Surveillance d'Environnement pour Entrepôts

Projet BTS SN-IR

Alphonse Benoit - L'Isle sur la Sorgue - 2021

Dossier V2.0

Revue 3

Table des matières

Table des matières

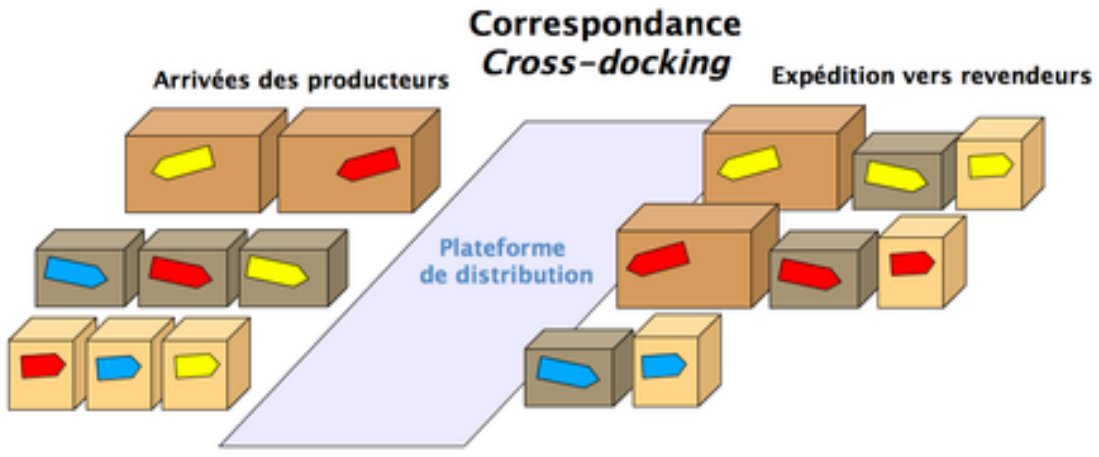
Présentation générale du projet S2E

➢ Présentation:

Nous devons assurer la traçabilité des bonnes conditions d'entreposage dans l'entrepôt de la société CrossDock, une entreprise de logistique dont l'activité consiste à préparer puis à expédier des commandes constituées de produits cosmétiques et/ou de parapharmacie.

Les produits arrivent des fournisseurs puis sont entreposés provisoirement, en attente de leur reconditionnement au cours de la préparation des commandes. Les colis préparés sont ensuite aiguillés vers des postes regroupant les commandes propres à un revendeur (client final de CrossDock) puis sont enfin expédiés.

Notre projet consiste donc à installer un système d'affichage composé de capteurs en indiquant différentes informations telles que: la température, l'humidité, la détection d'incendie et la batterie. Les mesures doivent aussi être instantanées.



<<Principe du « Cross-docking » [Source : Wikipédia]>>

Présentation générale du projet S2E

➢ Analyse de l'existant:

Il y a 2 ans, une solution de supervision de température et d'hygrométrie a été proposée pour remplacer à terme celle déjà en place chez CrossDock.

La solution consiste à placer des clé USB faisant thermometre et hygrometre, permettant d'enregistrer des données, le problème est que ces clés ont un espace de stockage limité à 1 semaine.

Ceci implique qu'une personne doit chaque semaine :

- Récupérer les 4 « data-logger » répartis aux 4 coins de l'unique entrepôt actuellement utilisé.
- Transférer les mesures effectuées dans le système d'information de l'entreprise, en veillant à respecter l'affectation de chaque « data-logger » à la zone de l'entrepôt à laquelle il est associé.
- Effacer les données de chaque « data-logger ».
- Remettre en place les « data-logger » dans l'entrepôt.

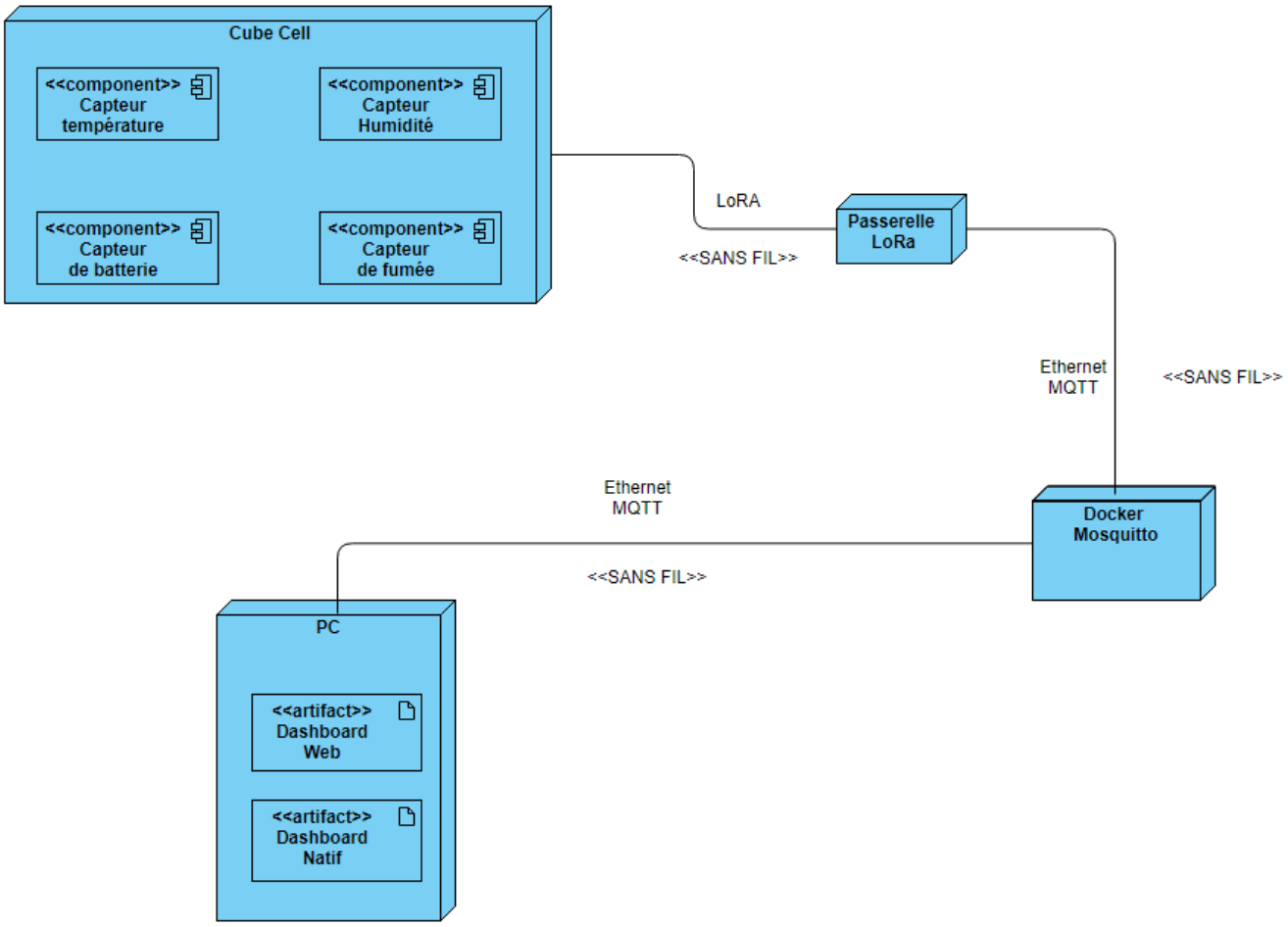
Le problème de cette solution est que les données ne sont consultables qu'une semaine après.

C'est pour cela que notre solution actuelle est meilleure, elle nous permet de mesurer la température et l'humidité ainsi qu'un capteur de fumée. Toutes ces données seront consultables sur le moment grâce à un point d'affichage, disposée dans l'entrepôt ce qui permettra de consulter les données de tous les capteurs instantanément.

Ces capteur sont situés sur une carte arduino alimentée par batterie, dont l'alimentation est affichée sur le point d'affichage. Le transfert de données se fera via une carte CubeCell Dev-board vers une passerelle LoRa qui sera ensuite envoyée vers un docker où les données seront consultables.

