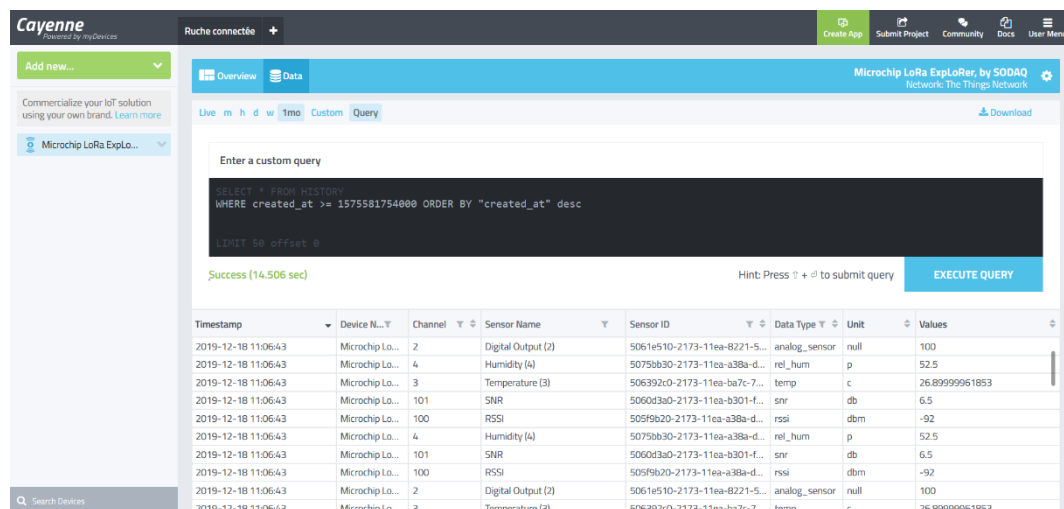
Cayenne can also localise the device and display its localisation on a map like this:



Moreover, there is a possibility to create an application easily with cayenne because there is already some API Key and code source on Github for this.



To finish, we can also have the data hard-coded. This can be really useful for the rest because we would like to catch these data and create a database on MySQL.



Timestamp	Device Name	Channel	Sensor Name	Sensor ID	Data Type	Unit	Values
2019-12-18 11:06:43	Microchip Lo...	2	Digital Output (2)	5061e510-2173-11ea-b221-5...	analog_sensor	null	100
2019-12-18 11:06:43	Microchip Lo...	4	Humidity (4)	5075b630-2173-11ea-a38a-d...	rel_hum	p	52.5
2019-12-18 11:06:43	Microchip Lo...	3	Temperature (3)	506392c0-2173-11ea-ba7c-7...	temp	c	26.899999961853
2019-12-18 11:06:43	Microchip Lo...	101	SNR	5060d3a0-2173-11ea-b301-f...	snr	db	6.5
2019-12-18 11:06:43	Microchip Lo...	100	RSSI	505f9b20-2173-11ea-a38a-d...	rssi	dbm	-92
2019-12-18 11:06:43	Microchip Lo...	4	Humidity (4)	5075b630-2173-11ea-a38a-d...	rel_hum	p	52.5
2019-12-18 11:06:43	Microchip Lo...	101	SNR	5060d3a0-2173-11ea-b301-f...	snr	db	6.5
2019-12-18 11:06:43	Microchip Lo...	100	RSSI	505f9b20-2173-11ea-a38a-d...	rssi	dbm	-92
2019-12-18 11:06:43	Microchip Lo...	2	Digital Output (2)	5061e510-2173-11ea-b221-5...	analog_sensor	null	100
2019-12-18 11:06:43	Microchip Lo...	3	Temperature (3)	506392c0-2173-11ea-ba7c-7...	temp	c	26.899999961853

