CHAN HAOR KING Channia
ALLARD DE GRANDMAISON Antoine

Rapport SAE LORA

TABLE DES MATIÈRES

PARTIE 1 : LORAWAN

1.1 Schéma de la solution

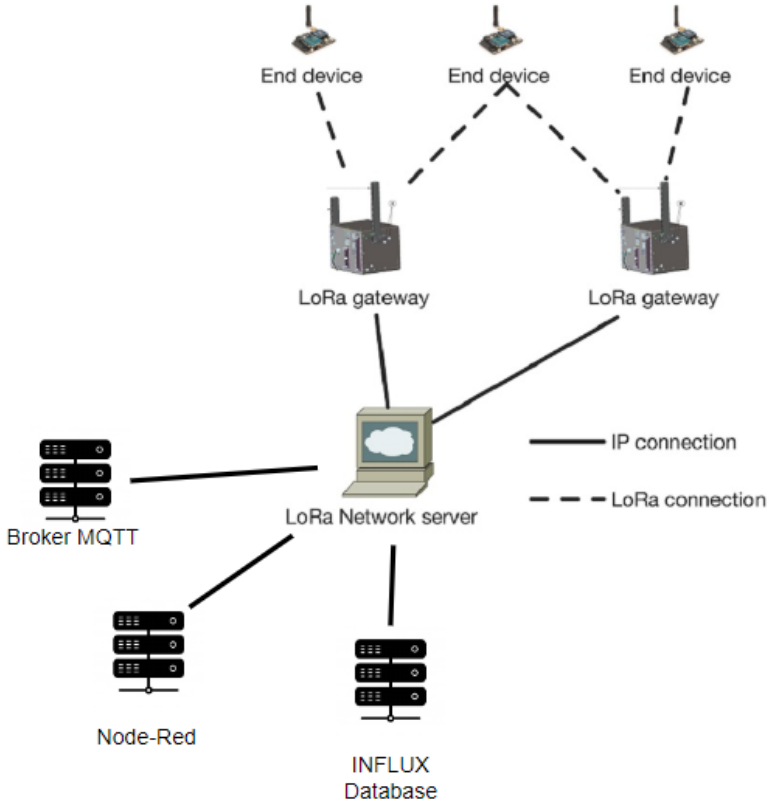


Schéma de principe du système LORAWAN

Le LoRaWAN (Long Range Wide Area Network) est un protocole de communication sans fil à basse consommation d'énergie conçu pour permettre la connectivité longue portée des appareils IoT (Internet of Things) sur de vastes zones géographiques.

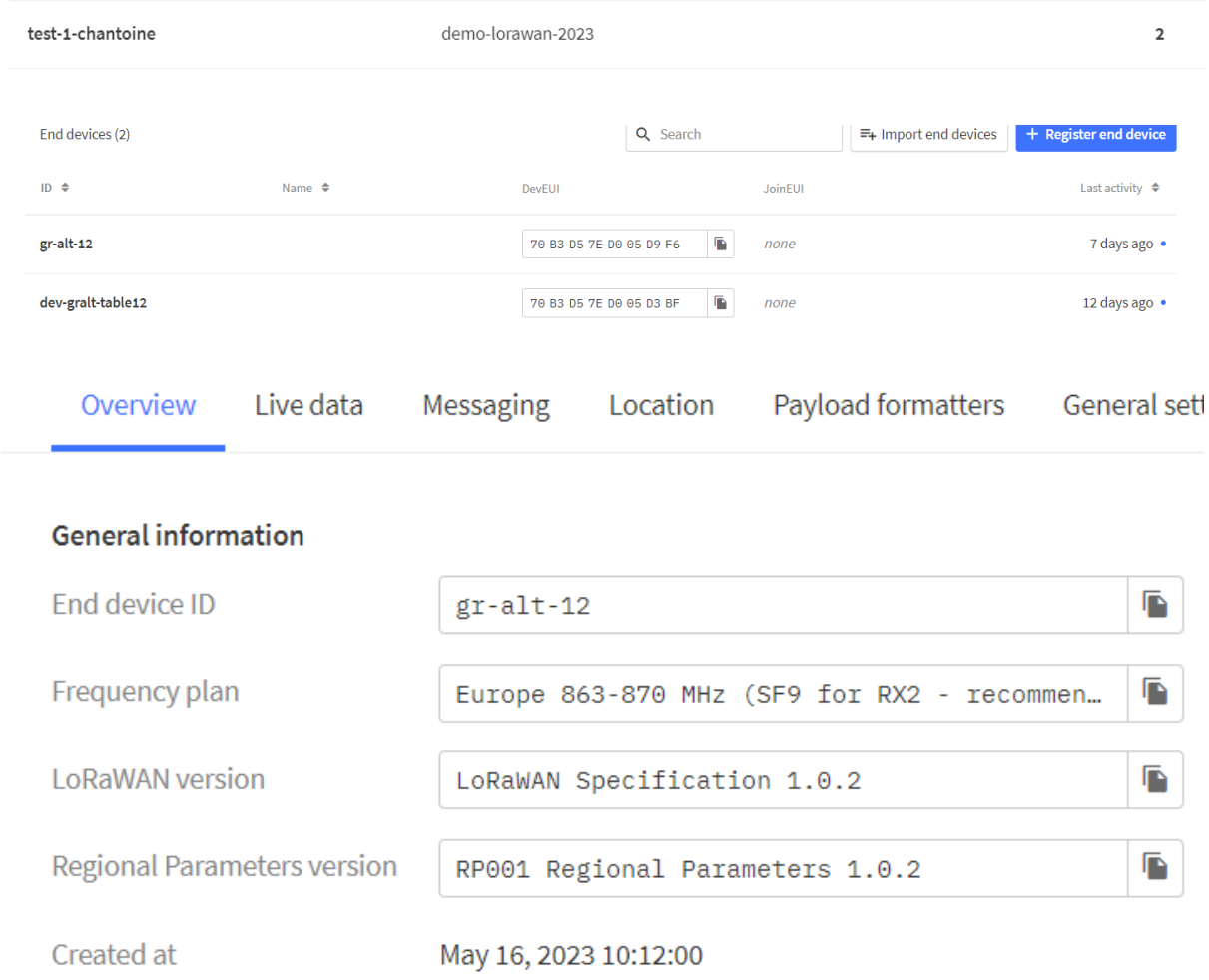
Le LoRaWAN est basée sur la technologie de modulation LoRa (Long Range), pour communiquer sur des fréquences de spectre librement accessibles. Les dispositifs envoient des messages à des passerelles responsables de la réception des signaux et de leur transmission vers le réseau.