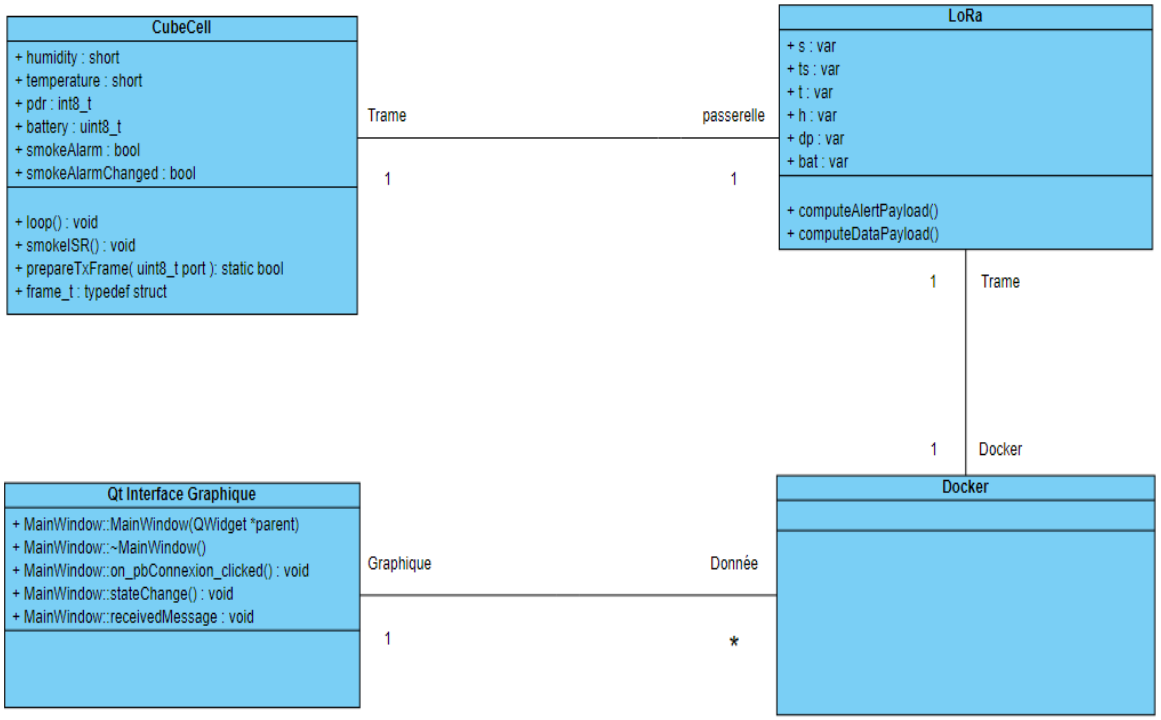
Présentation générale du projet S2E

➢ Diagramme de déploiement:



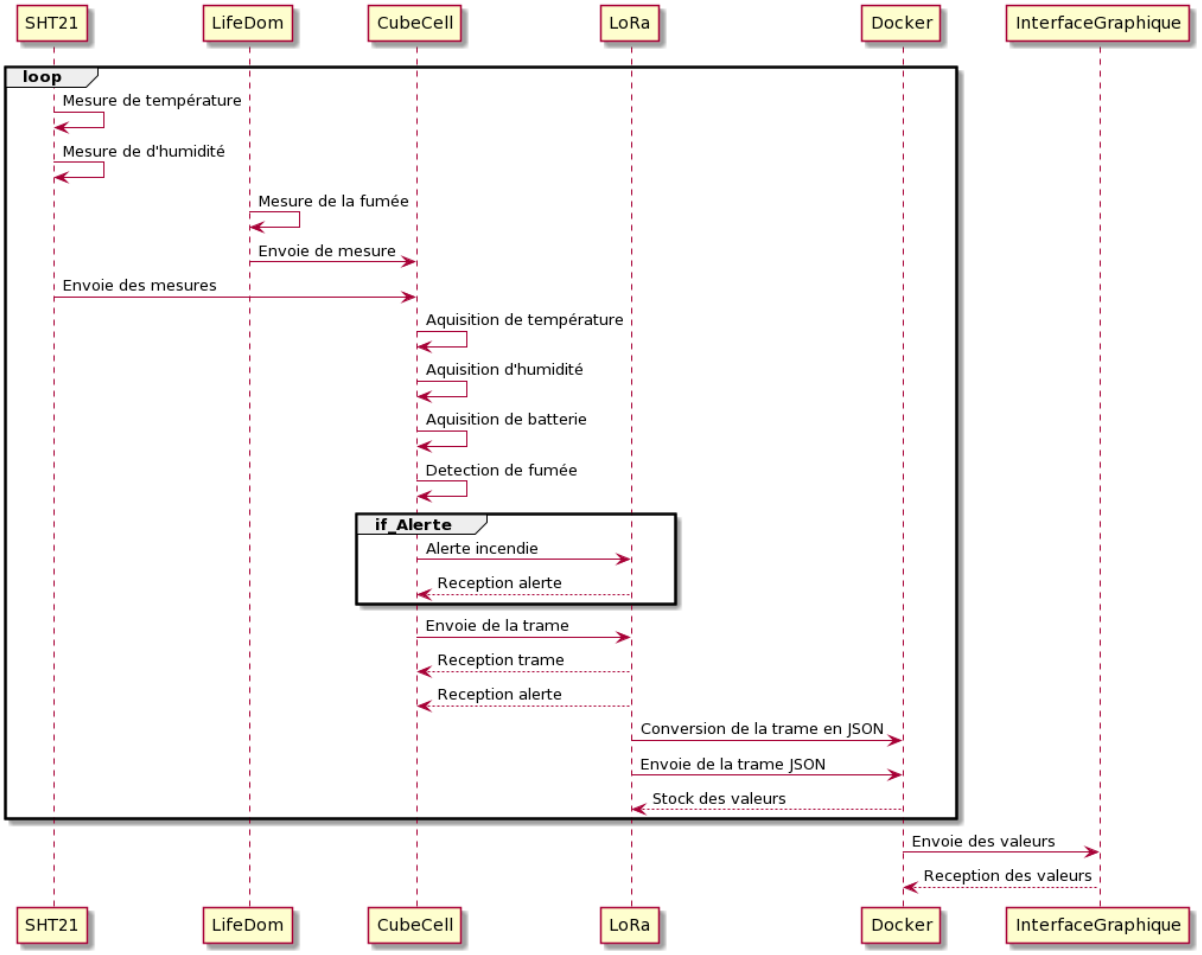
Présentation générale du projet S2E

➢ Diagramme de classe



Présentation générale du projet S2E

➢ Diagramme de séquence:



Présentation générale du projet S2E

➢ Planification:

Projet S2E	250 h?	Ven 08/01/2	Sam 22/05/2	
169	169 h	Ven 08/01/2	Mar 06/04/2	
Mise en place de la passerelle	25 h	Ven 08/01/21	Mar 19/01/21	
Définir la messagerie LoRa	12 h	Mar 19/01/21	Ven 22/01/21	3
Codage Temperature et Humidité	42 h	Ven 22/01/21	Jeu 11/02/21	4
Codage fumée	24 h	Jeu 11/02/21	Mar 09/03/2	5
Codage batterie	12 h	Mar 09/03/21	Ven 12/03/21	6
Transmition LoRa -> passerelle	20 h	Ven 12/03/21	Mar 23/03/21	7
Reception LoRa	24 h	Mar 23/03/2	Jeu 01/04/21	8
Rédaction manuel tuto	10 h	Jeu 01/04/21	Mar 06/04/21	9
IR 2	186 h	Ven 08/01/2	Mar 13/04/2	
Mise en place de la	25 h	Ven 08/01/21	Mar 19/01/21	
Définir la messagerie LoRa	20 h	Mar 19/01/21	Mer 27/01/21	13
MQTT - installation Mosquitto	12 h	Mer 27/01/21	Mar 02/02/21	14
MQTT - Tester la	15 h	Mar 02/02/21	Mar 09/02/21	15
MQTT - définir les topics	24 h	Mar 09/02/21	Jeu 18/02/21	16
Concevoir dashbord - Nod Red	20 h	Jeu 18/02/21	Ven 12/03/21	17
Concevoir dashbord -QT	40 h	Ven 12/03/21	Mer 31/03/21	18
Rédaction manuel tuto	30 h	Mer 31/03/21	Mar 13/04/21	19