6 – Database

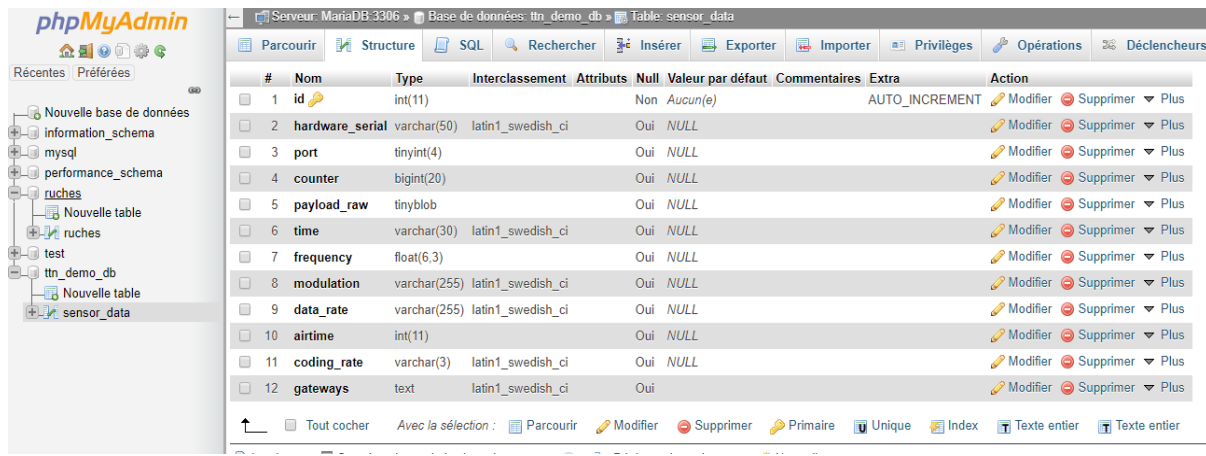
The temp sensor and humidity sensor data acquisition are available on TheThingNetwork website because we use the LoraWan Network services. It's important for our project to create our own database in order to be independent from the external website services.

We cannot create devices or provide programming services only from external website services. That's why we decide to create our database and allow us to manipulate our data to create statistics, graphics, SMS alert...

We found a Javascript code on internet to retrieve data from thething network and store them into our database.

First step: We create a database using Phpmyadmin, We create many table and especially many primary key for each table in order to sort our data more easily and then allow us to answer some question like "What time was the highest temp or

humidity value? " or otherwise in case of theft " Who is the owner of the hive whose GPS position value or movement sensor value has changed "



#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
2	hardware_serial	varchar(50)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
3	port	tinyint(4)			Oui	NULL			Modifier Supprimer Plus
4	counter	bigint(20)			Oui	NULL			Modifier Supprimer Plus
5	payload_raw	tinyblob			Oui	NULL			Modifier Supprimer Plus
6	time	varchar(30)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
7	frequency	float(6,3)			Oui	NULL			Modifier Supprimer Plus
8	modulation	varchar(255)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
9	data_rate	varchar(255)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
10	airtime	int(11)			Oui	NULL			Modifier Supprimer Plus
11	coding_rate	varchar(3)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
12	gateways	text	latin1_swedish_ci		Oui				Modifier Supprimer Plus

2nd step:

We copy the JS code and complete the program especially according to our specificity.
Source: https://www.mobilefish.com/download/lora/lora_part27.pdf

We create option for database connection and ttn connection with config.js

```

JS config.js > ...
16 const databaseOptions = {
17   host: 'localhost',
18   user: 'root',
19   password: ''
20 };
21
22 const TTNOptions = {
23   appID: 'projet_poc_test1',
24   accessKey: 'ttn-account-v2.hy88P0srrvfox7bI0My7A3PN9VDx3HoUA18_o55FQCoA'
25 };
26 module.exports = {databaseOptions: databaseOptions, TTNOptions: TTNOptions};
  
```

2 const : In databaseOptions we put login and password to connect to the database. In TTNOptions we put login and password to connect to our TTN account in order to retrieve data. These config.js attributes will then be used by all the other classes that use the database or ttn