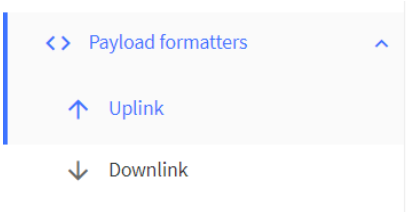
Le LoRaWAN est utilisé dans de nombreux domaines tels que les villes intelligentes.

1.2 TTN (MQTT et LoraWan)

TTN : The Things Network est un réseau LoRaWAN ouvert et décentralisé. Il fournit une infrastructure gratuite et ouverte pour connecter les appareils IoT. Les utilisateurs peuvent créer des applications sur la plateforme TTN pour collecter, traiter et analyser les données provenant des appareils connectés.

MQTT : Message Queuing Telemetry Transport est un protocole de messagerie pour les applications IoT. MQTT suit un modèle de publication/abonnement où les appareils clients, appelés abonnés, s'abonnent à des sujets spécifiques pour recevoir les messages publiés par les appareils émetteurs, appelés éditeurs. Les messages sont acheminés via un broker MQTT qui se charge de la distribution des messages aux abonnés. C’est également une communication bidirectionnelle.





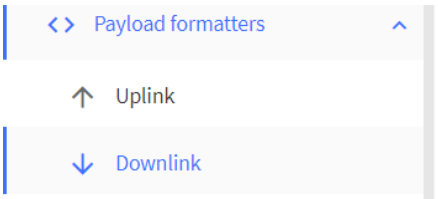
Dans le paramètre payload formateur, dans la section Uplink, nous avons mis le code ci dessous.

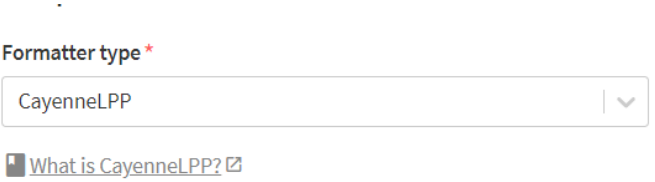
Formatter code *

```
function decodeUplink(input) {
  var bytes=input.bytes;
  var temperature =  bytes[2] << 24 >> 16 | bytes[3];
  var humidity = bytes[6];
  var longitude= bytes[12] << 16 | bytes[13] << 8 | bytes[14];
  var latitude = bytes[9] << 16 | bytes[10] << 8 | bytes[11];
  var altitude = bytes[15] << 16 | bytes[16] << 8 | bytes[17];
  var RFID=((bytes[19]>>4&0x0F)).toString(16)+((bytes[19]&0x0F)).toString(16)+((bytes[20]>>4&0x0F)).toString(16)+((bytes[20]&0x0F)).toSt
  var vec_id=((bytes[24]>>4&0x0F)).toString(16)+((bytes[24]&0x0F)).toString(16)+((bytes[25]>>4&0x0F)).toString(16)+((bytes[25]&0x0F)).to
 return {
    data: {
    temperature : temperature/10,
    humidite : humidity/2,
    latitude : latitude/10000,
    longitude : longitude/10000,
    altitude : altitude/100,
    rfid : RFID,//.toString(16)
    vehicule: vec_id

    },
    warnings: [],
    errors: []
  };
}
```

Ce code va traîter les données reçues par TTN depuis le device Mbed, puis en les “triant” les affections à des variables que nous allons retourner à Node Red. L’objectif ici est de faciliter le traitement des données depuis Node Red, duquel nous serons donc capable de retirer chaque information indépendamment des autres. (Par exemple dans Node Red nous pourrons facilement extraire la data ‘*Temperature*’)





Cayenne est une plateforme de développement IoT. Cayenne fournit une interface pour connecter et configurer des objets connectés, ainsi que pour visualiser et gérer les données collectées. Il est possible de surveiller les données en temps réel grâce à un tableau de bord comme suit :

