**Chapitre X : Station Météo Connectée**

1. **Introduction**

Dans le cadre du projet collectif sur la **serre connectée**, notre groupe a pris en charge le développement d’une **station météo intelligente**. Cette dernière permet de **surveiller les conditions climatiques extérieures** en temps réel, et de transmettre les données au système global via une architecture LoRaWAN. Ces informations sont essentielles pour ajuster automatiquement des paramètres de la serre tels que l’irrigation, la ventilation ou le chauffage.

1. **Objectifs**

* Collecter des **données météorologiques complètes** :
* Température de l'air
* Humidité relative
* Pression atmosphérique
* Vitesse du vent
* Direction du vent
* Hauteur de pluie (rainfall)
* Transmettre ces données via **LoRa** en utilisant **OTAA**.
* Publier les données dans un **MQTT broker** via **ChirpStack**.
* Partager ces informations avec les autres sous-systèmes de la serre

1. **Analyse Fonctionnelle**

## Besoin Fondamental

Le besoin fondamental du projet est de fournir en temps réel les conditions climatiques extérieures au système global de la serre connectée, afin de permettre l'ajustement automatique des paramètres de la serre tels que l'irrigation, la ventilation ou le chauffage

1. Bête d'analyse fonctionnelle (Bête à cornes)

Utilisateurs

Conditions Climatiques Extérieures

Ajustement Automatique des Paramètres de la Serre

Sur quoi agit-il?

À qui rend-il service?

Dans quel but?

Pour identifier le besoin fondamental du projet de station météo connectée, nous avons utilisé le diagramme appelé « bête à cornes ».

Ce schéma nous a permis de répondre à trois questions essentielles :

* **À qui rend service le système ?**
  + → Aux utilisateurs de la serre connectée
  + → Au système de gestion de la serre
* **Sur quoi agit-il ?**
  + → Sur les conditions climatiques extérieures (température, humidité, pression, vent, pluie)
* **Dans quel but ?**
  + → Pour surveiller et transmettre les données météorologiques en temps réel afin d'ajuster automatiquement les paramètres de la serre.

Ce diagramme nous aide à bien cerner le rôle de la station météo intelligente et à cadrer ses fonctions principales.

## Diagramme de pieuvre:

S’adapter aux conditions climatiques

Utilisateur

Résister à l’environnement extérieur

FC1

Collecter les données météorologiques

FC2

FP1

FC3

**Station Météo**

FC6

Respecter les normes de fréquence

Assurer une communication fiable

FC5

FC4

Fiabiliser les mesures

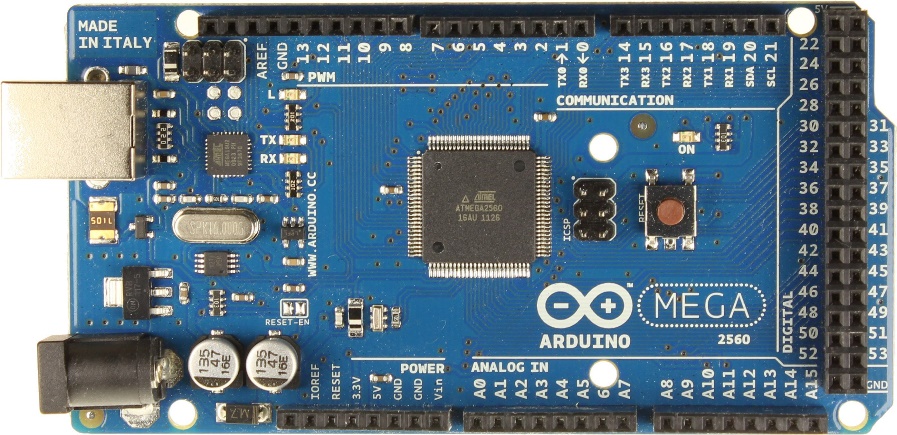
Fonctionner sur énergie limitée

|  |
| --- |
|  |
| | **Réf.** | **Type de Fonction** | **Fonction** | **Description** | | --- | --- | --- | --- | | FP1 | Fonction Principale | Collecter les données météorologiques | Mesurer en temps réel température, humidité, pression, vent, pluie, etc. | |
| |  |  |  |  | | --- | --- | --- | --- | | FC1 | Fonction Contrainte | S’adapter aux conditions climatiques | Fonctionner malgré le vent, la pluie, les fortes chaleurs ou le froid | |
| |  |  |  |  | | --- | --- | --- | --- | | FC2 | Fonction Contrainte | Résister à l’environnement extérieur | Être protégé contre la poussière, l’humidité, les insectes, les chocs, etc. | |
| |  |  |  |  | | --- | --- | --- | --- | | FC3 | Fonction Contrainte | Assurer une communication fiable | Utiliser des protocoles adaptés (LoRaWAN, MQTT), même à longue distance | |
| |  |  |  |  | | --- | --- | --- | --- | | FC4 | Fonction Contrainte | Fonctionner sur énergie limitée | Être opérationnel avec une alimentation sur batterie ou énergie solaire | |
| |  |  |  |  | | --- | --- | --- | --- | | FC5 | Fonction Contrainte | Fiabiliser les mesures | Fournir des données précises et valides pour automatiser correctement la serre | |
| |  |  |  |  | | --- | --- | --- | --- | | FC6 | Fonction Contrainte | Respecter les normes de fréquence | Émettre sur 868 MHz, conforme aux réglementations européennes LoRaWAN | |

1. **Matériel utilisé**

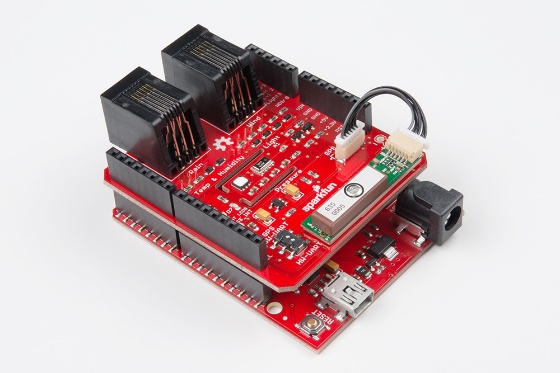
| **Composant** | **Référence** | **Fonction** |
| --- | --- | --- |
| **Carte microcontrôleur** | Arduino Mega | Contrôle des capteurs |
| **Carte météo Shield** | SparkFun Weather Shield | Mesure T°, H%, pression, vent, pluie |
| **Anémomètre + Girouette + Pluviomètre** | Intégrés via Weather Meters | Mesure vent et pluie |
| **Module LoRa** | RAk811 | Transmission LoRaWAN |
| **Passerelle LoRa** | RAK7243 | Relais vers ChirpStack |
| **Plateforme serveur** | ChirpStack | Gestion réseau et application |
| **Broker MQTT** | Mosquitto | Diffusion des données |

1. **Carte microcontrôleur – Arduino Mega 2560**



L’Arduino Mega est le cœur de notre station météo. Il joue le rôle d’unité centrale de traitement. Grâce à ses nombreux ports d’entrée/sortie et à sa mémoire étendue, il est parfaitement adapté à la lecture simultanée de plusieurs capteurs. Il récupère les données climatiques mesurées par la SparkFun Weather Shield, les traite, les organise et les prépare pour l’envoi via le module LoRa.

1. **Carte météo – SparkFun Weather Shield**



Cette carte spécialisée est conçue pour les stations météo Arduino. Elle intègre plusieurs capteurs environnementaux :

* **HTU21D** : mesure la température de l’air et l’humidité relative.
* **MPL3115A2** : capteur de pression barométrique.
* **ALS-PT19** : capteur de luminosité (optionnel selon version).

La carte possède également des connecteurs destinés à accueillir des capteurs mécaniques externes :

* **Anémomètre** : mesure la **vitesse du vent**.
* **Girouette** : mesure la **direction du vent**.
* **Pluviomètre** : mesure la **quantité de pluie tombée** grâce à un système de bascule à augets.

Elle permet donc de récupérer l’ensemble des paramètres météorologiques utiles à la gestion d’une serre.



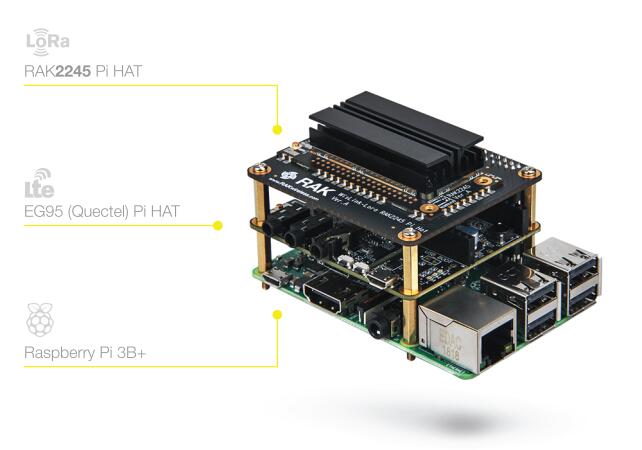
1. **Module de communication – LoRa (RA-02 / SX1276)**

Le module LoRa est utilisé pour transmettre les données météorologiques collectées vers la passerelle distante. Il fonctionne sur une fréquence de 868 MHz, adaptée à l’Europe, et permet une communication à très longue portée, tout en consommant peu d’énergie.  
Il communique avec l’Arduino via SPI et envoie les messages sous forme de paquets radio. Dans notre projet, le module est configuré pour une activation sécurisée par OTAA (Over-The-Air Activation).



1. **Passerelle LoRaWAN – RAK7243**

La passerelle RAK7243 joue un rôle fondamental dans le système. Elle reçoit les paquets radio émis par le module LoRa, les encapsule dans des paquets UDP, puis les transmet à un serveur LoRaWAN (ChirpStack).  
Elle est composée d’un concentrateur LoRa (SX1301) couplé à un Raspberry Pi 3, ce qui lui permet à la fois de gérer la partie radio et de se connecter au réseau Ethernet ou Wi-Fi pour transférer les données.

1. **Serveur réseau – ChirpStack**

ChirpStack est une plateforme serveur LoRaWAN open source. Elle est responsable de :

* La gestion des appareils connectés au réseau LoRa,
* L’authentification des nœuds via leurs identifiants OTAA (DevEUI, AppEUI, AppKey),
* Le décodage des paquets reçus (généralement encodés en base64),
* Et enfin de la transmission des données sous forme de messages MQTT.

Grâce à son interface web, ChirpStack nous permet de suivre en direct les paquets reçus, l’état des capteurs, et de configurer facilement notre station.



1. **Système de messagerie – MQTT (Mosquitto)**

Une fois les données traitées par ChirpStack, elles sont publiées sur un broker MQTT (Mosquitto dans notre cas). MQTT est un protocole de messagerie léger, idéal pour les communications entre machines.  
Les messages publiés sur un topic spécifique sont ensuite récupérables par d’autres modules de la serre (comme l’irrigation ou la ventilation), ou affichés sur des dashboards comme Node-RED ou Grafana.