Présentation générale du projet S2E

➢ Planification:

		Mode Tâche	Nom de la tâche	Début	Fin
0			**Projet S2E**	**Ven 08/01/21**	**Mer 07/04/21**
1			**Tests avec carte Arduino**	**Ven 08/01/21**	**Mer 17/02/21**
2			Prise de connaissance du projet et analyse du SHT21	Ven 08/01/21	Mer 13/01/21
4			Teste et selection du capteur de fumée	Jeu 14/01/21	Mer 20/01/21
5			Analyse électrique du capteur de fumée	Jeu 21/01/21	Ven 29/01/21
6			Assemblage de tout les éléments et prototypage rapide	Ven 29/01/21	Mer 17/02/21
7			**Création de la carte avec la cubecell**	**Mer 17/02/21**	**Mer 07/04/21**
8			Création du schéma structurel & liste de	Mer 17/02/21	Mer 03/03/21
9			Routage de la carte	Jeu 04/03/21	Ven 19/03/21
10			Prototypage rapide	Mer 24/03/21	Mer 31/03/21

Partie IR1:

Muel Mélina

Partie IR 1 : Muel Mélina

➢ Objectifs:

Durant ce projet, j'ai comme objectifs de mettre en place un programme permettant de faire les différentes mesures demandées, via divers capteurs, puis de les transférer via une **carte CubeCell vers une passerelle LoRa.**

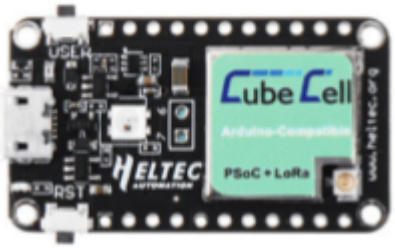
Voici les différents objectifs avec leurs outils pour les mettre en place:

Objectifs: Mesure
- Température
- Humidité

Outil: Sensirion SHT21

Objectifs: Mesure
- Présence de fumée .

Outil: Détecteur de fumée - LIFEDOM

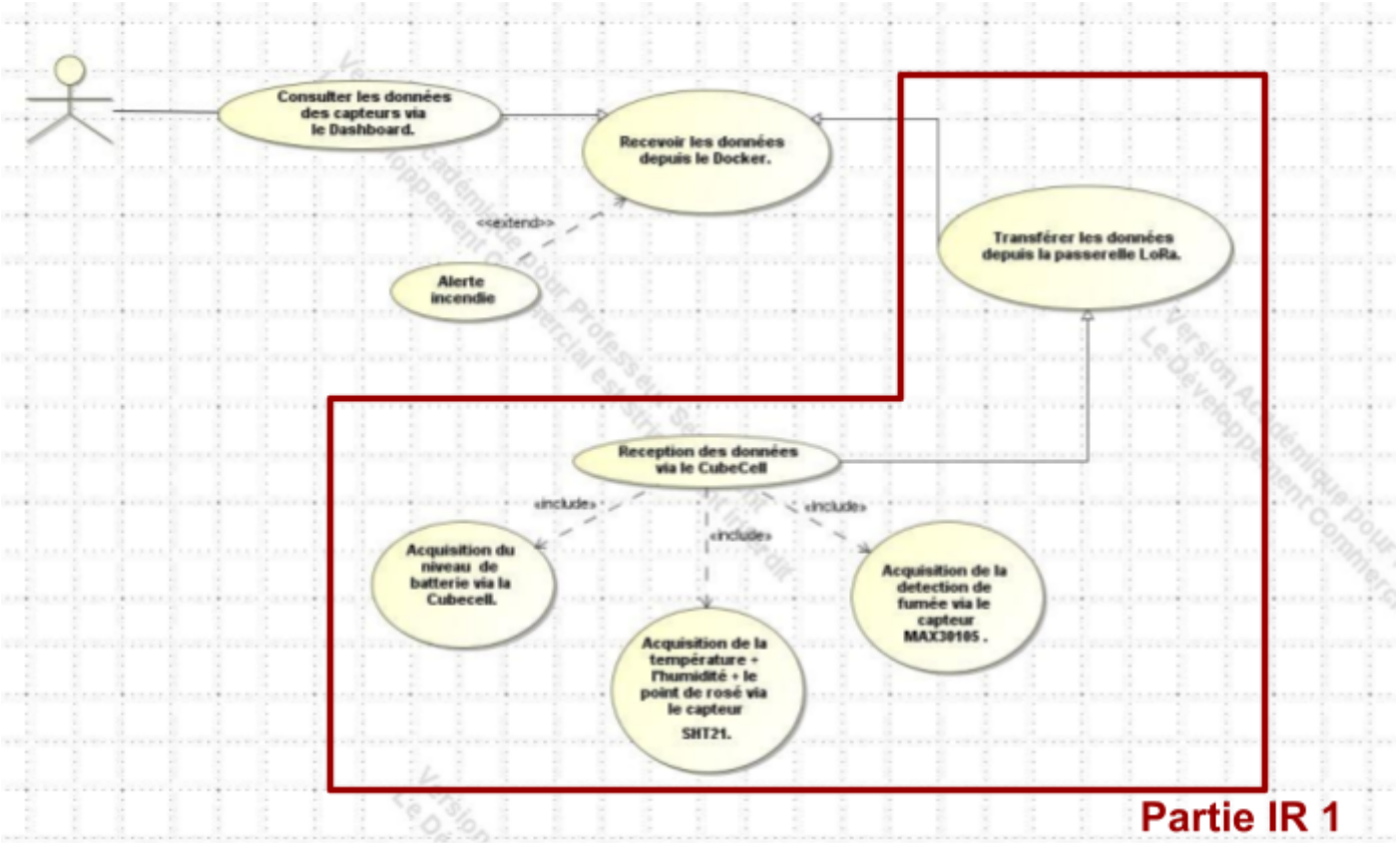


Objectifs: Transmission vers LoRA
- Envoyer les acquisitions vers la passerelle LoRa.
- Envoyer la batterie de la carte.
- Acquisitions et calculs.

Outil: Heltec Cubecell - Dev-Board

Partie IR 1 : Muel Mélina

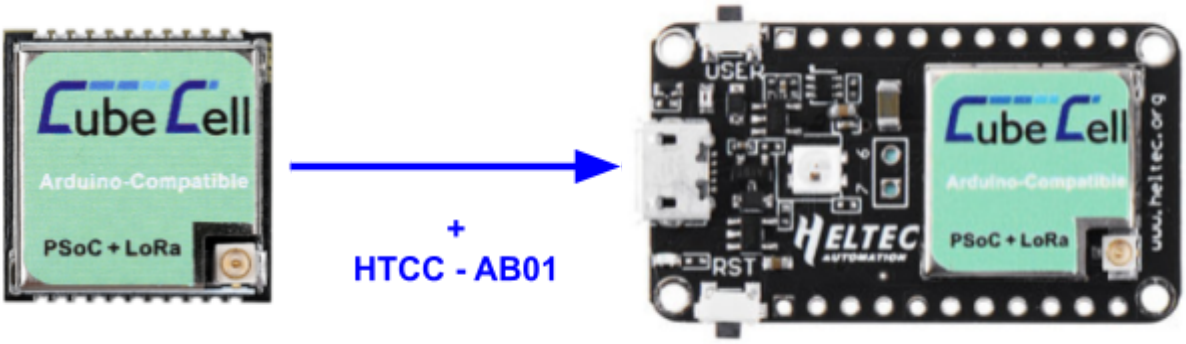
➢ Diagramme cas d'utilisation:



Comme on peut le voir sur le diagramme de cas d'utilisation, en tant qu'élève en Informatique et Réseau, je m'occuperais bel et bien sur le projet de la partie codage de l'acquisition des différentes données comme: la température, l'humidité, le point de rosée, la détection de fumée et enfin du niveau de batterie de l'appareil. Celles-ci seront alors transférées via la CubeCell tous les *X* temps vers la passerelle LoRa dont ensuite mon collègue, l'IR 2, s'occupera du traitement de ces données.