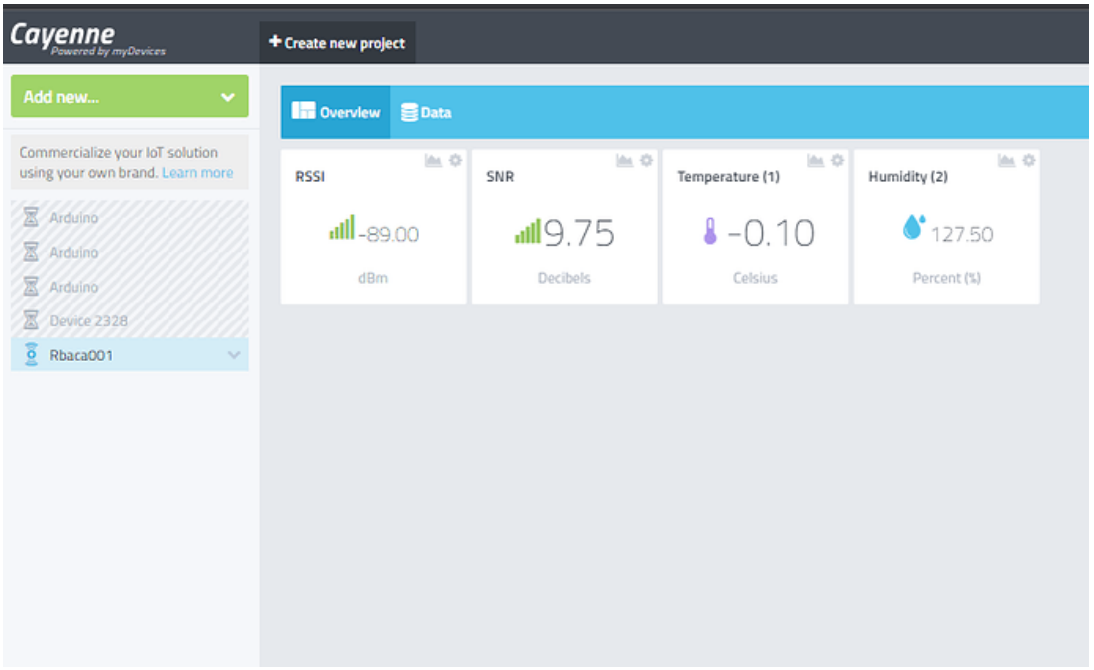


Tableau de bord sur Cayenne

À travers TTN nous allons recevoir les données venant du device MBED (abordé plus tard dans ce rapport), les traiter, ou trier, grâce au Payload Formater, puis les transmettre à Node Red dans lequel on pourra utiliser chaque information indépendamment.

1.3 La base de données InfluxDB

InfluxDB est une base de données conçue pour le stockage, la recherche et la récupération de données temporelles. Elle est optimisée pour la capture et l'analyse de données qui évoluent au fil du temps. Cette base de données est utilisée pour stocker et gérer des volumes massifs de données générées par les appareils IoT.

Premièrement, il faut créer la base de données avec InfluxDb grâce à la requête CREATE DATABASE [le nom de notre DB].
L'image suivante nous montre comment voir quelles databases sont présentes sur notre machine, ici il y en a deux : *gr12_tpres3* et *GRALT_2023*.

```
C:\Users\Public\Desktop\influx.exe
Connected to http://localhost:8086 version 1.7.3
InfluxDB shell version: 1.7.3
Enter an InfluxQL query
> show databases
name: databases
name
----
_internal
gr12_tpres3
GRALT_2023
```

```
> use GRALT_2023
Using database GRALT_2023
> show measurements
name: measurements
name
----
BADGES
demo_gr12_sae
> select * from BADGES
name: BADGES
time                BADGE_ID DEM
----                -------- ---
1687422858973281900 7d7b7889 1
1687422895782101100 8460ed00 0
1687506613049528700 43054402 1
>
```

Nous devons ensuite utiliser la base de données pour y effectuer davantage d'action, grâce à la première requête USE [nom DB] .
La requête suivante SHOW MEASUREMENTS permet de voir chaque table existant dans la base de données. Ici dans la base de données GRALT_2023 nous avons deux tables : la table BADGES et la table demo_gr12_sae .
Enfin nous souhaitons voir les données enregistrées dans la table BADGES grâce à la requête SELECT * FROM [nom de la table] .

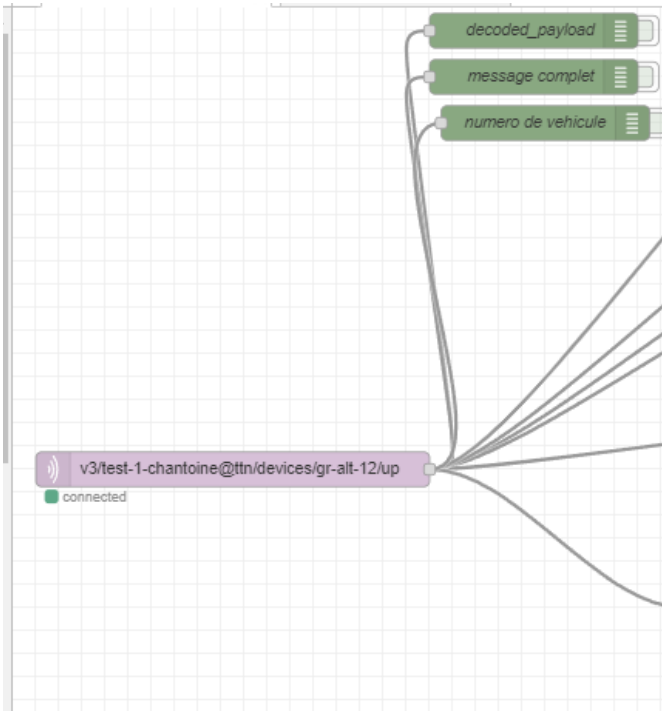
```
> select * from demo_gr12_sae
name: demo_gr12_sae
time                humidity temperature
----                -------- -----------
1687426160699648400 44       30.3
1687426177217776300 44.5     30.3
1687426186156827300 44       30.3
```

Ici nous lisons donc les données de la table demo_gr12_sae . Nous lisons les données triées et envoyées par le bloc "remplissage database" suivi du bloc InfluxDB OUT que nous allons étudier par la suite.