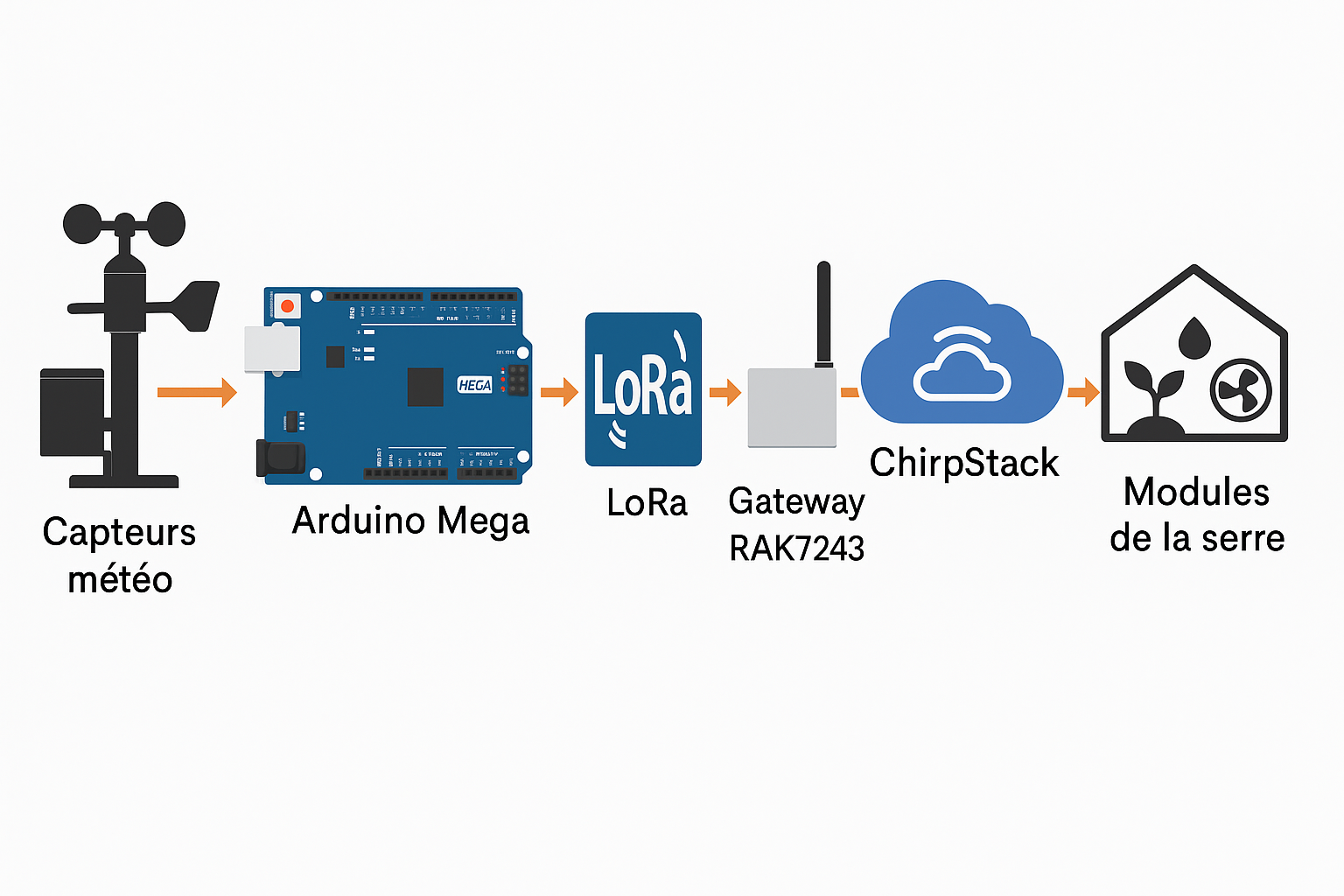


1. **Données mesurées**

La station météo mesure les paramètres suivants :

| **Donnée mesurée** | **Unité** | **Utilité dans la serre** |
| --- | --- | --- |
| Température (°C) | Celsius | Contrôle du chauffage ou des ouvertures |
| Humidité (%) | % | Gestion de l’humidité interne |
| Pression (hPa) | Hectopascals | Anticipation des changements météo |
| Direction du vent | Degrés (°) | Orientation automatique éventuelle |
| Vitesse du vent | m/s ou km/h | Sécurité (fermeture en cas de vent fort) |
| Pluie | mm | Gestion de l’irrigation |

1. **Architecture du système**

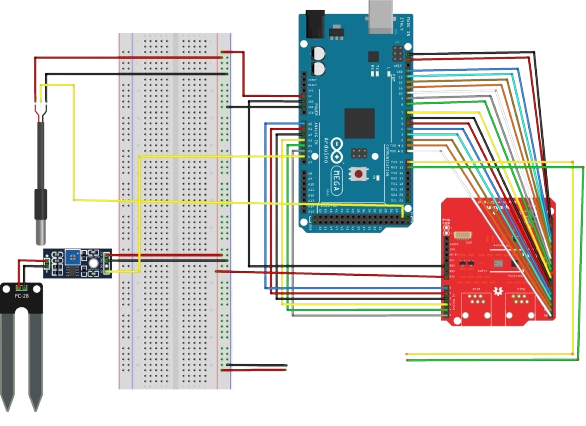
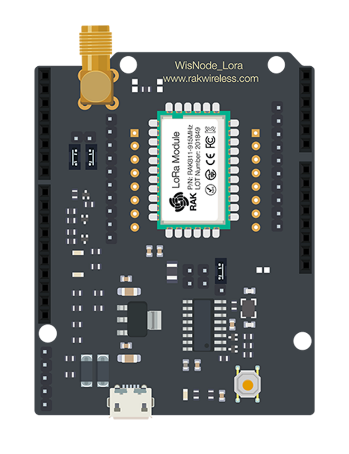


L’architecture de la station météo repose sur une chaîne de transmission structurée et optimisée pour l’environnement d’une serre connectée. Les capteurs météorologiques, reliés à la carte SparkFun Weather Shield, mesurent des paramètres tels que la température, l’humidité, la pression atmosphérique, la vitesse et la direction du vent, ainsi que la pluviométrie. Ces données sont collectées et traitées par une carte Arduino Mega, qui les transmet via un module LoRa fonctionnant à 868 MHz. La passerelle RAK7243 reçoit les paquets LoRa et les transfère au serveur réseau ChirpStack, chargé de l’authentification du dispositif, du décodage des données et de leur publication sur un broker MQTT. Grâce à ce schéma, les informations sont accessibles en temps réel par les autres modules de la serre, permettant une automatisation intelligente en fonction des conditions extérieures.

1. **Méthodologie et Implémentation**

La mise en œuvre de notre station météo connectée repose sur une architecture combinant capteurs environnementaux, traitement embarqué avec Arduino Mega, et transmission LoRaWAN via un module RAK811. Cette partie détaille le processus de configuration, de programmation, et d’intégration avec la passerelle RAK7243 et le serveur ChirpStack.

1. **Configuration de l’environnement Arduino**



Nous avons utilisé **Arduino IDE** pour le développement embarqué sur **Arduino Mega 2560**. Après installation des bibliothèques nécessaires (SparkFunMPL3115A2, SparkFun\_Si7021, RAK811, DallasTemperature, etc.), nous avons développé un programme structuré en trois grandes fonctions :

* **Acquisition des données capteurs** : température, humidité relative, pression atmosphérique, direction et vitesse du vent, pluie, luminosité, niveau de batterie et humidité du sol.
* **Affichage des mesures sur le port série**, avec conversion d’unités et interprétation des directions du vent.
* **Envoi des données formatées** via LoRaWAN, en mode OTAA, avec une trame ASCII codée en hexadécimal.

#include <Wire.h>

#include "SparkFunMPL3115A2.h"

#include "SparkFun\_Si7021\_Breakout\_Library.h"

#include "RAK811.h"

#include <SoftwareSerial.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#define WORK\_MODE LoRaWAN   // LoRaWAN or LoRaP2P

#define JOIN\_MODE OTAA      // OTAA or ABP

#define TX\_INTERVAL 3000  // 3 second between transmissions (adjust as needed)

// Hardware Serial for RAK811 (use SoftwareSerial if needed)

#define RAK811\_RX 15

#define RAK811\_TX 14

#define DebugSerial Serial

SoftwareSerial loraSerial(RAK811\_RX, RAK811\_TX); // RX, TX

RAK811 RAKLoRa(loraSerial, DebugSerial);

MPL3115A2 myPressure; //Create an instance of the pressure sensor

Weather myHumidity;//Create an instance of the humidity sensor

// Configuration capteurs

#define ONE\_WIRE\_BUS 22

#define FC28\_PIN A6

const int SEC\_VALUE = 1023;

const int WET\_VALUE = 220;

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

float temperature = 0.0;

int soilHumidity = 0;

unsigned long lastSendTime = 0;

//Hardware pin definitions

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

// digital I/O pins

const byte WSPEED = 3;

const byte RAIN = 2;

const byte STAT1 = 7;

const byte STAT2 = 8;

// analog I/O pins

const byte REFERENCE\_3V3 = A3;

const byte LIGHT = A1;

const byte BATT = A2;

const byte WDIR = A0;

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Weather sensor;

//Global Variables

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

long lastSecond; //The millis counter to see when a second rolls by

byte seconds; //When it hits 60, increase the current minute

byte seconds\_2m; //Keeps track of the "wind speed/dir avg" over last 2 minutes array of data

byte minutes; //Keeps track of where we are in various arrays of data

byte minutes\_10m; //Keeps track of where we are in wind gust/dir over last 10 minutes array of data

long lastWindCheck = 0;

volatile long lastWindIRQ = 0;

volatile byte windClicks = 0;

//We need to keep track of the following variables:

//Wind speed/dir each update (no storage)

//Wind gust/dir over the day (no storage)

//Wind speed/dir, avg over 2 minutes (store 1 per second)

//Wind gust/dir over last 10 minutes (store 1 per minute)

//Rain over the past hour (store 1 per minute)

//Total rain over date (store one per day)

byte windspdavg[120]; //120 bytes to keep track of 2 minute average

#define WIND\_DIR\_AVG\_SIZE 120

int winddiravg[WIND\_DIR\_AVG\_SIZE]; //120 ints to keep track of 2 minute average

float windgust\_10m[10]; //10 floats to keep track of 10 minute max

int windgustdirection\_10m[10]; //10 ints to keep track of 10 minute max

volatile float rainHour[60]; //60 floating numbers to keep track of 60 minutes of rain

//These are all the weather values that wunderground expects:

int winddir = 0; // [0-360 instantaneous wind direction]

float windspeedmph = 0; // [mph instantaneous wind speed]

float windgustmph = 0; // [mph current wind gust, using software specific time period]

int windgustdir = 0; // [0-360 using software specific time period]

float windspdmph\_avg2m = 0; // [mph 2 minute average wind speed mph]

int winddir\_avg2m = 0; // [0-360 2 minute average wind direction]

float windgustmph\_10m = 0; // [mph past 10 minutes wind gust mph ]

int windgustdir\_10m = 0; // [0-360 past 10 minutes wind gust direction]

float humidity = 0; // [%]

float tempf = 0; // [temperature F]

float rainin = 0; // [rain inches over the past hour)] -- the accumulated rainfall in the past 60 min

volatile float dailyrainin = 0; // [rain inches so far today in local time]

//float baromin = 30.03;// [barom in] - It's hard to calculate baromin locally, do this in the agent

float pressure = 0;

//float dewptf; // [dewpoint F] - It's hard to calculate dewpoint locally, do this in the agent

float batt\_lvl = 11.8; //[analog value from 0 to 1023]

float light\_lvl = 455; //[analog value from 0 to 1023]

// volatiles are subject to modification by IRQs

volatile unsigned long raintime, rainlast, raininterval, rain;

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Interrupt routines (these are called by the hardware interrupts, not by the main code)

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

void rainIRQ()

// Count rain gauge bucket tips as they occur

// Activated by the magnet and reed switch in the rain gauge, attached to input D2

{

    raintime = millis(); // grab current time

    raininterval = raintime - rainlast; // calculate interval between this and last event

    if (raininterval > 10) // ignore switch-bounce glitches less than 10mS after initial edge

    {

        dailyrainin += 0.011; //Each dump is 0.011" of water

        rainHour[minutes] += 0.011; //Increase this minute's amount of rain

        rainlast = raintime; // set up for next event

    }

}

void wspeedIRQ()

// Activated by the magnet in the anemometer (2 ticks per rotation), attached to input D3

{

    if (millis() - lastWindIRQ > 10) // Ignore switch-bounce glitches less than 10ms (142MPH max reading) after the reed switch closes

    {

        lastWindIRQ = millis(); //Grab the current time

        windClicks++; //There is 1.492MPH for each click per second.