We create database for sensor data with create_db.js

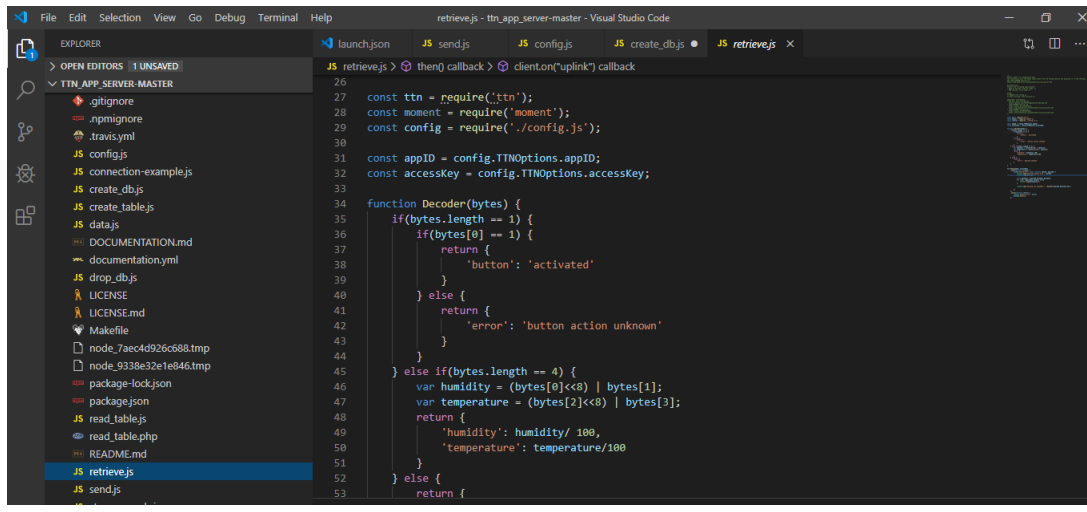
```

JS create_db.js > ...
19
20 const mysql = require('mysql');
21 const config = require('./config.js');
22
23 const con = mysql.createConnection(config.databaseOptions);
24
25 con.connect(function(err) {
26   if (err) throw err;
27
28   con.query("CREATE DATABASE Ruches", function (err, result) {
29     if (err) throw err;
30     console.log("Database ttn_demo_db created");
31     process.exit(1);
32   });
33 });
34
  
```

The JS class create_db use SQL Syntax to create the database Ruches in the PhpmyAdmin database. It use information from the previous class "config.js" to establish database connection and from mysql.js, a class that we installed using

npm install mysql it's a library available to write code more easily. In java code we would use "import mysql..."

We retrieve sensor data on the hinknetwork using retrieve.js



```
26
27 const ttn = require('ttn');
28 const moment = require('moment');
29 const config = require('./config.js');
30
31 const appID = config.TTNOptions.appID;
32 const accessKey = config.TTNOptions.accessKey;
33
34 function Decoder(bytes) {
35   if(bytes.length == 1) {
36     if(bytes[0] == 1) {
37       return {
38         'button': 'activated'
39       }
40     } else {
41       return {
42         'error': 'button action unknown'
43       }
44     }
45   } else if(bytes.length == 4) {
46     var humidity = (bytes[0]<<8) | bytes[1];
47     var temperature = (bytes[2]<<8) | bytes[3];
48     return {
49       'humidity': humidity/ 100,
50       'temperature': temperature/100
51     }
52   } else {
53     return {
```

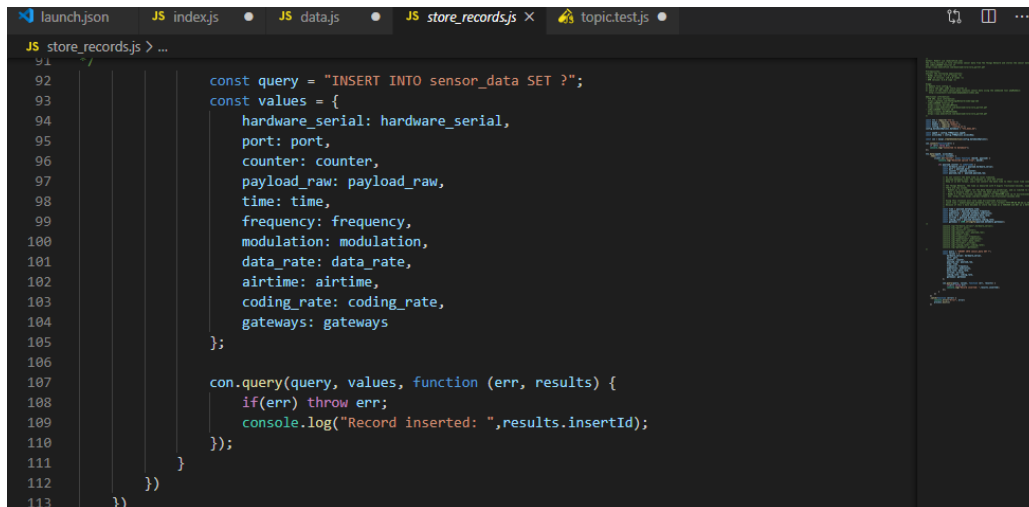
We import ttn module in order to have necessary tools to retrieve data from ttn network, moment module and config options in order to connect to ttn account. The decoder function allows to retrieve the temp and humidity sensor when the bytes length is 4. These numbers represent the coding of the information we sent using the arduino application



```
58
59 ttn.data(appID, accessKey)
60 .then(function (client) {
61   client.on("uplink", async function (devID, payload) {
62     console.log("Received uplink from ", devID);
63     console.log(payload);
64
65     const gateways = payload.metadata.gateways;
66     for (i=0; i<gateways.length; i++){
67       console.log(gateways[i]);
68     }
69
70     console.log("payload_raw decoded = ", Decoder(payload.payload_raw));
71
72   })
73 })
74 .catch(function (error) {
75   console.error("Error", error)
76   process.exit(1);
77 })
78
```