Partie IR 1 : Muel Mélina

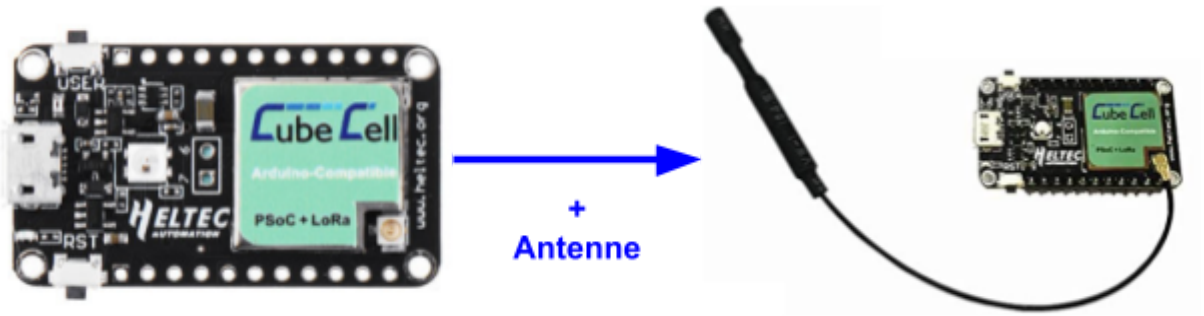
➢ HelTec - CubeCell Dev-Board:



La **CubeCell Dev-Board**, par la compagnie HelTec, est un outil principalement mis en place pour les applications de type **LoRa / LoRaWAN**. Elle est composée d'un **module CubeCell** permettant l'utilisation du **LoRa**, mais aussi d'une **carte de développement HTCC-AB01** qui permet de simplifier les communications entre les outils externes et la CubeCell.

Étant spécialisée dans le **LoRa/LoRaWAN**, elle dispose d'une **librairie** de code préfète fait spécialement à cet effet. Elle a aussi d'autres codes dans sa librairie qui n'ont pas de lien direct avec le LoRa.

C'est en effet grâce à celui-ci que nous pouvons réaliser la communication et le transfert des données vers la passerelle **LoRa**. Cela doit cependant être fait avec une **antenne connectée au module**, afin de pouvoir réaliser la **connexion entre celui-ci et la passerelle LoRa.**



Partie IR 1 : Muel Mélina

➢ Arduino IDE:

La **CubeCell Dev-Board** est totalement capable d'être codée dans le langage particulier qu'est **Arduino**.

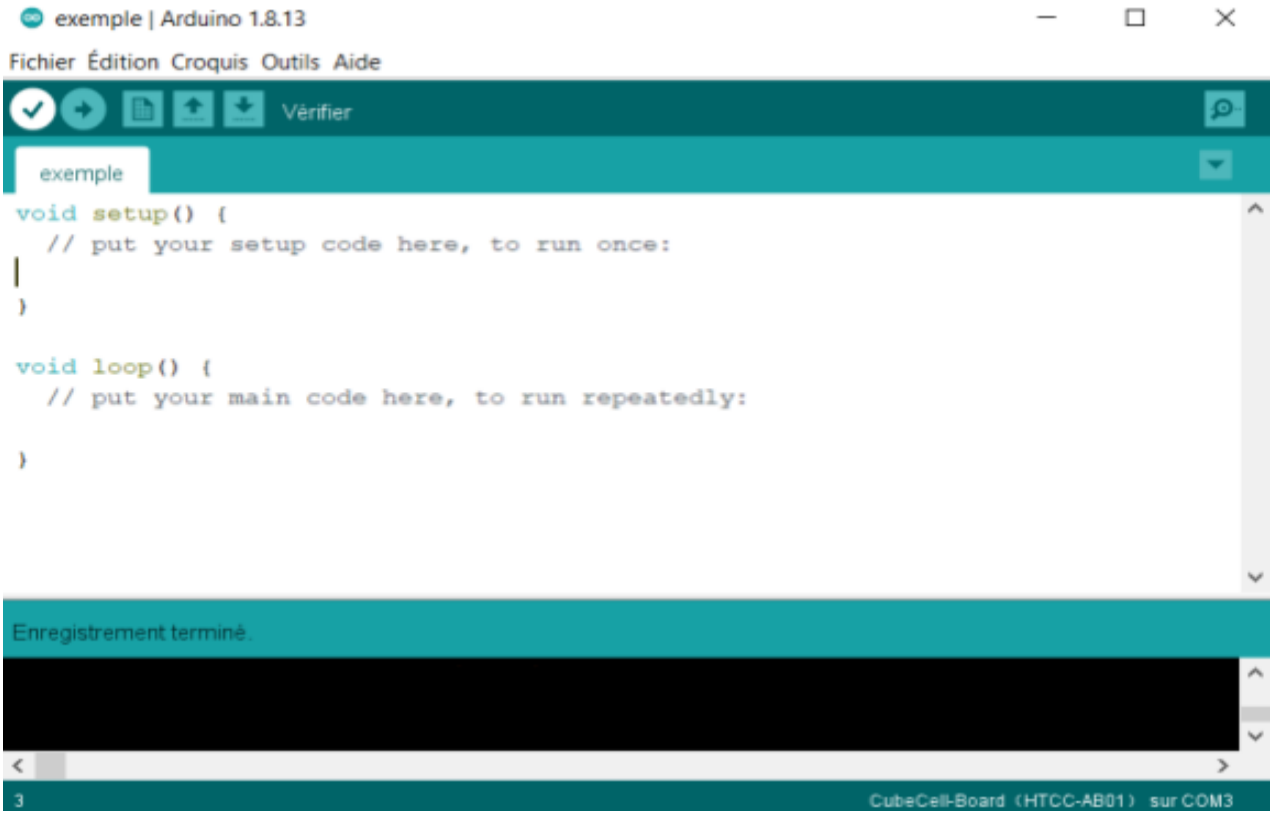
C'est ainsi que j'ai utilisé le logiciel **Arduino IDE** pour coder sur la carte.

Par ailleurs, on peut y trouver depuis le logiciel divers codes dans la **librairie** dédié à la **CubeCell Dev-Board** nommée:

"CubeCell Dev-boards"

Celui-ci permet justement la **communication** entre la CubeCell et la passerelle. Elle continue d'être **mise à jour** afin d'y **ajouter** divers programmes **et/ou améliorer** les anciens programmes.

L'interface de **Arduino IDE** se présente ainsi:



Partie IR 1 : Muel Mélina

➢ Transfert de données:

J'ai comme objectif de **transférer** les diverses **mesures** obtenues par **les capteurs** ou même la **CubeCell** elle-même **vers la passerelle LoRa.**

Pour cela, nous avons une **taille de trame de données limitées**, c'est pour cela que nous nous devons de faire une **trame optimisée** pouvant contenir toutes les informations possible **dans une et même trame.**

Afin de parvenir à réaliser une trame nommée "payload" rentrant dans cette limite, nous avons dû chercher des **types de variables** correspondant à nos variables, qui seront **attribuées** aux différentes **acquisitions**. De plus, les types se doivent d'être assez **"légers"** pour que le **transfert soit possible**. Nous avons donc recherché les meilleures solutions possibles.

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$3.4*10^{-38}$ à $3.4*10^{38}$
double	Flottant double	8	$1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant double long	10	$3.4*10^{-4932}$ à $3.4*10^{4932}$

Partie IR 1 : Muel Mélina

➢ Transfert de données:

Specifier	Signing	Bits	Bytes	Minimum Value	Maximum Value
`int8_t`	Signed	8	1	-2^7 which equals −128	$2^7 - 1$ which is equal to 127
`uint8_t`	Unsigned	8	1	0	$2^8 - 1$ which equals 255
`int16_t`	Signed	16	2	-2^{15} which equals −32,768	$2^{15} - 1$ which equals 32,767
`uint16_t`	Unsigned	16	2	0	$2^{16} - 1$ which equals 65,535
`int32_t`	Signed	32	4	-2^{31} which equals −2,147,483,648	$2^{31} - 1$ which equals 2,147,483,647
`uint32_t`	Unsigned	32	4	0	$2^{32} - 1$ which equals 4,294,967,295
`int64_t`	Signed	64	8	-2^{63} which equals −9,223,372,036,854,775,808	$2^{63} - 1$ which equals 9,223,372,036,854,775,807
`uint64_t`	Unsigned	64	8	0	$2^{64} - 1$ which equals 18,446,744,073,709,551,615

Nous avons donc décidé, avec l'IR2, de faire une **trame contenant ces types** ci-dessus. Pour cela, nou sommes passé sur différentes versions:

➢ **La V1:** La structure de base avec chaques données demandées, mais cette solution n'était **pas très optimisée**.