    }

}

void setup() {

    Serial.begin(115200);

    loraSerial.begin(115200);

    while (!Serial) { ; }

    delay(2000);

    Serial.println("Sending join command: at+join");

    RAKLoRa.sendRawCommand("at+join");

    delay(2000);

    //######################

    Serial.println("projet station meteo");

    pinMode(STAT1, OUTPUT); //Status LED Blue

    pinMode(STAT2, OUTPUT); //Status LED Green

    pinMode(WSPEED, INPUT\_PULLUP); // input from wind meters windspeed sensor

    pinMode(RAIN, INPUT\_PULLUP); // input from wind meters rain gauge sensor

    pinMode(REFERENCE\_3V3, INPUT);

    pinMode(LIGHT, INPUT);

    //Configure the pressure sensor

    myPressure.begin(); // Get sensor online

    myPressure.setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa

    myPressure.setOversampleRate(7); // Set Oversample to the recommended 128

    myPressure.enableEventFlags(); // Enable all three pressure and temp event flags

    //Configure the humidity sensor

    myHumidity.begin();

    sensor.begin();

    sensors.begin();

    pinMode(FC28\_PIN, INPUT);

    seconds = 0;

    lastSecond = millis();

    // attach external interrupt pins to IRQ functions

    attachInterrupt(0, rainIRQ, FALLING);

    attachInterrupt(1, wspeedIRQ, FALLING);

    // turn on interrupts

    interrupts();

    Serial.println("Weather Shield en ligne!");

}

void loop()

{

    static unsigned long lastSendTime = 0;

    //Keep track of which minute it is

  if(millis() - lastSecond >= 1000)

    {

        digitalWrite(STAT1, HIGH); //Blink stat LED

    lastSecond += 1000;

        //Take a speed and direction reading every second for 2 minute average

        if(++seconds\_2m > 119) seconds\_2m = 0;

        // mesure csensors soil and temperature soil

        readSensors();

        //Calc the wind speed and direction every second for 120 second to get 2 minute average

        float currentSpeed = get\_wind\_speed();

        windspeedmph = currentSpeed;//update global variable for windspeed when using the printWeather() function

        //float currentSpeed = random(5); //For testing

        int currentDirection = get\_wind\_direction();

        windspdavg[seconds\_2m] = (int)currentSpeed;

        winddiravg[seconds\_2m] = currentDirection;

        //if(seconds\_2m % 10 == 0) displayArrays(); //For testing

        //Check to see if this is a gust for the minute

        if(currentSpeed > windgust\_10m[minutes\_10m])

        {

            windgust\_10m[minutes\_10m] = currentSpeed;

            windgustdirection\_10m[minutes\_10m] = currentDirection;

        }

        //Check to see if this is a gust for the day

        if(currentSpeed > windgustmph)

        {

            windgustmph = currentSpeed;

            windgustdir = currentDirection;

        }

        if(++seconds > 59)

        {

            seconds = 0;

            if(++minutes > 59) minutes = 0;

            if(++minutes\_10m > 9) minutes\_10m = 0;

            rainHour[minutes] = 0; //Zero out this minute's rainfall amount

            windgust\_10m[minutes\_10m] = 0; //Zero out this minute's gust

        }

        //Report all readings every second

        printWeather();

        digitalWrite(STAT1, LOW); //Turn off stat LED

    }

    // Send data every 2 minutes

    if(millis() - lastSendTime >= 3000) {

        lastSendTime = millis();

        sendLoRaData();

    }

  delay(4000);

}

//Calculates each of the variables that wunderground is expecting

void calcWeather()

{

    //Calc winddir

    winddir = get\_wind\_direction();

    //Calc windspeed

    //windspeedmph = get\_wind\_speed(); //This is calculated in the main loop on line 185

    //Calc windgustmph

    //Calc windgustdir

    //These are calculated in the main loop

    //Calc windspdmph\_avg2m

    float temp = 0;

    for(int i = 0 ; i < 120 ; i++)temp += windspdavg[i];

    temp /= 120.0;

    windspdmph\_avg2m = temp;

    //Calc winddir\_avg2m, Wind Direction

    //You can't just take the average. Google "mean of circular quantities" for more info

    //We will use the Mitsuta method because it doesn't require trig functions

    //And because it sounds cool.

    //Based on: http://abelian.org/vlf/bearings.html

    //Based on: http://stackoverflow.com/questions/1813483/averaging-angles-again

    long sum = winddiravg[0];

    int D = winddiravg[0];

    for(int i = 1 ; i < WIND\_DIR\_AVG\_SIZE ; i++)

    {

        int delta = winddiravg[i] - D;

        if(delta < -180)

            D += delta + 360;

        else if(delta > 180)

            D += delta - 360;

        else

            D += delta;

        sum += D;

    }

    winddir\_avg2m = sum / WIND\_DIR\_AVG\_SIZE;

    if(winddir\_avg2m >= 360) winddir\_avg2m -= 360;

    if(winddir\_avg2m < 0) winddir\_avg2m += 360;

    //Calc windgustmph\_10m

    //Calc windgustdir\_10m

    //Find the largest windgust in the last 10 minutes

    windgustmph\_10m = 0;

    windgustdir\_10m = 0;

    //Step through the 10 minutes

   for(int i = 0; i < 10 ; i++)

    {

        if(windgust\_10m[i] > windgustmph\_10m)

        {

            windgustmph\_10m = windgust\_10m[i];

            windgustdir\_10m = windgustdirection\_10m[i];

        }

    }

    //Calc humidity

    humidity = myHumidity.getRH();

    //float temp\_h = myHumidity.readTemperature();

    //Serial.print(" TempH:");

    //Serial.print(temp\_h, 2);

    //Calc tempf from pressure sensor

    tempf = sensor.getTempF();

   // Serial.print(" TempP:");

   // Serial.println(tempf, 2);

    //Total rainfall for the day is calculated within the interrupt

    //Calculate amount of rainfall for the last 60 minutes

    rainin = 0;

    for(int i = 0 ; i < 60 ; i++)

        rainin += rainHour[i];

    //Calc pressure

    pressure = myPressure.readPressure();

    //Calc light level

    light\_lvl = get\_light\_level();

    //Calc battery level

    batt\_lvl = get\_battery\_level();

}

void readSensors() {

  sensors.requestTemperatures();

  temperature = sensors.getTempCByIndex(0);

  int rawHumidity = analogRead(FC28\_PIN);

  soilHumidity = constrain(map(rawHumidity, SEC\_VALUE, WET\_VALUE, 0, 100), 0, 100);

  Serial.print("Temp: ");

  Serial.print(temperature, 1);

  Serial.print("°C | Humidité: ");

  Serial.print(soilHumidity);

  Serial.println("%");

}

//Returns the voltage of the light sensor based on the 3.3V rail

//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)

float get\_light\_level()

{

    float operatingVoltage = analogRead(REFERENCE\_3V3);

    float lightSensor = analogRead(LIGHT);

    operatingVoltage = 3.3 / operatingVoltage; //The reference voltage is 3.3V

    lightSensor = operatingVoltage \* lightSensor;

    return(lightSensor);

}

//Returns the voltage of the raw pin based on the 3.3V rail

//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)

//Battery level is connected to the RAW pin on Arduino and is fed through two 5% resistors:

//3.9K on the high side (R1), and 1K on the low side (R2)

float get\_battery\_level()

{

    float operatingVoltage = analogRead(REFERENCE\_3V3);

    float rawVoltage = analogRead(BATT);

    operatingVoltage = 3.30 / operatingVoltage; //The reference voltage is 3.3V

    rawVoltage = operatingVoltage \* rawVoltage; //Convert the 0 to 1023 int to actual voltage on BATT pin

    rawVoltage \*= 20; //(3.9k+1k)/1k - multiple BATT voltage by the voltage divider to get actual system voltage

    return(rawVoltage);

}

//Returns the instataneous wind speed

float get\_wind\_speed()

{

    float deltaTime = millis() - lastWindCheck; //750ms

    deltaTime /= 1000.0; //Covert to seconds

    float windSpeed = (float)windClicks / deltaTime; //3 / 0.750s = 4

    windClicks = 0; //Reset and start watching for new wind

    lastWindCheck = millis();

    windSpeed \*= 1.492; //4 \* 1.492 = 5.968MPH

    /\* Serial.println();

     Serial.print("Windspeed:");

     Serial.println(windSpeed);\*/

    return(windSpeed);

}

//Read the wind direction sensor, return heading in degrees

int get\_wind\_direction()

{

   unsigned int adc;

    adc = analogRead(WDIR); // get the current reading from the sensor

    if (adc>380&&adc<460) return (0);

    //if (adc < 456) return (158);

    if (adc >460&&adc<550) return (45);

    //if (adc < 551) return (203);

    if (adc>550&&adc<690) return (90);

   // if (adc < 680) return (23);

    if (adc>690&&adc<790) return (315);

    //if (adc < 801) return (248);

    if (adc>790&&adc<890) return (135);

    //if (adc < 878) return (338);

    if (adc>890&&adc<910) return (270);

    //if (adc < 940) return (293);

    if (adc >910&&adc<960) return (225);

    if (adc>960&&adc<990) return (180);

    return (-1); // error, disconnected?

}

//Prints the various variables directly to the port

//I don't like the way this function is written but Arduino doesn't support floats under sprintf

void printWeather()

{

    calcWeather(); //Go calc all the various sensors

    Serial.println();

    Serial.print("#,Direction du vent=");

    if(winddir==0)Serial.print(",  north wind (N)");

    //if(winddir==23)Serial.print("north-northeast wind (NNE)");

    if(winddir==45)Serial.print("northeast wind (NE)");

   // if(winddir==68)Serial.print("east-northeast wind (ENE)");

    if(winddir==90)Serial.print(" east wind (E)");

   // if(winddir==113)Serial.print(" east-southeast wind (ESE)");

    if(winddir==135)Serial.print(" southeast wind (SE)");

    //if(winddir==158)Serial.print("south-southeast wind (SSE)");

    if(winddir==180)Serial.print("south wind (S)");

    //if(winddir==203)Serial.print("south-southwest wind (SSW)");

    if(winddir==225)Serial.print("southwest wind (SW)");

   //if(winddir==248)Serial.print("north-northwest wind (NNW)");

    if(winddir==270)Serial.print(",  west wind (W)");

   //if(winddir==293)Serial.println(",  west-northwest wind (WNW)");

    if(winddir==315)Serial.print(",  northwest wind (NW)");

    //if(winddir==338)Serial.println(",  north-northwest wind (NNW)");

    Serial.print(winddir);//winddir

    Serial.println("°");

    Serial.print(",vitesse du vent=");

    Serial.print(windspeedmph\*1.60934, 1);

    Serial.println("m/h");

    Serial.print(",Rafale du vent=");

    Serial.print(windgustmph, 1);

    Serial.println("m/h");

    Serial.print(",Direction du Rafale du vent=");

    if(windgustdir==0)Serial.print(",  north wind (N)");

    //if(windgustdir==23)Serial.print("north-northeast wind (NNE)");

    if(windgustdir==45)Serial.print("northeast wind (NE)");

   // if(windgustdir==68)Serial.print("east-northeast wind (ENE)");

    if(windgustdir==90)Serial.print(" east wind (E)");

   // if(windgustdir==113)Serial.print(" east-southeast wind (ESE)");

    if(windgustdir==135)Serial.print(" southeast wind (SE)");

    //if(windgustdir==158)Serial.print("south-southeast wind (SSE)");

    if(windgustdir==180)Serial.print("south wind (S)");

    //if(windgustdir==203)Serial.print("south-southwest wind (SSW)");

    if(windgustdir==225)Serial.print("southwest wind (SW)");

   //if(windgustdir==248)Serial.print("north-northwest wind (NNW)");

    if(windgustdir==270)Serial.print(",  west wind (W)");

   //if(windgustdir==293)Serial.println(",  west-northwest wind (WNW)");

    if(windgustdir==315)Serial.print(",  northwest wind (NW)");

    //if(windgustdir==338)Serial.println(",  north-northwest wind (NNW)");

    Serial.print(windgustdir); Serial.println("°");

    Serial.print(",Vitesse du vent\_Moy2m=");

    Serial.print(windspdmph\_avg2m, 1);

    Serial.println("m/h");

    Serial.print(",Direction du vent\_Moy2m=");

    if(winddir\_avg2m>=349||winddir\_avg2m<11)Serial.print(",  north wind (N)");

    if(winddir\_avg2m>=11&&winddir\_avg2m<34)Serial.print("north-northeast wind (NNE)");

    if(winddir\_avg2m>=34&&winddir\_avg2m<56)Serial.print("northeast wind (NE)");

   if(winddir\_avg2m>=56&&winddir\_avg2m<79)Serial.print("east-northeast wind (ENE)");

    if(winddir\_avg2m>=79&&winddir\_avg2m<101)Serial.print(" east wind (E)");

    if(winddir\_avg2m>=101&&winddir\_avg2m<124)Serial.print(" east-southeast wind (ESE)");

    if(winddir\_avg2m>=124&&winddir\_avg2m<146)Serial.print(" southeast wind (SE)");

    if(winddir\_avg2m>=146&&winddir\_avg2m<169)Serial.print("south-southeast wind (SSE)");

    if(winddir\_avg2m>=169&&winddir\_avg2m<191)Serial.print("south wind (S)");

    if(winddir\_avg2m>=191&&winddir\_avg2m<214)Serial.print("south-southwest wind (SSW)");

    if(winddir\_avg2m>=214&&winddir\_avg2m<236)Serial.print("southwest wind (SW)");

   if(winddir\_avg2m>=236&&winddir\_avg2m<259)Serial.print("north-northwest wind (NNW)");

    if(winddir\_avg2m>=259&&winddir\_avg2m<281)Serial.print(",  west wind (W)");

   if(winddir\_avg2m>=281&&winddir\_avg2m<304)Serial.print(",  west-northwest wind (WNW)");

    if(winddir\_avg2m>=304&&winddir\_avg2m<326)Serial.print(",  northwest wind (NW)");

    if(winddir\_avg2m>=326&&winddir\_avg2m<349)Serial.print(",  north-northwest wind (NNW)");