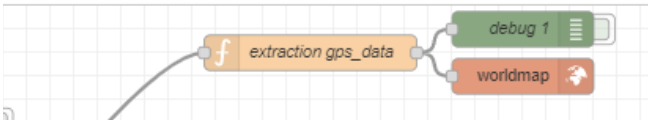
Toutes ces requêtes, que nous effectuons directement dans l'invite de commande de la base de données, nous allons par la suite avoir pour objectif de les envoyer à travers Node Red de manière autonome.

1.4 Node Red

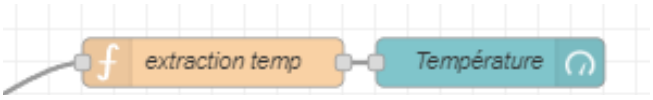
Node-RED est un environnement de développement graphique. Il permet de connecter des dispositifs, à l'aide de blocs fonctionnels prédéfinis appelés "nodes" et de les assembler visuellement en les connectant.



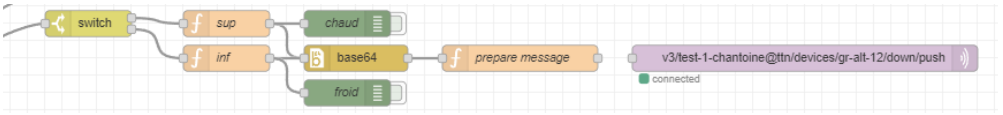
Ici nous voyons tout à gauche le bloc qui va nous permettre de se connecter à TTN, et au device TTN qui reçoit les données. Le bloc est un *MQTT IN*, il a fonction de réception d'un payload venant donc ici du device "gr-alt-12" dans l'application "test-1-chantoine" .



Ici nous extrayons les données GPS dans le payload grâce à une fonction qui va stocker dans des variables les latitude, longitude, et altitude. Le bloc worldmap va créer une interface qui va nous permettre de voir sur une carte en temps réelle la position ainsi extraite.

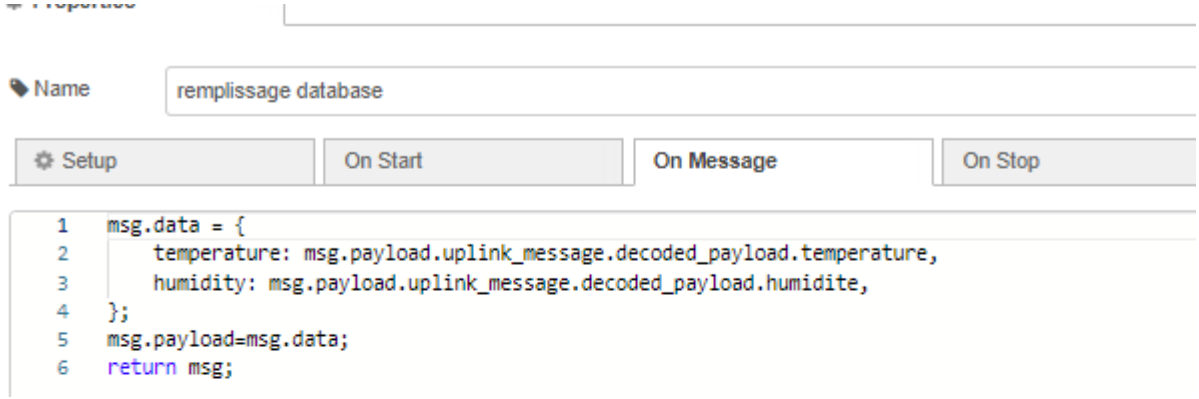


Dans le même principe, nous allons ici extraire la température pour enfin l'afficher sur une autre interface qui va nous permettre d'afficher une jauge.

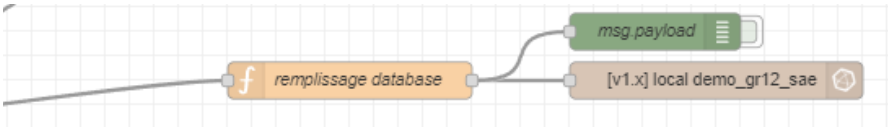


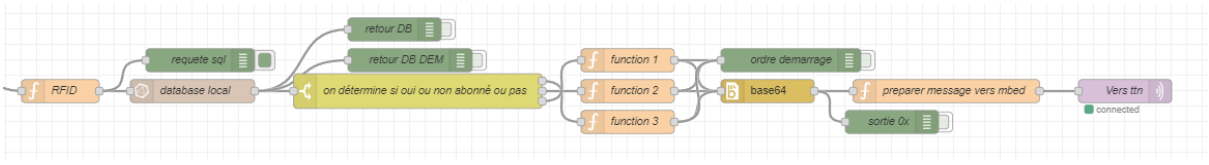
Le bloc switch ici va nous permettre d'activer une sortie ou l'autre en fonction d'une condition. Ici la condition est liée à la température. Si nous sommes supérieur ou inférieur,

nous allons activer le bloc correspondant et créer une variable, qui sera convertie en hexadécimale (base64) et stockée dans une variable qui sera envoyé vers TTN (bloc *MQTT OUT*) puis vers le MBED qui devra agir en fonction.
Ce scénario permet de déterminer si la température est froide ou chaude.