En effet, dans ce code nous avons défini une **structure** dont le marqueur d'utilisation sera **"trame"**, où l'on pourra créer une structure stockant les données de température, humidité, la batterie, le point de rosée et enfin la fumée..

Cependant elle n'était plus d'actualité car nous voulons avoir 2 trames **différentes et indépendantes** entre la fumée et le reste des valeurs.

```
typedef struct {          // Champs de bits contenu dans la trame

  short temperature;      // Température pouvant aller de -327 C° à 327 C°
  short humidity;         // Humidité pouvant aller de -327%RH à 327%RH
  uint8_t point de rosé;  // Point de rosée pouvant aller de -128 à 127
  bool smoke :            // Détection de fumée Oui = 1 / Non = 0
  uint8_t battery;        // Niveau batterie allant de 0% à 255%

} trame;
```

Partie IR 1 : Muel Mélina

➢ Transfert de données:

➢ **La V2:** Une structure similaire mais **plus optimisée** pour notre projet.

En effet, dans ce code nous avons défini une **structure** dont le marqueur d'utilisation sera **"frame_t"**, où l'on pourra créer une structure stockant les données de température, humidité, point de rosé et enfin de batterie.

Ici nous avons appelé notre structure **"s2ePayload"**.

De plus, la variable de la fumée est en dehors de la structure car elles sont **indépendantes l'une de l'autre** et nous visons à envoyer la trame de la fumée sur un port dédié donc différent des données de conditions de l'entrepôt.

```
typedef struct {
  short temperature;      // température exprimée en centièmes de degré °C
  short humidity;         // humidité exprimée en "pour dix mille" (°/ooo)
  int8_t pdr              // Point de rosé
  // identifiant du module Heltec => présent implicitement dans la trame LoRa
  uint8_t battery;        // niveau batterie
} frame_t;

frame_t s2ePayload;
bool smoke;               // état détecteur fumée sur 1 bit
```

Ensuite, en ce qui concerne le programme de la transmission de trame, j'ai utilisé l'exemple donné dans la **librairie *"CubeCell Dev-boards"***, nommé "LoraWan". J'y ai juste modifié le ***"DevEui"*** qui représente en quelque sorte l'identifiant de celui qui envoie la trame.

Partie IR 1 : Muel Mélina

➢ Transfert de données:

Maintenant que nous avons créé cette structure adéquate pour notre projet, nous pouvons passer à **la création du payload**.

Pour cela, nous reprenons la méthode de base inclus dans le code "***CubeCell Dev-boards"***, où se trouve la méthode:

```
/* Application port */
uint8_t appPort = 2;

/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
   appDataSize = 4;
   appData[0] = 0x00;
   appData[1] = 0x01;
   appData[2] = 0x02;
   appData[3] = 0x03;
}

[...]

/* Envoi du payload*/
case DEVICE_STATE_SEND:
            {
                  prepareTxFrame( appPort );
                  LoRaWAN.send();
                  deviceState = DEVICE_STATE_CYCLE;
                  break;
            }
```

Dans cette méthode de base nommée **"prepareTxFrame"**, qui revient pour nous à notre **payload**, se trouve un tableau.

Tout d'abord, il **crée un tableau** de X colonne(s) avec **"appDataSize"** (ici X = 4). Ensuite il **définit la valeur prédéfinie** dans chacune de ses colonnes via le **"appData[X]"**. De plus, ses données seront envoyées sur le port **"appPort"** qui a été défini au préalable.

Pour réaliser **l'envoi du payload**, il suffit alors de mettre dans le case dédié à cet effet la ligne **"prepareTxFrame"** avec le port où envoyer le payload.

Partie IR 1 : Muel Mélina

➢ Transfert de données:

Ensuite nous reprenons cette méthode et mais en l'adaptant à notre façon:

```
/* Ports LoRa sur lesquels seront envoyées les données */
#define PERIODIC_DATA_PORT  1       // T°/%HR/Niveau de charge batterie
#define ASYNC_DATA_PORT  2          // Fumée

/* Application port */
uint8_t appPort = PERIODIC_DATA_PORT;

/* Prepares the payload of the frame */
static bool prepareTxFrame( uint8_t port )
{
  appPort = port;
  switch (port) {

    case ASYNC_DATA_PORT:       // Réveil par interruption
      Serial.println("Envoi début/fin alerte incendie");
      appDataSize = 1;
      appData[0] = smokeAlarm ? 0xff : 0x00;
      break;


    case PERIODIC_DATA_PORT:    // Réveil périodique
      uint8_t *pDataSrc = reinterpret_cast<uint8_t *>(&s2ePayload);
      uint8_t *pDataDst = appData;
      appDataSize = sizeof(s2ePayload);    // taille max. payload : 64
      for (int i = 0; i < appDataSize; i++) {
        *pDataDst++ = *pDataSrc++;
      }
      break;
  }
  return true;
}
```

Dans un premier temps, nous avons défini deux ports différents:

- **PERIODIC_DATA_PORT:** Port pour toutes les données de la structure **"s2ePayload"**.
- **ASYNC_DATA_PORT:** Port pour l'alerte incendie, donc la fumée.

Ensuite dans la méthode **"prepareTxFrame"**, nous préparons les 2 cas possibles pour chaque port:

- **PERIODIC_DATA_PORT:** Si le port demandé est celui de **"s2ePayload"**, la structure sera transformée en une **variable uint8_t** puis calculera automatiquement le nombre de colonne à obtenir. on aura donc un **uint8_t s2ePayload** avec un tableau de 4 colonnes donc un **"appData[3]"** avec respectivement une valeur par colonne.
- **ASYNC_PORT:** Si le port demandé est celui de l'alerte incendie, seulement un tableau de 1 sera créé dédié à la fumée.

Partie IR 1 : Muel Mélina

➢ Transfert de données:

Enfin, l'appel de la méthode **"prepareTxFrame"** se présente donc comme ceci dans la partie de l'envoi du payload du code:

```
[...]

/* Envoi du payload*/
case DEVICE_STATE_SEND:
        {
            prepareTxFrame( PERIODIC_DATA_PORT );
            LoRaWAN.send();
            deviceState = DEVICE_STATE_CYCLE;
            break;
        }
```

Effectivement, cela ne change pas beaucoup du code de base étant donné qu'il s'agit du même principe.

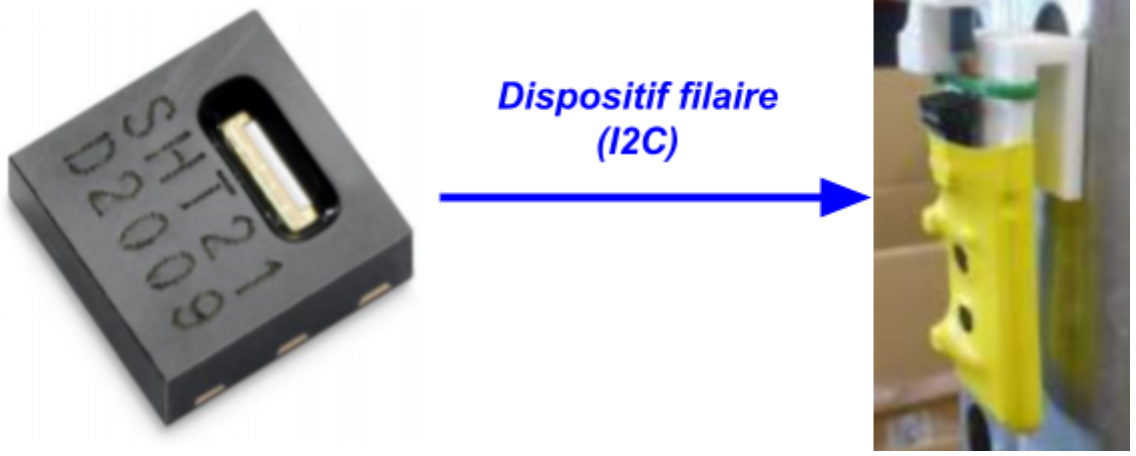
En effet, s'ils ont souhaite envoyer une trame qui concerne les conditions de l'entrepôt, il suffira de mettre: **prepareTxFrame(PERIODIC_DATA_PORT)**

De plus, s'il on souhaite envoyer une trame en rapport avec l'alerte incendie, il faudra mettre: **prepareTxFrame(ASYNC_DATA_PORT)**

Enfin, le but sera de faire en sorte que l'envoie du **payload** s'envoie de façon **périodique** pour les valeurs demandées, tandis que l'**alerte incendie** se fasse de façon **événementielle**.