    Serial.print(winddir\_avg2m);Serial.println("°");

    Serial.print(",Rafale du vent\_10m=");

    Serial.print(windgustmph\_10m, 1);

    Serial.println("m/h");

    Serial.print(",Direction du Rafale du vent\_10m=");

    if(windgustdir\_10m==0)Serial.print(",  north wind (N)");

    //if(windgustdir\_10m==23)Serial.print("north-northeast wind (NNE)");

    if(windgustdir\_10m==45)Serial.print("northeast wind (NE)");

   // if(windgustdir\_10m==68)Serial.print("east-northeast wind (ENE)");

    if(windgustdir\_10m==90)Serial.print(" east wind (E)");

   // if(windgustdir\_10m==113)Serial.print(" east-southeast wind (ESE)");

    if(windgustdir\_10m==135)Serial.print(" southeast wind (SE)");

    //if(windgustdir\_10m==158)Serial.print("south-southeast wind (SSE)");

    if(windgustdir\_10m==180)Serial.print("south wind (S)");

    //if(windgustdir\_10m==203)Serial.print("south-southwest wind (SSW)");

    if(windgustdir\_10m==225)Serial.print("southwest wind (SW)");

   //if(windgustdir\_10m==248)Serial.print("north-northwest wind (NNW)");

    if(windgustdir\_10m==270)Serial.print(",  west wind (W)");

   //if(windgustdir\_10m==293)Serial.println(",  west-northwest wind (WNW)");

    if(windgustdir\_10m==315)Serial.print(",  northwest wind (NW)");

    //if(windgustdir\_10m==338)Serial.println(",  north-northwest wind (NNW)");

    Serial.print(windgustdir\_10m);Serial.println("°");

    Serial.print(",Humidite=");

    Serial.print(humidity, 1);

    Serial.println("%,");

    Serial.print(",Temperature=");

    Serial.print(tempf, 1);

    Serial.print(" F,");

    Serial.print(((tempf-32)\*5)/9, 1);

    Serial.println(" °C,");

    Serial.print(",Pluie=");

    Serial.print(rainin, 2);

    Serial.println("   mm/h,");

    Serial.print(",Pluie quotidienne=");

    Serial.println(dailyrainin, 2);

    Serial.print(",Pression=");

    Serial.print(pressure, 2);

    Serial.print("Pa ,   ");

    Serial.print((pressure/100000), 2);

    Serial.println("Bar ,");

    Serial.print(",Niveau de batterie=");

    Serial.print(batt\_lvl, 2);

    Serial.println("%,");

    Serial.print(",Niveau de lumière=");

    Serial.print(light\_lvl\*48.5, 2);

    Serial.println("%,");

    Serial.print(",");

    Serial.println("#");

}

void sendLoRaData() {

    // 1. Get fresh measurements

    calcWeather();

    Serial.println("--- Preparing to Send LoRa Data ---");

    // 2. Prepare data (same scaling as before)

    uint16\_t windDir = winddir;

    uint16\_t windSpeed\_x10 = windspeedmph \* 1.60934 \* 10; // km/h \* 10

    int16\_t temperature\_x10 = ((tempf - 32) \* 5.0/9.0) \* 10; // Celsius \* 10

    uint8\_t humidity\_x2 = humidity \* 2; // % \* 2

    uint16\_t rainfall\_x10 = rainin \* 10; // mm \* 10

    uint16\_t pressure\_hPa = pressure / 100; // hPa

    // 3. Create ASCII payload (JSON-like format)

    String payload = "{";

    payload += "\"wd\":" + String(windDir) + ",";

    payload += "\"ws\":" + String(windSpeed\_x10) + ",";

    payload += "\"tp\":" + String(temperature\_x10) + ",";

    payload += "\"hm\":" + String(humidity\_x2) + ",";

    payload += "\"rn\":" + String(rainfall\_x10) + ",";

    payload += "\"pr\":" + String(pressure\_hPa) + ",";

    payload += "\"tm\":" + String(temperature\*10 , 0) + ",";

    payload += "\"hs\":" + String(soilHumidity);

    payload += "}";

    // 4. Debug output

    Serial.print("  ASCII Payload: ");

    Serial.println(payload);

    // 5. Send as hex-encoded ASCII (required by RAK811)

    String hexPayload = "";

    for (int i = 0; i < payload.length(); i++) {

        char c = payload[i];

        if (c < 0x10) hexPayload += "0"; // Pad single digits

        hexPayload += String(c, HEX);

    }

    String command = "at+send=lora:1:" + hexPayload;

    Serial.print("  Sending Command: ");

    Serial.println(command);

    // 6. Send the command

    RAKLoRa.sendRawCommand(command);

    Serial.println("--- LoRa Command Sent ---");

    // 7. Check response

    delay(500);

    while(loraSerial.available()) {

        Serial.write(loraSerial.read());

    }

}"

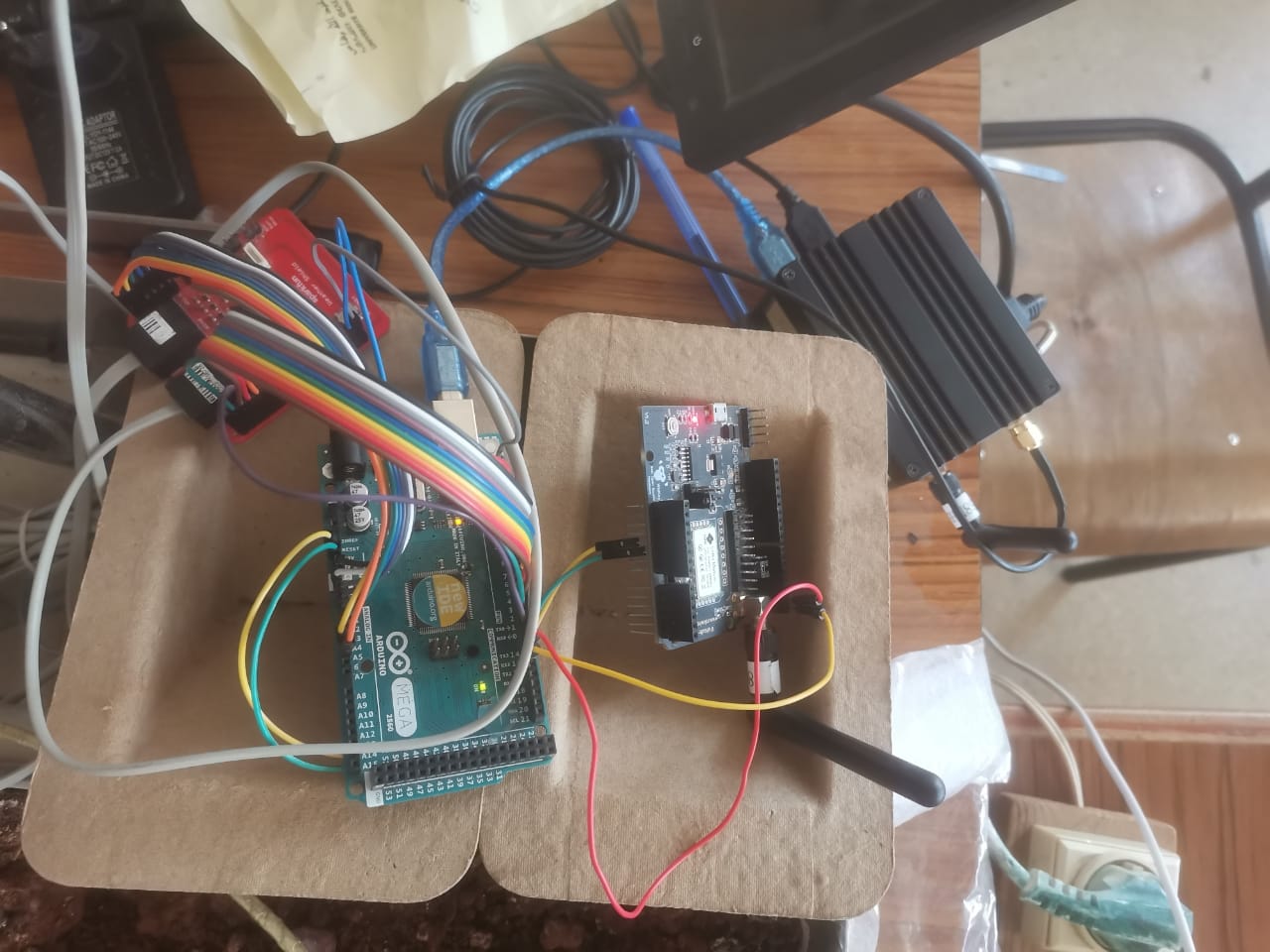
1. **Lecture et traitement des capteurs**

La station utilise une combinaison de capteurs :

* **SparkFun Weather Shield** : pour mesurer température (capteur MPL3115A2), humidité (SI7021), pression, pluie, vent.
* **FC-28** : pour l’humidité du sol.
* **Sonde Dallas DS18B20** : pour la température du sol.
* **Entrées analogiques** : pour la batterie et la luminosité.

Les interruptions matérielles sur les broches D2 et D3 permettent de détecter les impulsions du pluviomètre et de l’anémomètre avec précision.

Toutes les données sont calculées, moyennées (ex : rafale de vent sur 10 min), puis transformées en un format JSON-like avant envoi.





1. **Structure de la trame LoRa**

Le message transmis est préparé au format JSON, puis converti en hexadécimal pour être envoyé avec la commande AT :

**at+send=lora:1:<payload\_hex>**

Exemple de trame JSON générée :

*{"wd":315,"ws":85, "tp":245,"rn":12,"pr":1013,"tm":228,"hs":63}*

Chaque champ représente un paramètre environnemental (direction vent, vitesse, température, humidité, pluie, pression, etc.), avec une mise à l’échelle pour réduire la taille du message.



**Figure : Code de conversion de JSON vers Hexadécimale**

Code de décodage sur plateforme Chirpstack en Javascript

function Decode(fPort, bytes, variables) {

    try {

        // Convert bytes to string

        var payloadStr = String.fromCharCode.apply(null, bytes);

        // Handle different ports

        if (fPort === 1) {

            // Port 1: Weather data (JSON format)

            var data = JSON.parse(payloadStr);

            return {

                // Weather data

                wind\_direction: data.wd,

                wind\_speed\_kmh: data.ws / 10,

                temperature\_c: data.tp / 10,

                humidity\_percent: data.hm / 2,

                rainfall\_mm: data.rn / 10,

                pressure\_hpa: data.pr,

                Temperature\_soil: data.tm /10,

                Humdity\_soil: data.hs,

                Planet\_height: data.ht ,

                port: fPort,

            };

        } else {

            return {

                error: "Unknown port: " + fPort,

                raw\_payload: payloadStr

            };

        }

    } catch (e) {

        return {

            error: "Decoding failed: " + e.message,

            port: fPort,

            raw\_bytes: bytes,

            raw\_payload: String.fromCharCode.apply(null, bytes)

        };

    }

}

1. **Configuration du module RAK811 via RAK Serial Port Tool**

Le module RAK811 est configuré via l’outil officiel **RAK Serial Port Tool**. Ce logiciel permet d’envoyer des commandes AT au module pour définir les paramètres LoRaWAN.