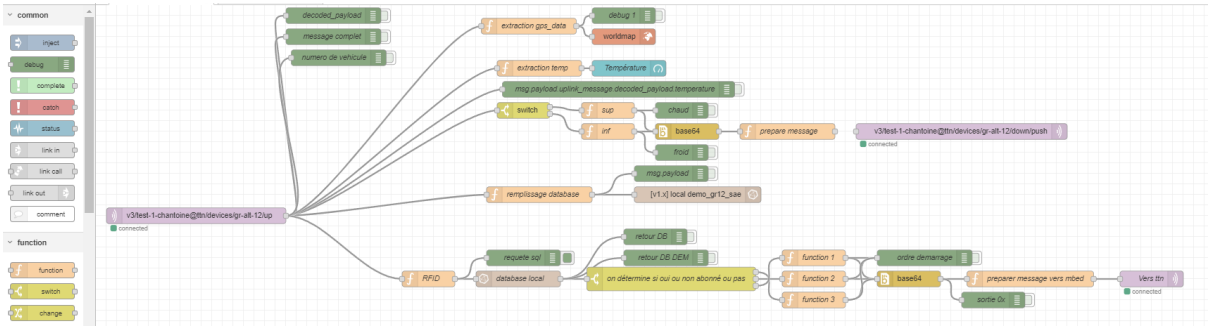


Dans cette capture nous entrons en détail dans une fonction dans laquelle on va extraire la température et l'humidité, puis les stocker dans des variables que nous souhaitons stocker dans une base de données.
La fonction se nomme "remplissage database", et nous voyons comment le bloc est utilisé par la suite, activant le bloc INFLUXDB Out, où sera envoyé msg.data créé juste avant grâce à la fonction.





Cette partie permet d'utiliser le module RFID. Si un badge est détecté il va chercher l'identification dans la base de donnée, puis détermine s'il est associé à un compte abonné, résilié ou non existant. Ensuite la réponse est envoyée à la MBED qui allumera une lumière correspondante à chaque situation : badge connu et autorisé (led verte), badge connu mais non-autorisé (led orange), et badge inconnu (led rouge), annonçant aussi l'absence de badge.



Ici nous voyons l'ensemble du programme Node Red, comportant chaque extrait que nous avons détaillé plus haut.
En résumé, grâce à Node Red, nous traitons les données les données reçues par TTN et les exploitons :
Nous affichons la position GPS du device, nous affichons la température ambiante, tout en remplissant une base de données avec ces données de température et d'humidité, nous sommes capables de renvoyer une information au device MBED en fonction de quel badge est lu, après avoir interrogé la base de données.

PARTIE 2 : Le device MBED

Nous avons eu accès à un code Mbed dans lequel nous avions déjà des fonctionnalités de prêtes, comme la communication via l'antenne, et la création du Payload qui sera envoyé vers TTN par la suite.

2.1 Capteur température

```
                    // Read Sensor temp and humidity values
1 ──────────►       fTemp = myTH02.ReadTemperature();
                    printf("Temp=%.2f\t",fTemp);
2 ──────────►       fHumid = myTH02.ReadHumidity();
                    printf("Humidity=%.2f\n",fHumid);
                    latitude=(float)mon_gps.gps_gga.latitude;
                    longitude=(float)mon_gps.gps_gga.longitude;
                    altitude=(float)mon_gps.gps_gga.msl_altitude;
                    printf("Latitude : %.2f ",latitude);
                    printf("Longitude : %.2f ",longitude);
                    printf("Altitude : %.2f ",altitude);


                    RfChip.PCD_Init(); //faut initialiser le truc passque c'est mieux


                    Payload.reset();
3 ──────────►       Payload.addTemperature(1,(float) fTemp);
4 ──────────►       Payload.addRelativeHumidity(2,fHumid);
                    Payload.addGPS(3,latitude,longitude,altitude);
```

Sur les flèches 1 et 2 nous attribuons des valeurs aux variables fTemp et dHumid grâce aux méthodes .ReadTemperature() et .ReadHumidity() qui sont des méthodes venant de la bibliothèque du capteur même.
Les lignes 3 et 4, nous ajoutons au Payload grâce aux méthodes fournies (.addTemperature & .addRelativeHumidity) les valeurs que nous avons mesurées. Respectivement, dans le payload, on aura dans le champ 1 la température et dans le champ 2 l'humidité.

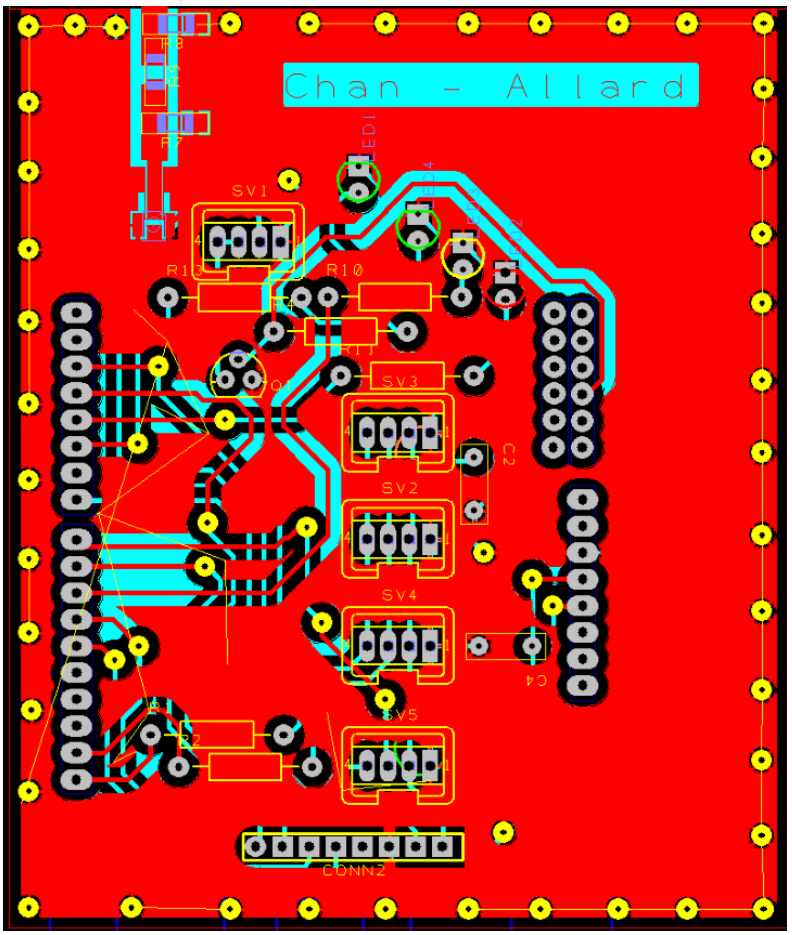
2.2 GPS

La méthode .update() (1) est spécifiée dans la bibliothèque du composant, elle est nécessaire avant d'utiliser ses données.
De la même manière qu'auparavant, on attribut les données dans des variables (2, 3 et 4), et on les ajoute au Payload (5).