Partie IR 1 : Muel Mélina

➢ Sensirion SHT21:

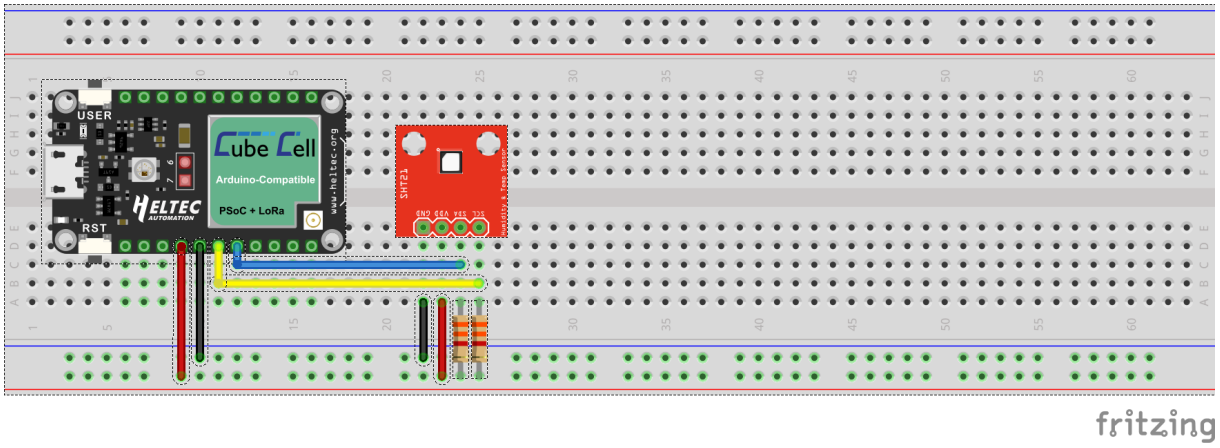


Le **Sensirion SHT21** est un capteur permettant de faire différentes **mesures** pour notre projet:

- La **température** (C° ou F°): Connaître la température en degré de la salle où se trouve les divers produits dans le cross-dock.
- L'**humidité** (%RH): Connaître l'humidité de la salle où se trouve les divers produits dans le cross-dock.

Etant donné que nous utilisons le **filaire entre la CubeCell Dev-Board**, et non le WiFi, nous devons utiliser un **bus I2C**. Il s'agit d'un bus permettant la **communication** entre les équipements avec un principe de maître/esclave. Le maître (CubeCell) commence un **échange** avec son esclave (SHT21 ici) pour qu'il donne une **trame de donnée** (SDA), créant à la même occasion **une horloge de synchronisation** (SCL) afin de simplifier la **"traduction" de l'échange**.

En ce qui concerne son **branchement** sur la carte, voici le schéma:



Partie IR 1 : Muel Mélina

➢ Acquisitions Température / Humidité:

Afin d'**obtenir la température et l'humidité** d'une pièce, j'ai fait appel à la fameuse **librairie** du CubeCell Dev-Board où se trouve en exemple le programme d'acquisition dans **Arduino IDE**.

Il se nomme **"ReadSHT2X"** et le voici:

```
/**********************************************************************
 * ReadSHT2x
 *  An example sketch that reads the sensor and prints the
 *  relative humidity to the PC's serial port
 **********************************************************************/

#include <Wire.h>
#include <SHT2x.h>


void setup()
{
   Serial.begin(115200);
}

void loop()
{
   pinMode(Vext, OUTPUT);
   digitalWrite(Vext, LOW);
   delay(50);

   Wire.begin();                        // Mise en marche I2C
   Serial.print("Humidity(%RH): ");
   Serial.print(SHT2x.GetHumidity());
   Serial.print("     Temperature(C): ");
   Serial.println(SHT2x.GetTemperature());
   Wire.end();                          // Mise en arrêt I2C

   digitalWrite(Vext, HIGH);

   delay(3000);
}
```

Partie IR 1 : Muel Mélina

➢ Acquisitions Température / Humidité:

Je me suis donc inspirée de ce code pour en **remodeler** un code similaire mais qui nous convenait un peu plus, bien que cela ne change en rien le résultat final.

Voici la V1 du **programme remodelé** nommé simplement **“SHT21”**:

```
/*****************************************************************
*                                                                *
*              Capteur Température + Humidité                    *
*                                                                *
*****************************************************************/

#include <Wire.h>
#include <SHT2x.h>


void setup()
{
   Serial.begin(9600);
}

void loop()
{
      digitalWrite(Vext, LOW);
      delay(500);

      Wire.begin();
      Serial.print("\n");

      Serial.print("Humidity (%RH): ");
      short Hum = SHT2x.GetHumidity();
      Serial.print(Hum);
      Serial.print("\n");

      Serial.print("Tempétarure (C): ");
      short Temp = SHT2x.GetTemperature();
      Serial.print(Temp);
      Serial.print("\n");

      Wire.end();

      digitalWrite(Vext, HIGH);
      delay(3000);
}
```

J’ai utilisé délibérément des variables **“Hum”** et **“Temp”** contenant les valeurs obtenues par le capteur **SHT21**, afin de pouvoir **les réutiliser plus tard** dans le programme.

Partie IR 1 : Muel Mélina

➢ Acquisitions Température / Humidité:

Cependant, maintenant que nous avons **retravaillé** la mise en place d'un **payload**, nous avons dû refaire une **nouvelle forme** pour **l'acquisition de la température et humidité**.

Voici la V2 du **programme remodelé** nommé simplement **"SHT21"**:

```
/*****************************************************************
*                                                                *
*              Capteur Température + Humidité                    *
*                                                                *
*****************************************************************/
[...]

case DEVICE_STATE_SEND:
    {
     /* Capteur Température + Humidité */
     digitalWrite(Vext, LOW);
     delay(500);

     Wire.begin();

     s2ePayload.humidity = static_cast<short>(SHT2x.GetHumidity() * 100);
     Serial.print("Humidité : ");
     Serial.println(s2ePayload.humidity);
     s2ePayload.temperature = static_cast<short>(SHT2x.GetTemperature() * 100);
     Serial.print("Température : ");
     Serial.println(s2ePayload.temperature);


     Wire.end();

     digitalWrite(Vext, HIGH);
     delay(3000);
}

[...]
```

Dans cette nouvelle version, nous utilisons la structure "**s2ePayload"**.

En effet, dans le code ci-dessus, nous pouvons voir que chaque valeur est d'abord acquise puis transformée en une valeur de leur type respectif.

Ensuite, elle est stockée dans "**s2ePayload"** dans leurs variable respective, comme on peut le voir avec la structure: **s2ePayload.nomVariable**.

Partie IR 1 : Muel Mélina

➢ Calcul du point de rosé:

Ensuite, avec l'aide de ses deux acquisitions par le SHT21, nous pouvons **calculer le point de rosé**. Il représente la température que doit avoir l'air pour devenir de la vapeur d'eau saturée.