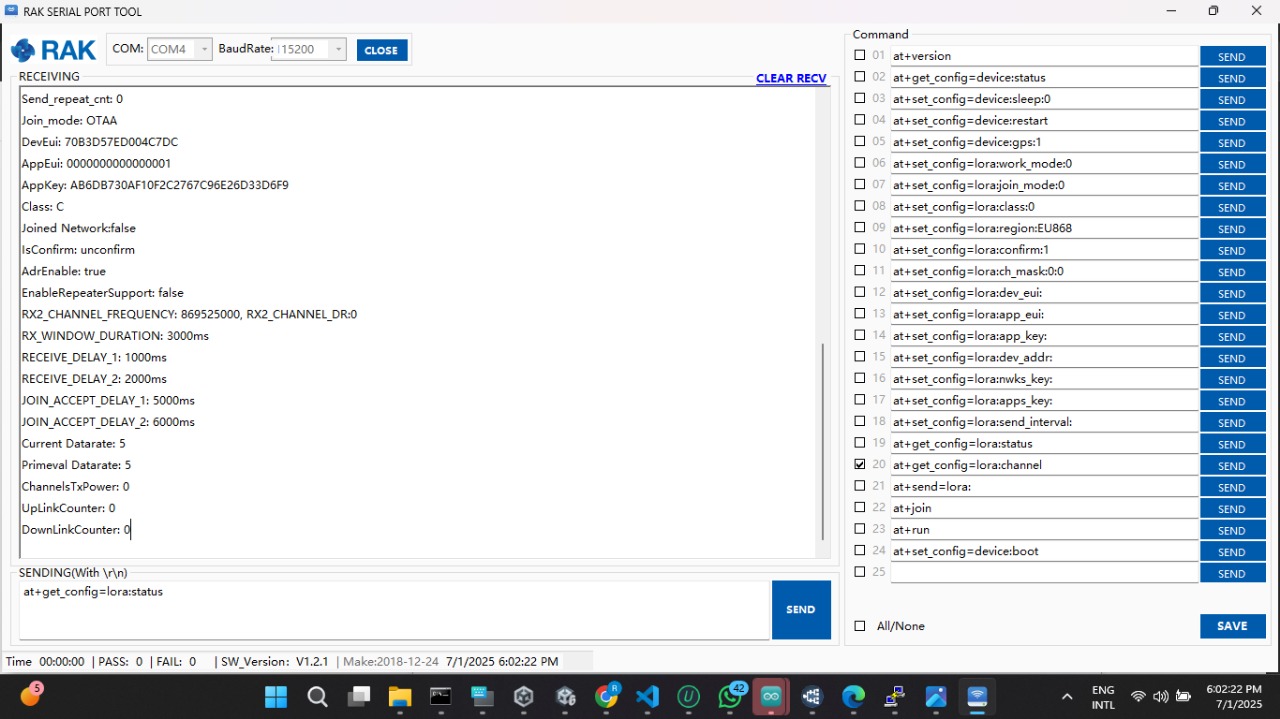
**a) Consultation des paramètres LoRa**

****Après connexion au port série, la commande suivante permet de consulter le statut du module :

**at+get\_config=lora:status**

Les paramètres affichés sont :

* **Mode** : LoRaWAN
* **Join Mode** : OTAA
* **DevEUI / AppEUI / AppKey** : identifiants uniques du nœud
* **Fréquence** : EU868
* **Classe** : C
* **Statut Join** : false (en attente de commande at+join)
* **IsConfirm** : unconfirm (transmission non confirmée)



**Figure : Détails supplémentaires RAKwireless : RX windows, datarate, délais, compteur uplink/downlink**

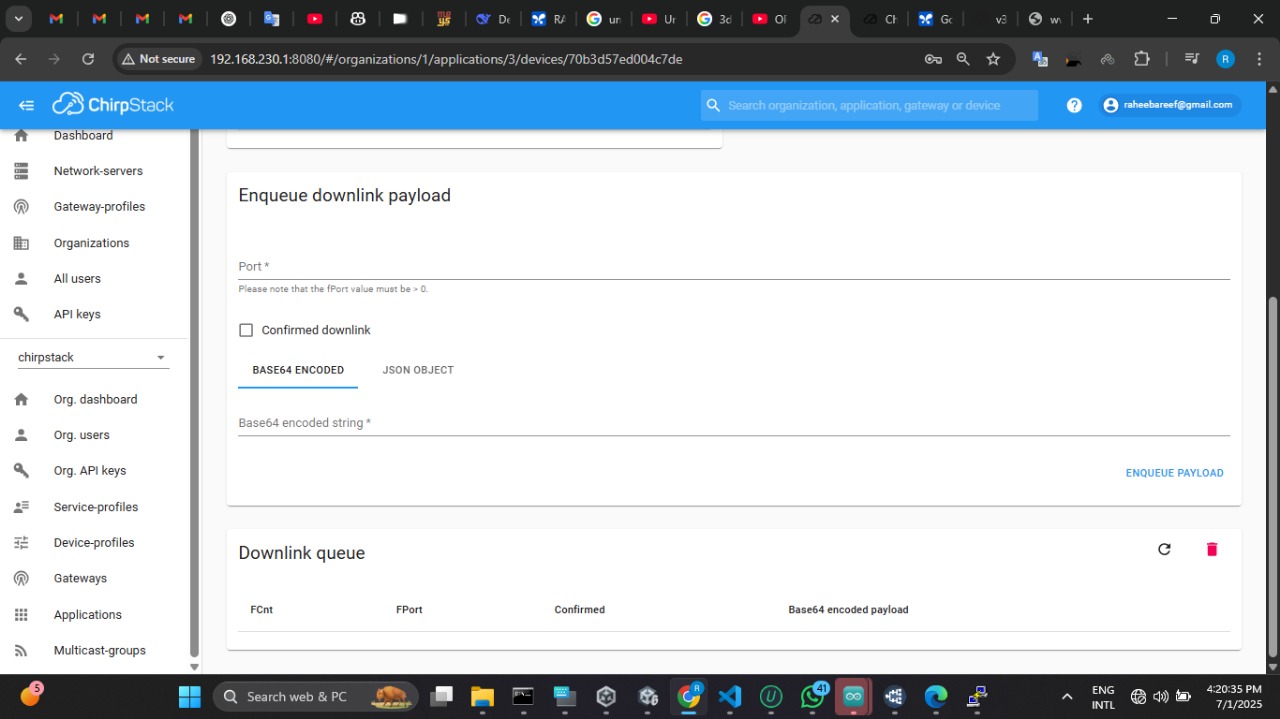
1. **Envoi de la commande at+join**

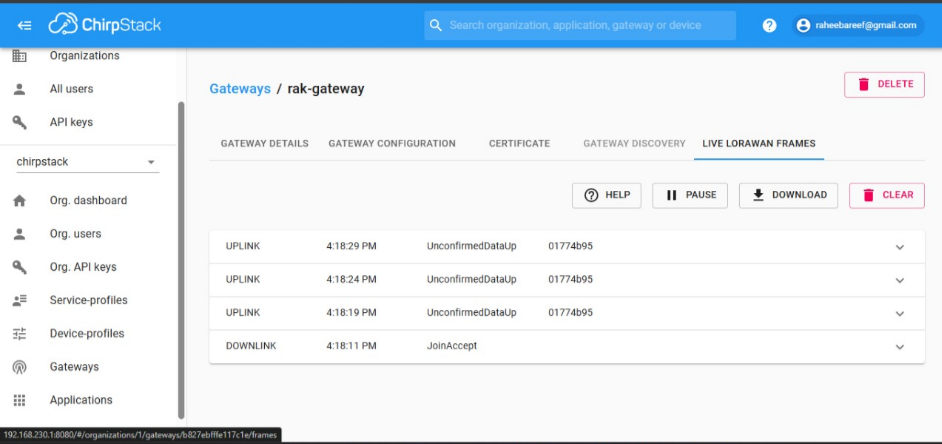
Une fois les paramètres vérifiés, la commande suivante est envoyée depuis l’Arduino (ou directement via l’outil RAK) :

**at+join**

Le module tente alors de se connecter au réseau LoRaWAN via la passerelle RAK7243. Une réponse OK Join Success valide l’enregistrement du nœud sur le serveur ChirpStack.

On voit sur chipstack un downlink dont « joinAccept » notre join est accepter





**Figure : test de message d’envoi sur Gateway depuis RAK8111**

1. **Objectif de la suite**

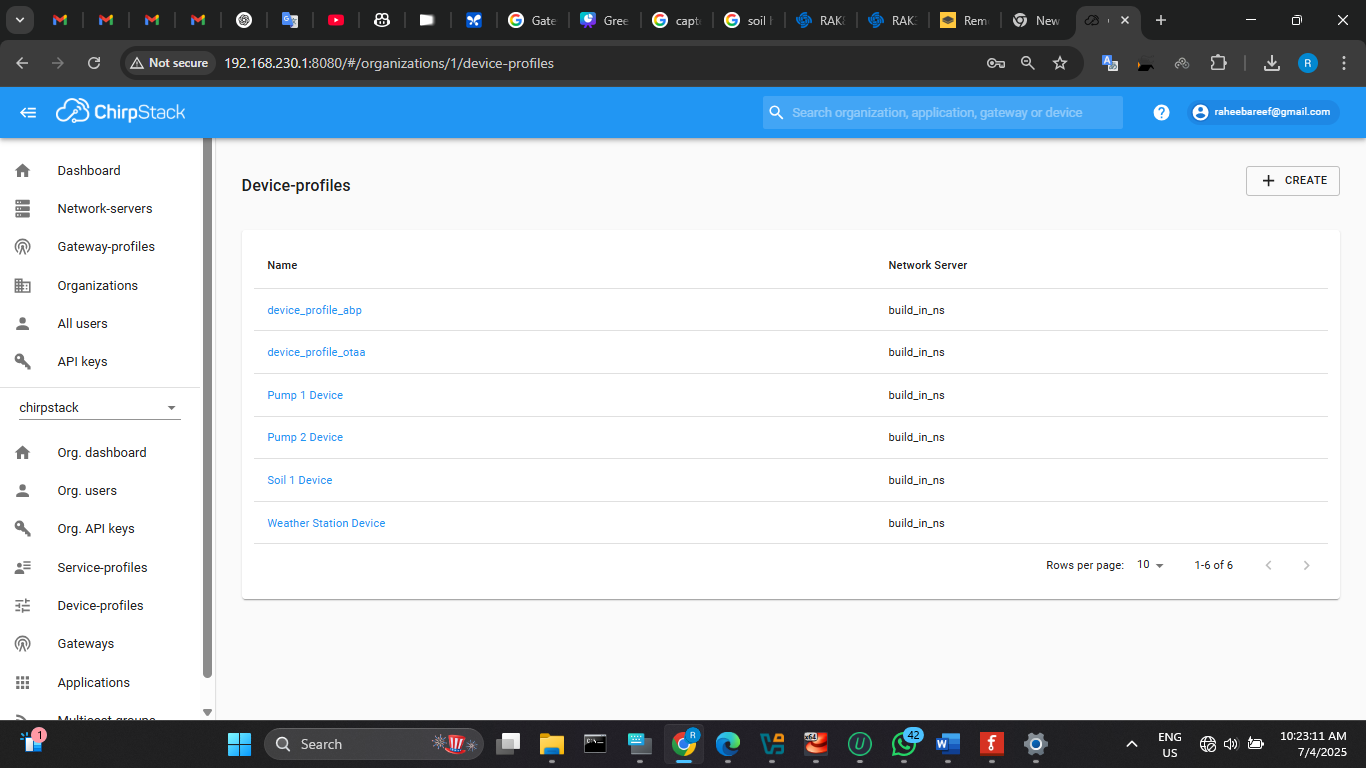
Une fois connecté, le module transmet les trames météo à la passerelle, qui les relaie vers ChirpStack. Dans la suite du rapport, nous détaillerons :

* L’enregistrement du device dans ChirpStack
* La configuration du payload decoder
* La publication dans MQTT et affichage via dashboard

1. **Configuration ChirpStack**

**Partie 1 : Gestion des Device-profiles dans ChirpStack**

Les **Device-profiles** dans ChirpStack sont des modèles de configuration prédéfinis permettant de standardiser les paramètres des appareils LoRaWAN. Ils simplifient le déploiement en évitant une configuration manuelle pour chaque device.



**Figure : Gestion des Device-profiles dans ChirpStack**

* 1. **Structure des Device-profiles**

Le tableau extrait de l’interface ChirpStack présente deux colonnes principales :

* **Name** : Identifie le profil (ex: device\_profile\_abp, RAK8111-Profile).
* **Network Server** : Indique le serveur LoRaWAN associé (ici, build\_in\_ns pour tous).
  1. **Types de Profils et Cas d’Usage**

1. **Profils ABP (Activation By Personalization)**
   * Utilisés pour une activation sécurisée hors réseau (clés pré-configurées).
   * Exemple : device\_profile\_abp.
2. **Profils OTAA (Over-The-Air Activation)**
   * Permettent une activation dynamique via le réseau (méthode plus flexible).
   * Exemple : device\_profile\_otaa.
3. **Profils Spécifiques (ex: RAK8111)**
   * Optimisés pour des modules particuliers (fréquences, puissances, etc.).