*This is the part of the code which is problematic: "ttn.data(appID, accessKey)" We can't have access to ttn because ttn module is unfinished. Yet we tried to install it using "npm install ttn" "npm install ttn -v***". The problem is there is no data.js code taking 2 parameters "AppID and accesskey" in ttn module*

The store_record.js and send.js code allows to insert the retrieved data to Phymyadmin database



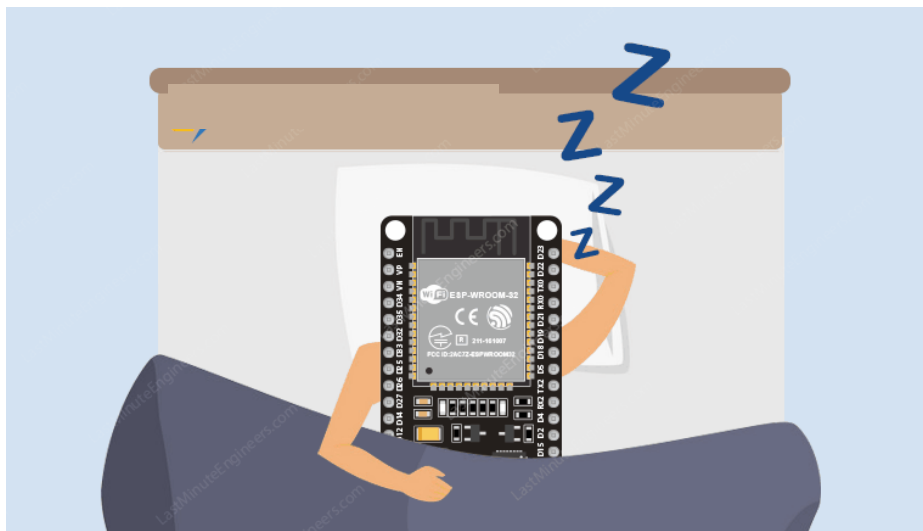
```
91 /
92
93 const query = "INSERT INTO sensor_data SET ?";
94 const values = {
95   hardware_serial: hardware_serial,
96   port: port,
97   counter: counter,
98   payload_raw: payload_raw,
99   time: time,
100   frequency: frequency,
101   modulation: modulation,
102   data_rate: data_rate,
103   airtime: airtime,
104   coding_rate: coding_rate,
105   gateways: gateways
106 };
107
108 con.query(query, values, function (err, results) {
109   if(err) throw err;
110   console.log("Record inserted: ",results.insertId);
111 });
112
113 })
114 })
```

The send.js file creates a downlink and sends binary data from your computer to your LoRa end node.

The store_records.js file retrieves sensor data from The Things Network and stores the sensor data in the MySQL database.

All has been put in place in order to exploit our data correctly, but given that was our first use, it took some time to become familiar with this programming language and after understanding the program we noticed that it missing some library that we tried to install as for the raspberry by using without succes the function "npm install....."

7 – Deep Sleep Mode



Because our project is to create a smart hive, we need to manage the battery of our module to make it efficient as possible. For this, we used the deep sleep mode available on Lora.