Pour cela, on utilisera de nouveau la **librairie "ReadSHT2X"**. Celle-ci contient le calcul permettant d'obtenir **le point de rosé** (Bien qu'elle ne soit pas montré dans l'exemple "ReadSHT2X"). Ce programme utilise la **formule de Magnus** afin de trouver notre valeur. La voici:

Calcul du "gamma":

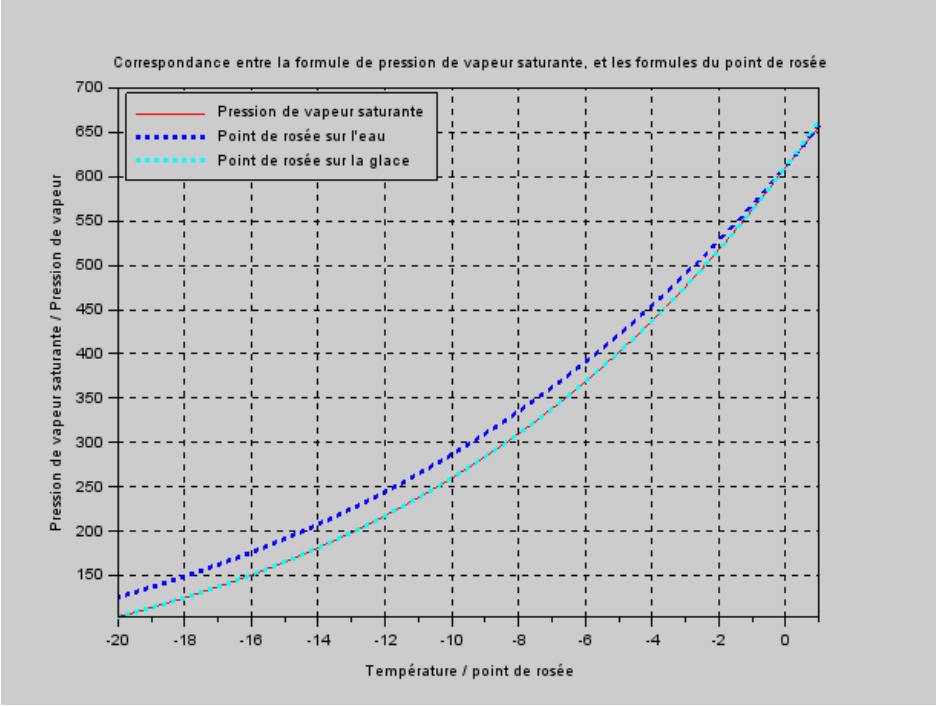
$\gamma(T,RH) = \ln (RH / 100) + Vp*T / (Pr + T)$

Calcul du point de rosé:

$T_{dp} = Pr * \gamma(T,RH) / (Vp - \gamma(T,RH))$

- T = Température
- RH = Humidité
- Vp = Vapeur d'eau
- Pr = Pression barométrique

De plus, un schéma illustrant ce phénomène:



Partie IR 1 : Muel Mélina

➢ Calcul du point de rosé:

De ce fait, nous avons pu rajouter le code du calcul **du point de rosé** dans notre code **"SHT21"**, où se trouvent déjà la température et l'humidité dans celui-ci.

Le voici dans la **V1**:

```
/**********************************************************************
*                                                                     *
*    Capteur Température + Humidité + Point de rosé                   *
*                                                                     *
**********************************************************************/

#include <Wire.h>
#include <SHT2x.h>


void setup()
{
   Serial.begin(9600);
}

void loop()
{
      digitalWrite(Vext, LOW);
      delay(500);

      Wire.begin();
      Serial.print("\n");

      Serial.print("Humidity (%RH): ");
      short Hum = SHT2x.GetHumidity();
      Serial.print(Hum);
      Serial.print("\n");

      Serial.print("Temperature (C): ");
      short Temp = SHT2x.GetTemperature();
      Serial.print(Temp);
      Serial.print("\n");

      Serial.print("Point de rosé: ");
      int8_t Pdr = SHT2x.GetDewPoint();
      Serial.print(Pdr);
      Serial.print("\n");


      Wire.end();

      digitalWrite(Vext, HIGH);
      delay(3000);
}
```

C'est ainsi que nous avons fini le traitement du programme utilisant le **SHT21.**

Partie IR 1 : Muel Mélina

➢ Calcul du point de rosé:

Le voici dans la **V2**:

```
                /*****************************************************************
                *                                                                *
                *   Capteur Température + Humidité + Point de Rosé               *
                *                                                                *
                *****************************************************************/
[...]

case DEVICE_STATE_SEND:
    {
      /* Capteur Température / Humidité + Point de rosé */
      digitalWrite(Vext, LOW);
      delay(500);

      Wire.begin();

      s2ePayload.humidity = static_cast<short>(SHT2x.GetHumidity() * 100);
      Serial.print("Humidité : ");
      Serial.println(s2ePayload.humidity);
      s2ePayload.temperature = static_cast<short>(SHT2x.GetTemperature() * 100);
      Serial.print("Température : ");
      Serial.println(s2ePayload.temperature);
      s2ePayload.pdr = static_cast<int8_t>(SHT2x.GetDewPoint());
      Serial.print("Point de Rosée : ");
      Serial.println(s2ePayload.pdr);

      Wire.end();

      digitalWrite(Vext, HIGH);
      delay(3000);
}

[...]
```

Le code pour le calcul du point de rosé est très similaire à l'acquisition de la température et de l'humidité.

En effet, la valeur du point de rosée est calculée automatiquement puis transformée en **type uint8_t** et enfin stockée dans sa variable dédiée de la structure "**s2ePayload".**

Partie IR 1 : Muel Mélina

➢ Acquisition et détection de fumée:

L'**EC1** a choisi **un détecteur de fumée** adéquat pour cet objectif, nommé LifeDom.

Il nous permettra justement de faire une **acquisition** pour savoir si de la **fumée** (voire **un incendie**) est présente dans la pièce où se trouvera notre carte.

Comme vu précédemment nous mettrons un **type "bool"** pour sa valeur, étant donné que soit il y a de la fumée (**bool Fum = 1**) soit il n'y en a pas (**bool Fum = 0**).

Afin de **simuler la présence de fumée**, nous utiliserons un bouton. Dès qu'il sera actionné, cela représentera une détection de fumée.

Voici un schéma explicatif:

Mon objectif sera que dès qu'une présence de fumée sera **détectée**, une **trame d'alerte** soit transférée à la passerelle LoRa. Celle-ci **transférera** l'alerte sur le **docker** qui sera accessible via un autre **port différent** que celui des autres acquisitions, dû au fait qu'elles sont transférées tous les X temps au docker. Ce qui **n'est pas admissible en cas d'incendie**.

Partie IR 1 : Muel Mélina

➢ Acquisition et détection de fumée:

En ce qui concerne le codage de cette partie, nous allons tout d'abord créer un code permettant d'**afficher l'état d'un bouton poussoir** (= Simule une détection de fumée) **ainsi que le changer**.

Il se présente ainsi:

```
#define INT_GPIO GPIO2              // Pin du bouton
bool smokeAlarm = false;            // = Alerte incendie
bool smokeAlarmChanged = false;     // Change si passage d'état

void smokeISR()
{
  pinMode(INT_GPIO, INPUT);
  attachInterrupt(INT_GPIO, smokeISR, BOTH);
}

void smokeISR()
{
  int pin = digitalRead(INT_GPIO);  // -+
  delay(10);
  if (digitalRead(INT_GPIO) == pin) // -+-> Anti-rebond sommaire
  {
    smokeAlarm = pin == LOW ? true : false;
    smokeAlarmChanged = true;
  }
}
```

Dans un premier temps, nous avons décidé de mettre le bouton sur la broche GPIO2. De plus, dès que le bouton sera actionné elle ira dans **smokeISR()**.

On déclare donc **2 variables** qui sont:

- **smokeAlarm**: Correspond à la mise en alerte (ici en appuyant sur le bouton). Avec false = pas de fumée et true = alerte fumée.
- **smokeAlarmChanged**: Correspond au fait qu'un changement d'état à eu lieu, si c'est le cas il passe à "true".

Ensuite on crée une **fonction** nommée **"smokeISR()"**. Dans celle-ci, l'on code justement le **changement d'état** du bouton quand il est pressé, tout en essayant de réaliser un **anti-rebond**. Donc, dès que le bouton est activé, le **smokeAlarm prend la valeur correspondant à l'état**, tout en se basant sur celui précédemment (S'il était à **false**, il passe à **true** et inversement). Enfin le smokeAlarmChanged passe alors à true, dû au changement d'état du bouton.