Pour ajouter un appareil RAK8111 :

1. Sélectionner RAK8111-Profile.
2. Le device hérite automatiquement des paramètres (fréquence, méthode d’activation, etc.).
3. Le trafic est géré par le build\_in\_ns (serveur intégré).
   1. **Configuration des périphériques, applications et analyse des trames montantes dans ChirpStack**

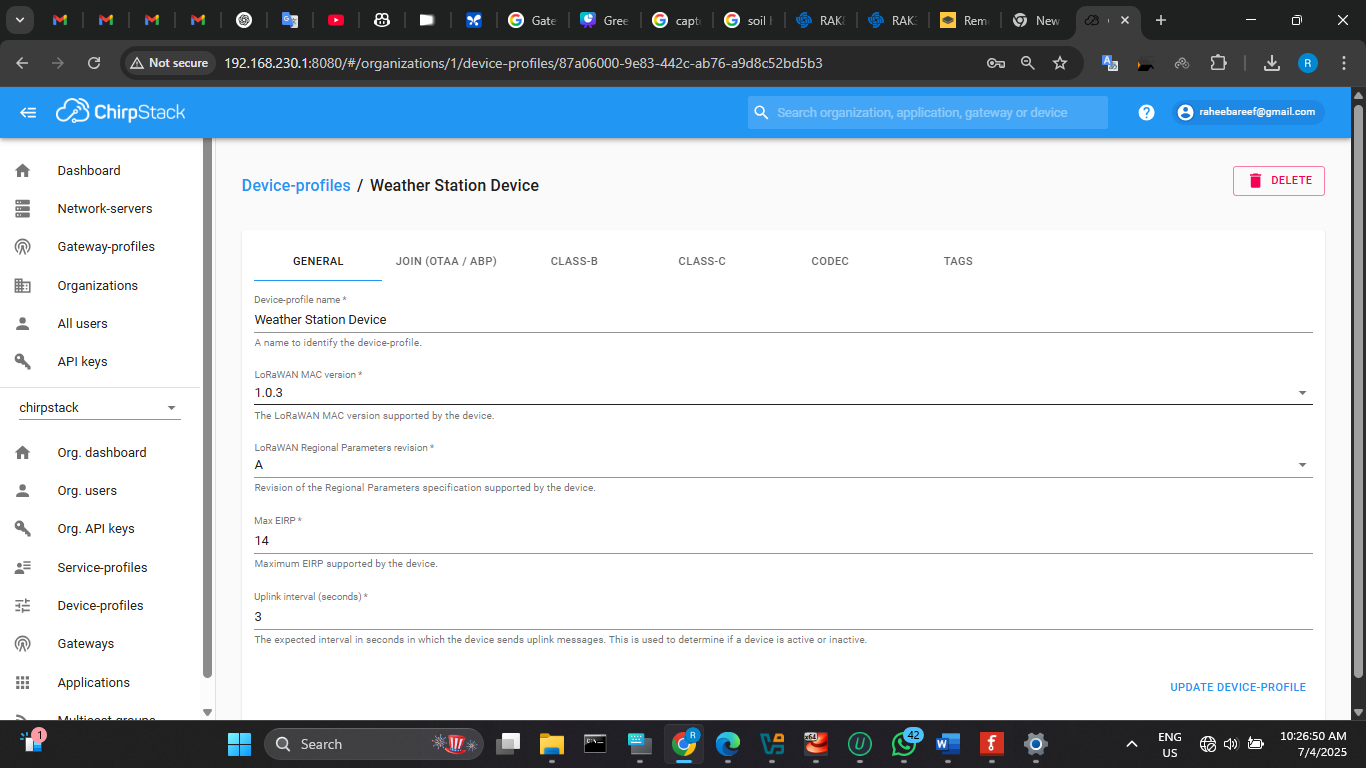
Cette section présente la configuration principale des périphériques LoRaWAN dans l’interface ChirpStack. Elle inclut également une analyse succincte des échanges de données entre les capteurs et le réseau.

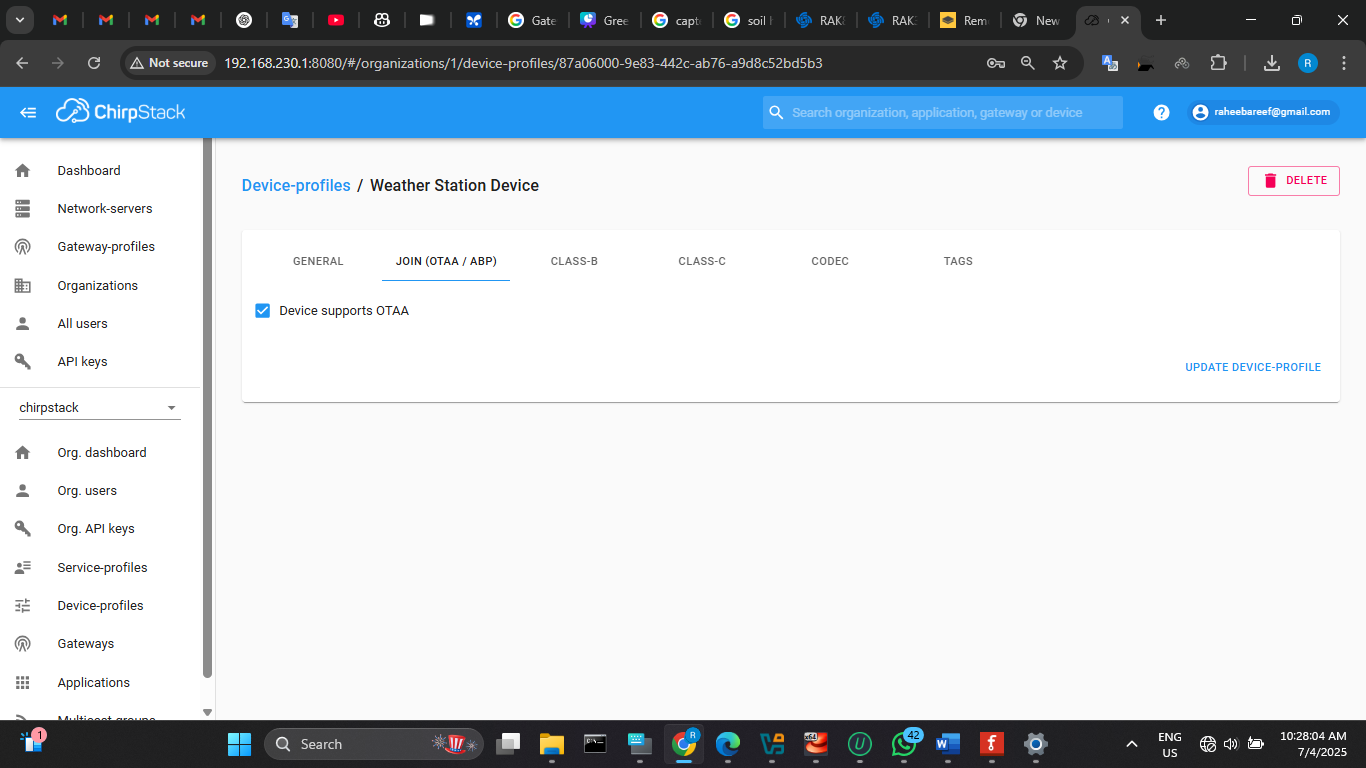
* 1. **Profil du périphérique (Device-profile)**

Le profil RAK8111-Profile définit les paramètres généraux utilisés par les périphériques RAK811 dans le réseau. Ces paramètres incluent :

* Version LoRaWAN : 1.0.3
* Puissance maximale d’émission : 14 dBm
* Intervalle entre deux messages montants : 3 secondes
* Mode d’activation : OTAA activé
* Classe C : activée

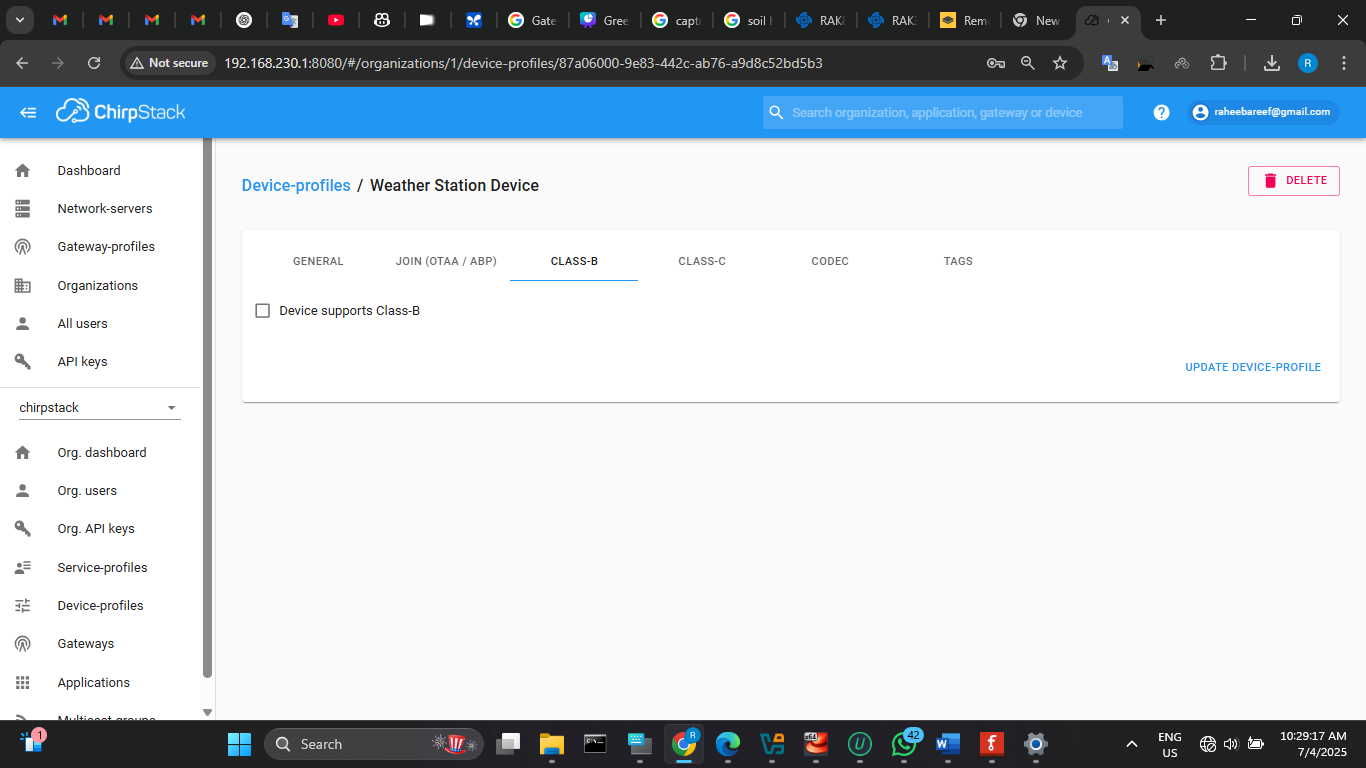
Ces réglages permettent au périphérique de fonctionner conformément aux normes LoRaWAN.