2.3 Lecteur de badges

Nous devons d'abord initialiser le composant.
Ensuite nous testons la présence d'un badge, s'il y en a un nous envoyons donc le nom lu, sinon nous envoyons un nom tout à 0 .

PARTIE 3 : CARTE MEZZANINE



PCB de la carte sur DesignSpark

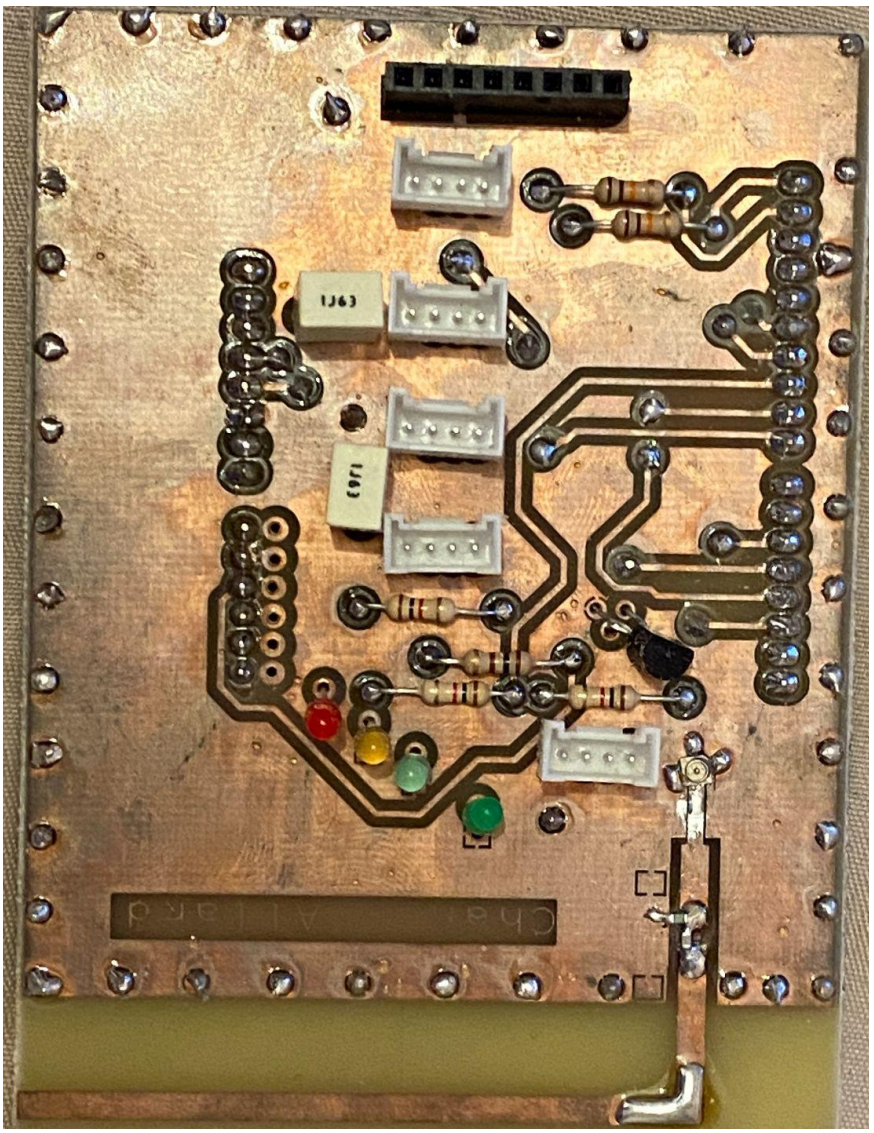
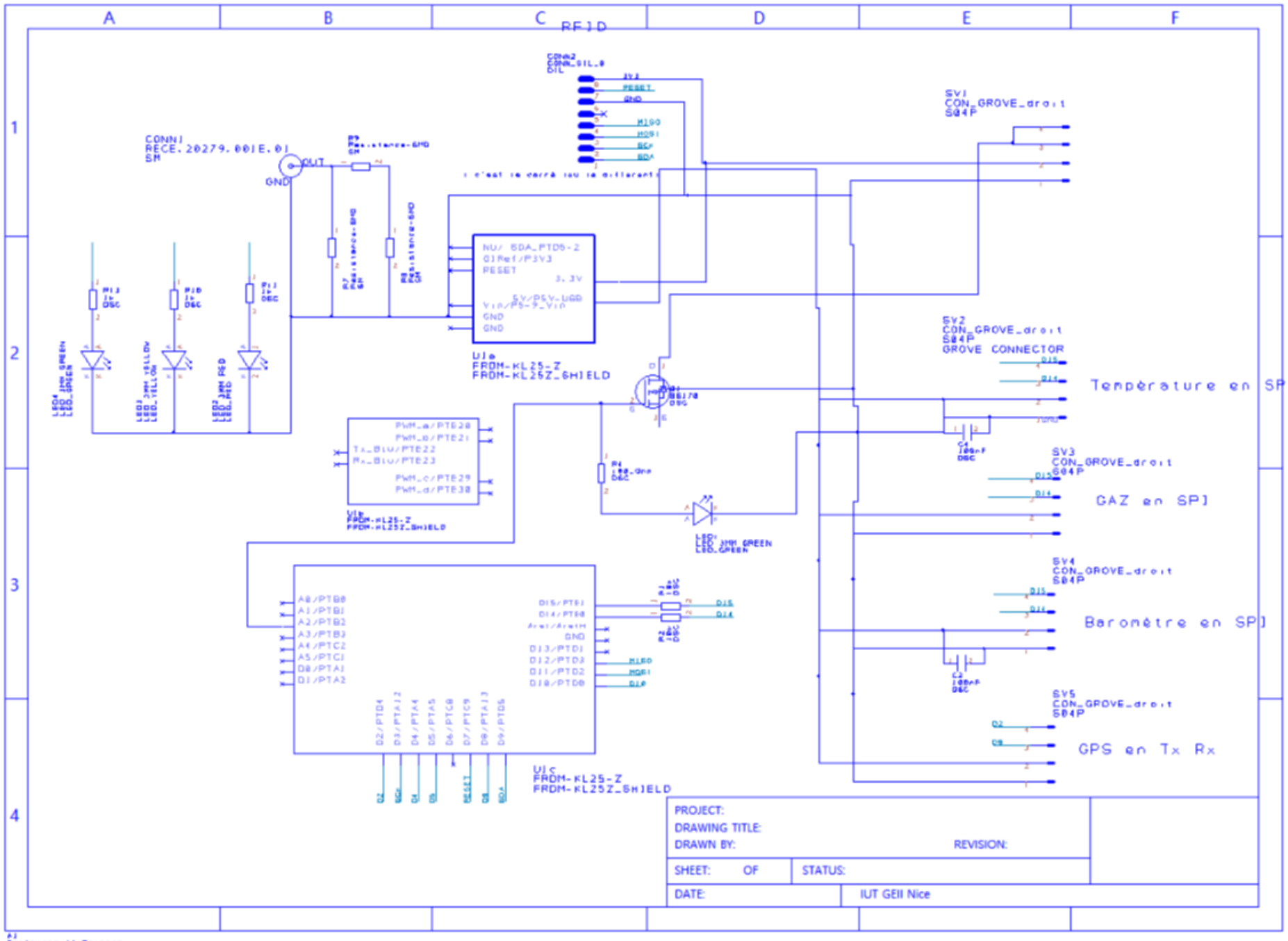