To do this, we went on the site of SEMTECH where they recently open the LoraWan Academy for free:

<https://lora-developers.semtech.com/resources/lorawan-academy/>

So, we tried different code to switch into sleep different component of our microcontroller. we managed to turn off the Bluetooth component, the LoraWan antenna for example. But we encountered a problem, the complete code proposed by LoraWan Academy could not make the SODAQ Explorer reactivate like we can see here when we ran the code and waited few minutes:

COM14

```
setup lora
setup lora otaa
setup lora OTAA
Network_connection_failed!
setup lora OTAA
setup lora fin
init_rtc_timer
event set
init timer
wdt reset
usb device
Sleep setup
DFU
transmit
transmit end
reset SPI pin
reset pin
reset pin fin
reset pin
reset pin fin
reset pin
reset pin fin
reset pin
reset pin fin
reset SPI pin fin
DFU end
lora sleep
lora sleep
BT powerdown
BT powerdown fin
Sleep setup fin
sleep
no interrupt
  interrupt
Sleeping
```

☒ Défilement automatique ☐ Afficher l'horodatage

So, we studied the code by displaying a message for each step and we concluded that the problem came from the timer. So, we tested the library RTCTimer with simple codes. We saw that the Timer can't synchronise, so we tried another library called Time:

```
void setup() {
  Serial.begin(9600);
  while (!Serial) ;
  delay(5000);
  SerialUSB.println("while");// the function to get the time from the RTC
  // wait until Arduino Serial Monitor opens

  DateTime timeSet = setSyncProvider(RTC.get);
  SerialUSB.println("while");// the function to get the time from the RTC
  if(timeStatus() != timeSet)
    Serial.println("Unable to sync with the RTC");
  else
    Serial.println("RTC has set the system time");
}

void loop()
{
  SerialUSB.println("E000");// the function to get the time from the RTC

  if (timeStatus() == timeSet) {
    SerialUSB.println("1 er pas");// the function to get the time from the RTC

    digitalClockDisplay();
    SerialUSB.println("deuxieme");// the function to get the time from the RTC

    delay(500);
  } else {
    Serial.println("The time has not been set. Please run the Time");
    Serial.println("TimeRTCSet example, or DS1307RTC SetTime example.");
    Serial.println();
    delay(4000);
  }
  delay(1000);
}
```

Like before, we display every action done by the microcontroller on the console. With this technique, we understood that the function setSynchProvider(RTC.get) was not working. We saw that it was possible that the SODAQ Explorer wasn't compatible with these libraries.

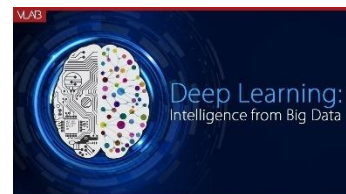
So, we looked after another library for a timer especially for the SODAQ Explorer. We find the library Sodaq_RN2483 so we tried an example purposed by the library. We think that the test was done successfully. We think this because we don't have access to LoraWan Network from our home and we can't go to the university during holidays. But we can see that the microcontroller sleep and wake up here:

COM14

```
Connection to the network was successful.
Sleeping for 5 seconds before starting sending out test packets.
5
4
3
2
1
Connection timed-out. Check your serial connection to the device! Sleeping for 20sec.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
The device is busy. Sleeping for 10 extra seconds.
```

8 – Development

For the development, in case of success for our project, we expect to create mobile application for beekeepers. This application will display the temperature and the humidity in the hive.



We also want to develop a code to analyse all the data we took with Deep Learning to predict the temperature and the reaction of the bees. So, the result and action to do will be written on the mobile application.

Otherwise, we know there is another problem and we have looked into it. Indeed, there is a big problem with the theft of hives. So, we want to combat this problem with a connected alarm. The hive will be located and equipped with an accelerometer that will wake up the microcontroller when the hive undergoes axis changes. We will also make sure to wake up the microcontroller if the initial position is changed. When a theft attempt is detected, a powerful sound



transmitter will be activated, and the connected hive will send its position in real time. In addition, an SMS or call will be sent to the beekeeper. We have already found the necessary hardware to do this but have not started coding it.

9 – Conclusion

Finally, we developed what we wanted for the connected hive. The temperature and humidity are captured and then sent on low frequency waves via LoraWan and we use a cloud facilitating the creation of applications. In addition, we have almost finished developing the code for the deep sleep mode and we could soon put it in place.

It was hard to do all of this because the community and knowledge about LoraWan is evolving. So, we couldn't get much help by going to see different teachers in our school. And even SEMTECH's site on the LoraWan Academy was created recently and is therefore not yet complete. But there is a lot of work on LoraWan and others on Sdaq as well as forums that allowed us to progress well.

We have made a good start on the development of the MySQL database that will allow us to start processing the data quickly afterwards. And as we explained before, we already know that we will develop the anti-theft system afterwards once all this is done.

All the Code on GitHub:

<https://github.com/WassimSERRADJ/Connected-Hive-LoraWan>