**Figure : Activation de OTAA**





**FIGURE : Activation de Classe C et désactivation de classe B**

* 1. **Périphérique (Device)**

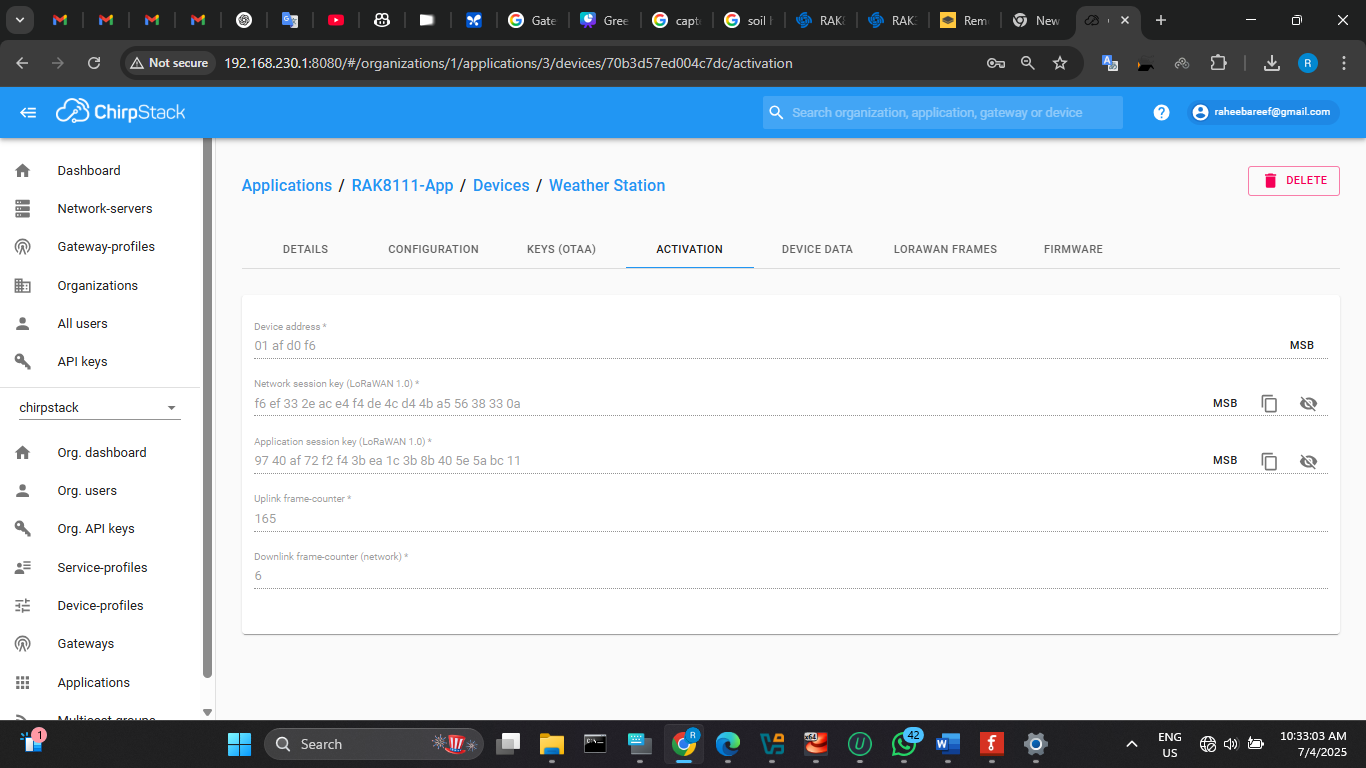
Le périphérique étudié porte le nom RAK8111-Node-01. Il utilise le profil décrit ci-dessus et est associé à l’application RAK8111-App.

**Clés de sécurité (OTAA) :**

* DevEUI : 01 77 4b 95
* NwkSKey et AppSKey : clés générées après l’activation OTAA
* Le compteur de trames montantes indique 20

La validation des compteurs de trames est active, ce qui protège contre les attaques de type "rejeu".



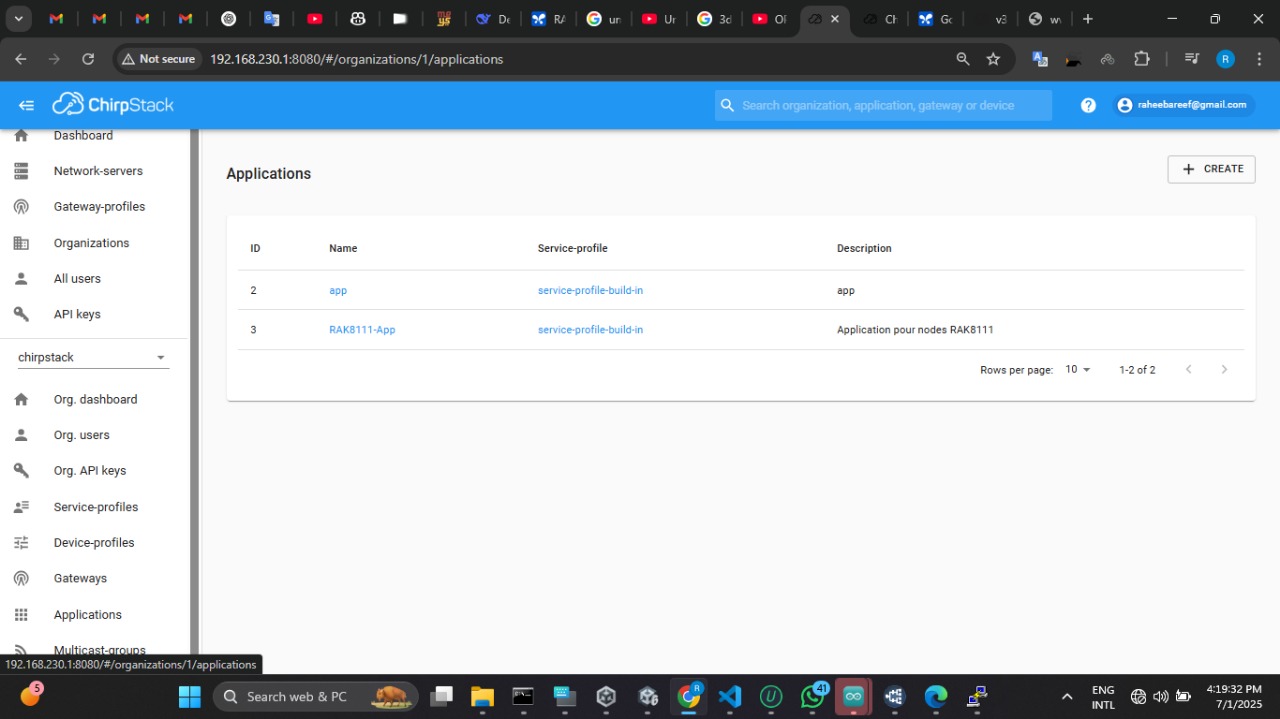


* 1. **Application**

Deux applications sont configurées :

* app : application générique
* RAK8111-App : dédiée aux périphériques RAK811

L’application RAK8111-App est liée au périphérique RAK8111-Node-01. Cela permet de regrouper les données provenant de ce capteur et de les traiter de manière organisée.



* 1. **Analyse des trames montantes (Uplink)**

L'analyse des trames montantes permet de comprendre les paramètres techniques et les données transmises par le périphérique vers la passerelle. Voici les détails observés :

a. Informations générales sur la trame montante

* Type de message : UnconfirmedDataUp
* Adresse du périphérique (DevAddr) : 01774b95
* Numéro de compteur de trame montante (fCntUp) : 3
* Port : 1
* Statut de CRC : CRC\_OK (indique que la trame a été correctement reçue sans erreur)

b. Données brutes (Payload)

* Données brutes (bytes) : EREHzaiivrsDUwAuK3CK6pNgEzM4hPdSMKozzbhkNeXDXQHJUpo64N5T2UTcOc5IJJRj
* Code MIC (Message Integrity Check) : 849b1c07

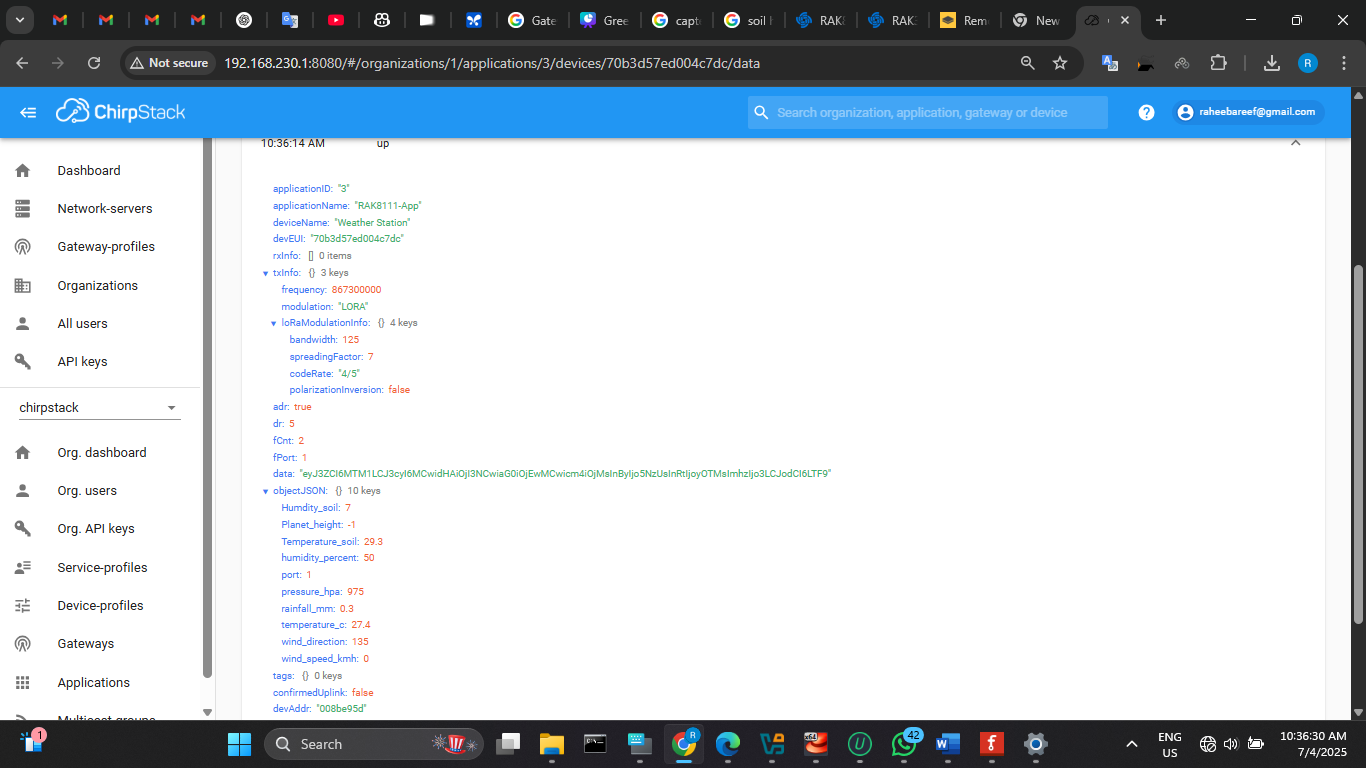
c. Informations sur la réception (rxInfo)

* ID de la passerelle : b827ebfffe117c1e
* SNR (Signal-to-Noise Ratio) : 9.2
* Canal utilisé : 1
* Fréquence : 868300000 Hz
* Modulation : LoRa
* Largeur de bande (bandwidth) : 125 kHz
* Facteur de diffusion (spreadingFactor) : 7
* Taux de codage (codeRate) : 4/5

d. Informations sur la transmission (txInfo)

* Antenne utilisée : 0
* Position GPS de la passerelle :
  + Latitude : 34.03508
  + Longitude : -4.97324
  + Altitude : 619

Ces informations fournissent un aperçu complet des conditions de transmission et de réception de la trame montante, ce qui est essentiel pour diagnostiquer les problèmes éventuels liés à la couverture ou à la qualité du signal.