Photo de la carte mezzanine finale

PARTIE 3: ADS

Mettre ça en annexe.

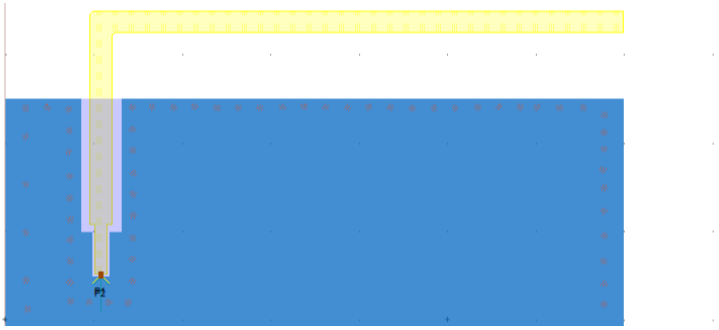

A
B
C
D
E
F
1
2
3
4
RFID
CONN1
FECE.20279.001E.01
SM
OUT
GND
U1a
FRDM-KL25-Z
FRDM-KL25Z_SHIELD
U1b
FRDM-KL25-Z
FRDM-KL25Z_SHIELD
U1c
FRDM-KL25-Z
FRDM-KL25Z_SHIELD
SV1
CON_GROVE_droit
S04P
SV2
CON_GROVE_droit
S04P
GROVE CONNECTOR
Température en SPI
SV3
CON_GROVE_droit
S04P
CAZ en SPI
SV4
CON_GROVE_droit
S04P
Baromètre en SPI
SV5
CON_GROVE_droit
S04P
GPS en Tx Rx
LED1
LED_GREEN
LED2
LED_YELLOW
LED3
LED_RED
LED4
LED_GREEN
PROJECT:
DRAWING TITLE:
DRAWN BY:
REVISION:
SHEET: OF
STATUS:
DATE:
IUT GEII Nice
A3
Cartouche A4 Paysage

PARTIE 4 : ADS

Définitions
IOT : L'Internet des objets (IoT) désigne les appareils connectés capables de communiquer entre eux et avec des utilisateurs via internet.

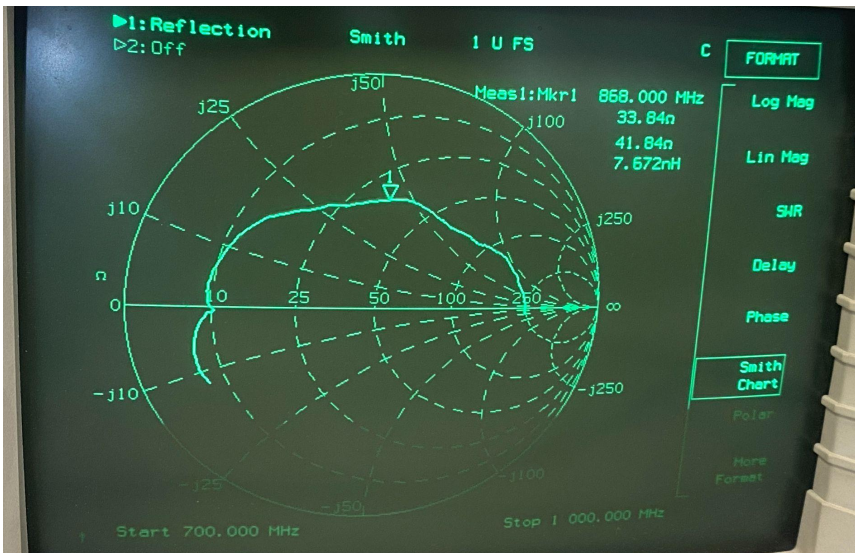
Dans le cadre du projet nous allons fabriquer une antenne pour IoT de type monopole imprimé sur du FR4 d'épaisseur 1,6mm et travaillant à la fréquence de 868MHz. Afin que l'antenne soit capable de recevoir la puissance du récepteur et puisse la transmettre correctement, il faut qu'elle soit adaptée. C'est-à-dire que la source d'impédance soit à 50 Ohms.