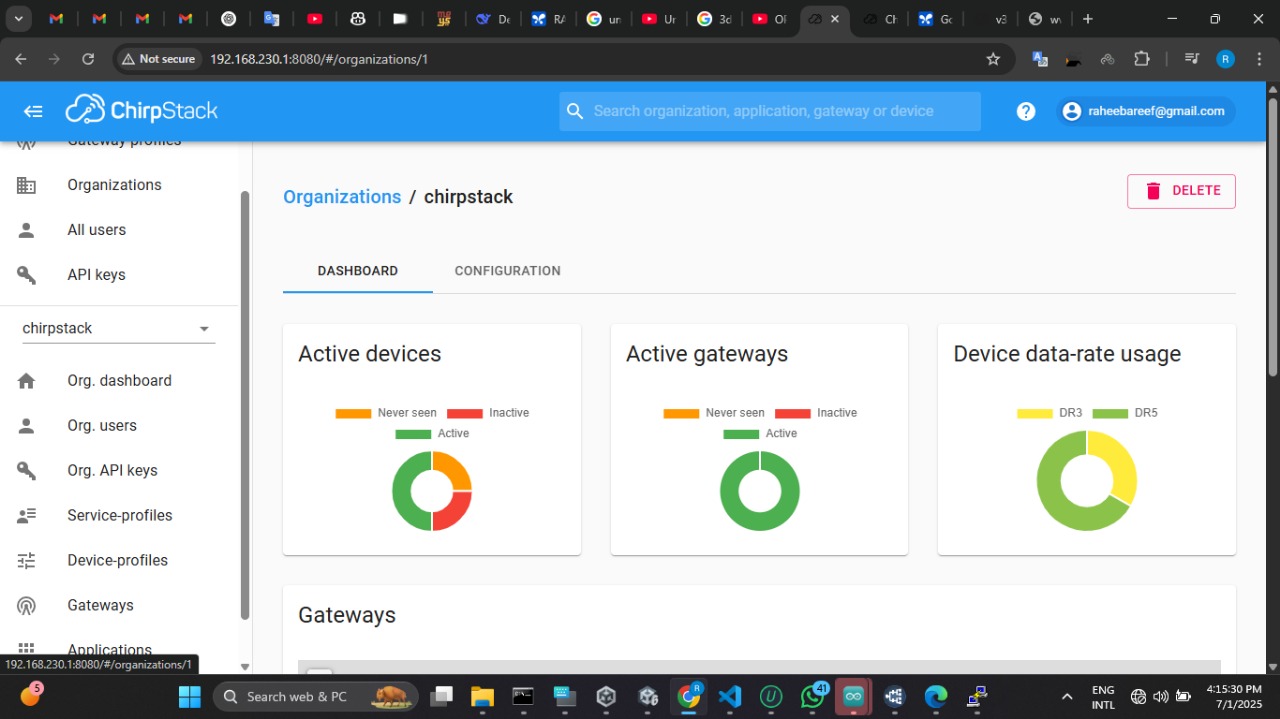


1. **Résultats obtenus**

La mise en œuvre de la station météo a permis d’obtenir un système fonctionnel, fiable et bien intégré dans l’architecture globale de la serre connectée. Grâce à l’interconnexion entre les capteurs, le module LoRa, la passerelle RAK7243, ChirpStack et le broker MQTT, nous avons pu valider toutes les étapes de la chaîne de communication, depuis la mesure jusqu’à la réception des données.

1. **Réception des données sur ChirpStack**

Une fois la configuration terminée, les paquets LoRa transmis par l’Arduino ont été reçus correctement sur la passerelle RAK7243, puis transmis à ChirpStack. Dans l’interface web de ChirpStack, les paquets uplink sont visibles, avec le DevEUI du nœud, l’heure de réception, la qualité du signal (RSSI, SNR) et le payload brut encodé (souvent en Base64 ou JSON).

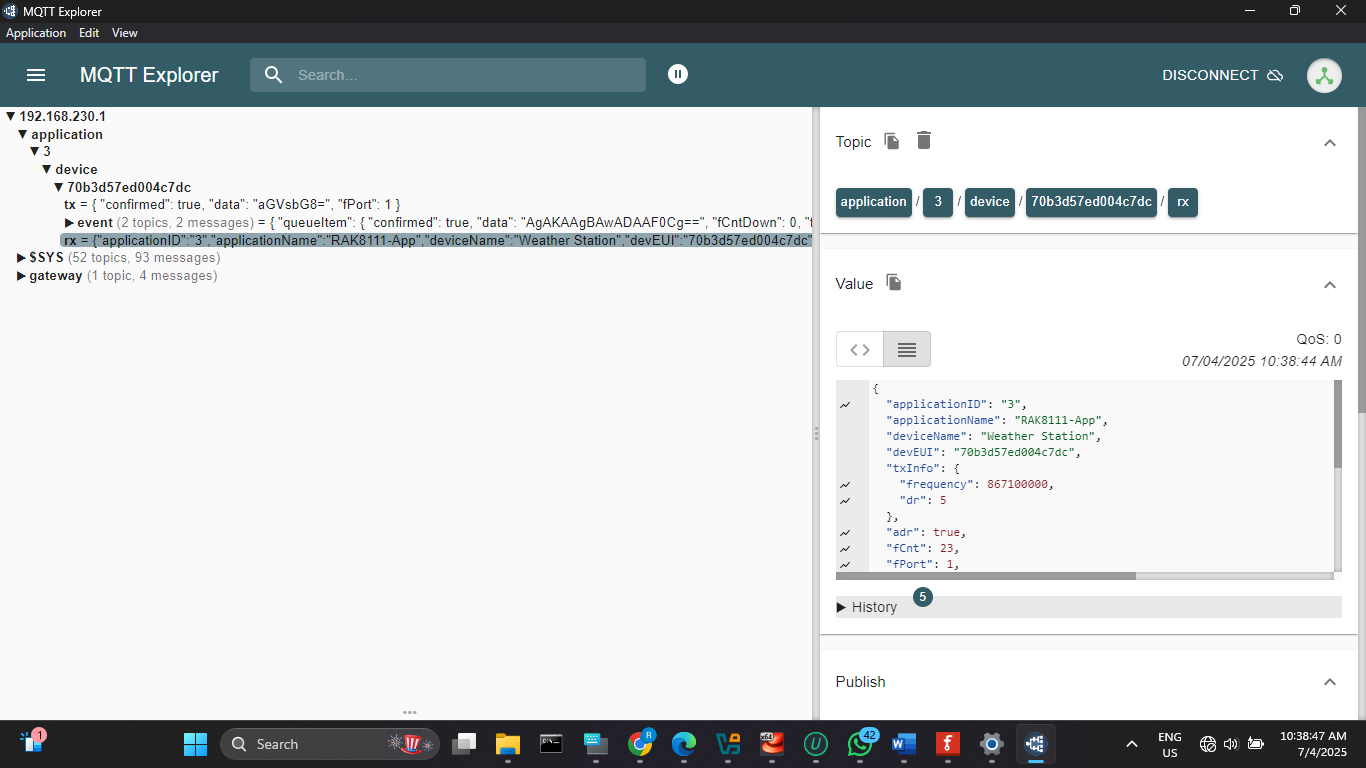


1. **Visualisation des données sur MQTT**

Les messages générés par les capteurs et transmis via LoRa sont réceptionnés par ChirpStack, puis publiés automatiquement sur un topic MQTT. En utilisant le logiciel MQTT Explorer, nous avons pu visualiser en direct les messages arrivant sur le broker local.  
Les messages sont décodés pour afficher les valeurs de température, humidité, pression, vitesse du vent, direction du vent et pluviométrie.

Le topic utilisé est le suivant :

**« application/3/device/70b3d57ed004c7dc/rx »**



🡺 Données météo en format JSON

Une fois le message décodé, les mesures envoyées apparaissent dans un format JSON clair, contenant toutes les valeurs mesurées :

* Température ambiante : 27.2 °C
* Humidité relative : 50 %
* Pression : 975 hPa
* Direction du vent : 135°
* Vitesse du vent : 0 km/h
* Pluviométrie : 10 mm
* Température du sol : 29.7 °C
* Humidité du sol : 7 %

{

"applicationID": "3",

"applicationName": "RAK8111-App",

"deviceName": "Weather Station",

"devEUI": "70b3d57ed004c7dc",

"txInfo": {

"frequency": 868100000,

"dr": 5

},

"adr": true,

"fCnt": 40,

"fPort": 1,

"data": "eyJ3ZCI6MTM1LCJ3cyI6MCwidHAiOjI3MiwiaG0iOjEwMCwicm4iOjEwNywicHIiOjk3NSwidG0iOjI5NywiaHMiOjcsImh0IjotMX0=",

"object": {

"Humdity\_soil": 7,

"Planet\_height": -1,

"Temperature\_soil": 29.7,

"humidity\_percent": 50,

"port": 1,

"pressure\_hpa": 975,

"rainfall\_mm": 10.7,

"temperature\_c": 27.2,

"wind\_direction": 135,

"wind\_speed\_kmh": 0

}

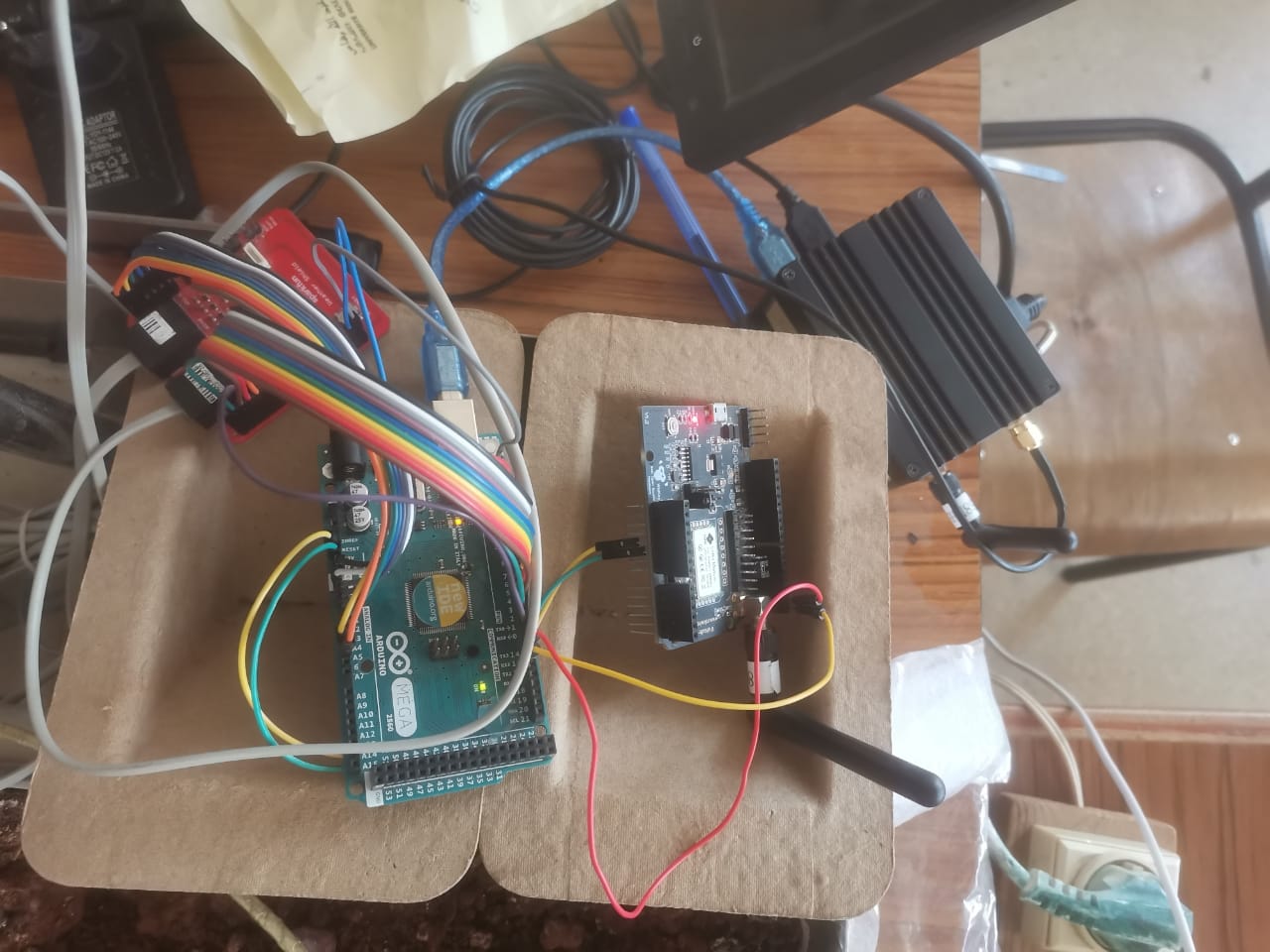
}

▶ History32

1. **Réalisation physique du prototype**

Le prototype a été entièrement monté en conditions réelles. On y retrouve l’Arduino Mega, la SparkFun Weather Shield, les capteurs météo externes (anémomètre, girouette, pluviomètre), le module LoRa ainsi que le système d’alimentation. Tous les composants ont été correctement câblés, testés, et protégés pour une utilisation en environnement extérieur.





1. **Coclusion**

La réalisation de la station météo a permis de mettre en œuvre une solution complète de mesure et de transmission des données climatiques, essentielle au bon fonctionnement de la serre intelligente. Grâce à l’intégration des capteurs météorologiques avec la carte Arduino Mega et la carte SparkFun Weather Shield, les paramètres environnementaux tels que la température, l’humidité, la pression atmosphérique, la pluviométrie, ainsi que la vitesse et la direction du vent sont mesurés avec précision.

La communication sans fil via LoRa, combinée à la passerelle RAK7243 et à la plateforme ChirpStack, a permis une transmission fiable des données vers un serveur central, puis leur publication sur un broker MQTT. Ces données sont désormais accessibles en temps réel par les autres modules de la serre, facilitant ainsi l’automatisation de l’irrigation, de la ventilation ou du chauffage selon les conditions extérieures.