Le design se fait sur Advanced Design System.
Voici notre antenne sur ads.



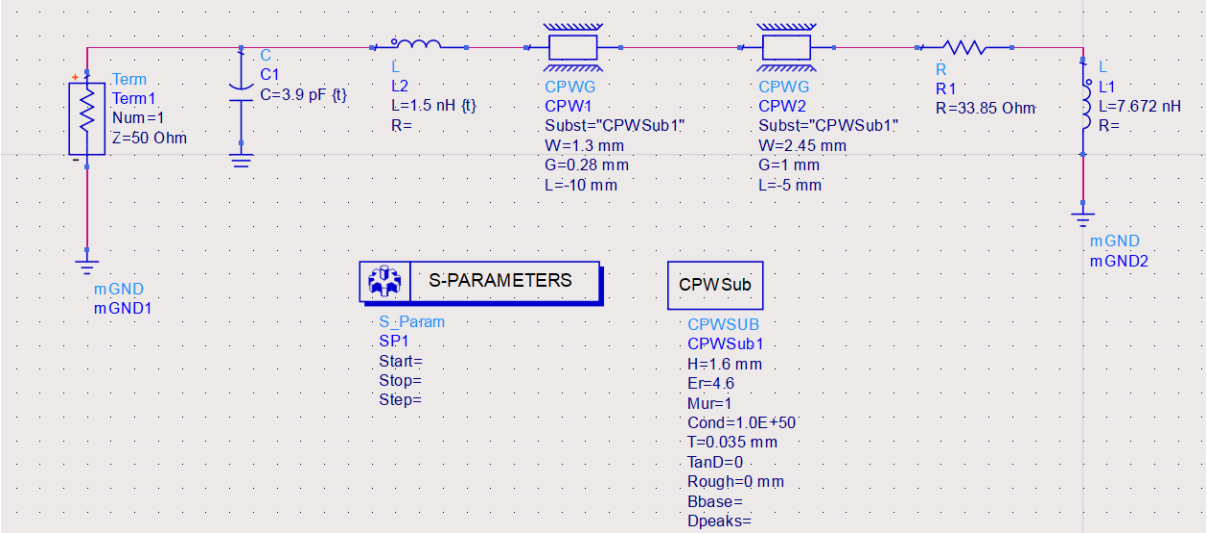
Antenne sous ADS

Notre antenne imprimée sur la carte, les composants soudés et le connecteur UFL fonctionnel, nous avons mesuré l'impédance de l'antenne sur l'abaque de Smith afin de l'adapter ensuite.



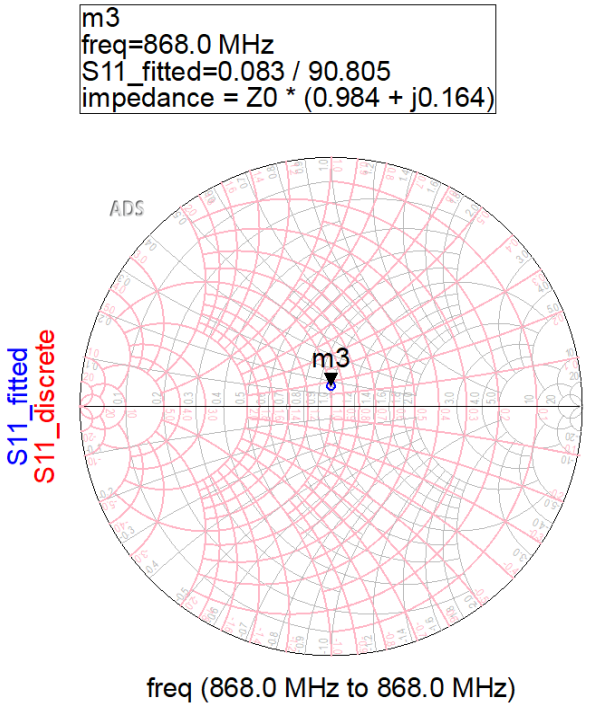
Abaque de Smith réelle.

Voici l'adaptation sur ADS



Schématique de l'antenne adaptée sur ADS

Puis nous avons simulé le circuit d'adaptation.



Abaque de Smith sur ADS de l'antenne.

Les composants CMS à souder sur la carte sont un condensateur de 3.9nF et une inductance de 1.5nH.

CONCLUSION

Nous avons donc dans ce projet, créé un réseau composé d'un device connecté, du serveur, d'une base de données.
Nous sommes capable de lire des données venant du device connecté, d'exploiter ces données à travers un traitement via Node Red et à travers d'une base de données afin d'avoir un suivi en temps réel de plusieurs informations.
Nous pouvons aussi agir sur le device en fonction de diverses conditions : nous pouvons déverrouiller ou non le vélo (projet final) en fonction du badge lu et de son autorisation, et en fonction également de l'état du véhicule (en service ou non).