Partie IR 1 : Muel Mélina

➢ Acquisition et détection de fumée:

Après cela, l'on cherche un code permettant de créer justement l'interruption provoquée par l'alerte, que l'on a encore simulée ici avec un bouton poussoir.

Il se présente ainsi:

```
void loop()
{
  /* Interruption Alerte incendie.*/
  if (smokeAlarmChanged) {
    uint32_t now = TimerGetCurrentTime();
    Serial.print(now);
    Serial.print(" Alerte incendie : ");
    Serial.println(digitalRead(INT_GPIO) == LOW ? "ON" : "OFF");
  }
}
```

Dans ce code, plus exactement le loop(), on fait en sorte que si l'alerte incendie a été activée ou désactivée, il faut:

- **Afficher à partir de quand il a eu lieu**, grâce à la variable "**now**" a qui été attribuée justement le temps accumulé avec "**TimerGetcurrentTime()**".

- **Afficher dans la console "Alerte Incendie:"** ainsi que suivi par **l'état du bouton**. Avec **ON** = Alerte Incendie et **OFF** = Rien.

Partie IR 1 : Muel Mélina

➢ Acquisition et détection de fumée:

Enfin il ne reste plus qu'à réaliser le programme permettant d'envoyer l'alerte incendie à la passerelle, si le bouton à été pressé.

Il se présente ainsi:

```
/* Envoie à la passerelle si alerte / fin de l'alerte incendie */
if (smokeAlarmChanged) {
        if (IsLoRaMacNetworkJoined) {
          if (prepareTxFrame(ASYNC_DATA_PORT)) {
            LoRaWAN.send();
          }
        }
        smokeAlarmChanged = false;
      }
```

Ce programme se trouvera donc dans le code faite pour la connectivité entre la passerelle et la CubeCell et tous ses autres états.

Cet envoie de la trame d'alerte incendie à la passerelle se fera selon **certaines conditions prérequis**. Il faut qu'**un changement d'état** eut lieu, ce que l'on déduit avec **smokeAlarmChanged**. Ensuite, il faut que la **connexion entre la passerelle et la CubeCell** soit faite, ce qui se vérifie avec **IsLoRaMacNetwork-Joined** qui est justement prévu à cet effet. Enfin la dernière condition est présente pour **spécifier où elle doit être envoyée,** ici à **ASYNC_DATA_PORT** qui est le port dédié à la trame d'alerte incendie.

Une fois envoyée, le **changement d'état se réinitialise** car il n'a plus lieu d'être à ce qu'il soit actif. Il devient alors **false**.

Partie IR 1 : Muel Mélina

➢ Acquisition et détection de fumée:

Pour finir, on le regroupe à notre code de base afin vérifier sa compatibilité et obtenir enfin la version complète de ce programme.

```
/* Déclaration */
#define INT_GPIO GPIO2

[...]

/*-------------------------------------------------------*
 |                                                       |
 |                       PAYLOAD                         |
 |                                                       |
 *-------------------------------------------------------*/
/* format du payload */
typedef struct {
  [...]
} frame_t;
frame_t s2ePayload;

bool smokeAlarm = false;
bool smokeAlarmChanged = false;

[...]

/*-------------------------------------------------------*
 |                                                       |
 |                       SETUP                           |
 |                                                       |
 *-------------------------------------------------------*/

void setup() {
  boardInitMcu();
  Serial.begin(115200);
  pinMode(Vext, OUTPUT);

#if(AT_SUPPORT)
  enableAt();
#endif
  deviceState = DEVICE_STATE_INIT;
  LoRaWAN.ifskipjoin();

  smokeAlarmChanged = false;
  pinMode(INT_GPIO, INPUT);
  attachInterrupt(INT_GPIO, smokeISR, BOTH);
  Serial.println("Interrupts attached");
}
```

```
/*-------------------------------------------------------*
 |                                                       |
 |                  ENVOIES DE MESURES                   |
 |                    ALERTE FUMEE                       |
 |                                                       |
 *-------------------------------------------------------*/


void smokeISR()
{
  int pin = digitalRead(INT_GPIO);  // -+
  delay(10);
  if (digitalRead(INT_GPIO) == pin) // -+-> Anti-rebond sommaire
  {
    smokeAlarm = pin == LOW ? true : false;
    smokeAlarmChanged = true;
  }
}


void loop()
{
  /* Interruption Alerte incendie. */
  if (smokeAlarmChanged) {
    uint32_t now = TimerGetCurrentTime();
    Serial.print(now);
    Serial.print(" Alerte incendie : ");
    Serial.println(digitalRead(INT_GPIO) == LOW ? "ON" : "OFF");
  }

  /* CONNECTION A LA PASSERELLE LoRa */
switch ( deviceState )
  {
    [...]
case DEVICE_STATE_SLEEP:
    {
      if (smokeAlarmChanged) {
        if (IsLoRaMacNetworkJoined) {
          if (prepareTxFrame(ASYNC_DATA_PORT)) {
            LoRaWAN.send();
          }
        }
        smokeAlarmChanged = false;
      }
      LoRaWAN.sleep();
      break; }
    }
}
```

L'on obtient donc ce qui va nous servir de code final.

Partie IR 1 : Muel Mélina

➢ Regroupement des différents programmes:

Au début du projet, j'avais rencontré une sorte de problème de compatibilité entre les divers codes des **V1 de chaque code**. Qui est le voici:

Ayant terminé à ce moment le code de transmission de trame de la CubeCell **(V1)** et du SHT21 **(V1)**, mon objectif était de rassembler les deux programmes afin qu'ils fonctionnent ensemble. Ce qui me permettrait de faire en sorte de **récupérer les valeurs** de: la température, l'humidité, le point de rosé; afin de tenter de les rajouter au **payload** envoyé à la passerelle dont mon collègue, **l'IR 2** sera chargé de réceptionner et les traduire.

A ce moment, la réussite était assez aléatoire pour des raisons qui me sont obscures pour le moment.

J'ai donc tenté plusieurs façons différentes:

- ➔ **Test N°1**: Au début, J'ai placé le code du SHT21 en premier puis en second celui de la transmission de trame. Je n'avais alors que l'acquisition du SHT21 affiché et la communication CubeCell/Passerelle LoRa ne se faisait plus.
- ➔ **Test N°2:** Ensuite j'ai tenté le contraire, placer le code de la transmission de trame en premier puis celui du SHT21. Cela donnait le même résultat que le Test N°1.
- ➔ **Test N°3:** J'ai placé cette fois-ci le code du SHT21 dans celui de la transmission de trame. Plus particulièrement dans un des cases nommé "DEVICE_STATE_INIT". Il a fonctionné un temps, puis c'est là que je ne comprends pas encore pourquoi il ne fonctionnait plus le lendemain… Maintenant, il ne fait que la connexion CubeCell/Passerelle.

D'ailleurs, depuis ce mystère, peu importe où je plaçais le code du SHT21 il ne faisait plus les acquisitions. Du moins, s'il était assemblé avec le code de transmission.

Partie IR 1 : Muel Mélina

➢ Regroupement des différents programmes:

Maintenant, avec les **V2 de chaque code**, même rassembler les données **s'envoient** belle est bien sur la passerelle.

L'on retrouve donc bien ceci:

- Un **port dédié** aux **mesures** (**PERIODIC_DATA_PORT**) telles que température et humidité, ainsi que le **calcul** du point de rosée.
- Un **port dédié aux alertes incendies** (**ASYNC_DATA_PORT**), où dès qu'un **changement d'état** a lieu il reçoit le signal.

Cependant un nouveau **problème** s'était présenté face à nous, il s'agit de **l'ordre des valeurs reçues** par la passerelle. Pour une raison que j'ignore, celle-ci reçoit mes valeurs mais dans le **désordre**.

Comme illustré ci-dessous:

Après une discussion avec le **IR2**, nous avons convenus que cela n'avait pas vraiment d'importance. Pour la simple et bonne raison que celui-ci aurait juste à bien cibler de son côté quelle valeur est à quelle mesure.

Partie IR2:

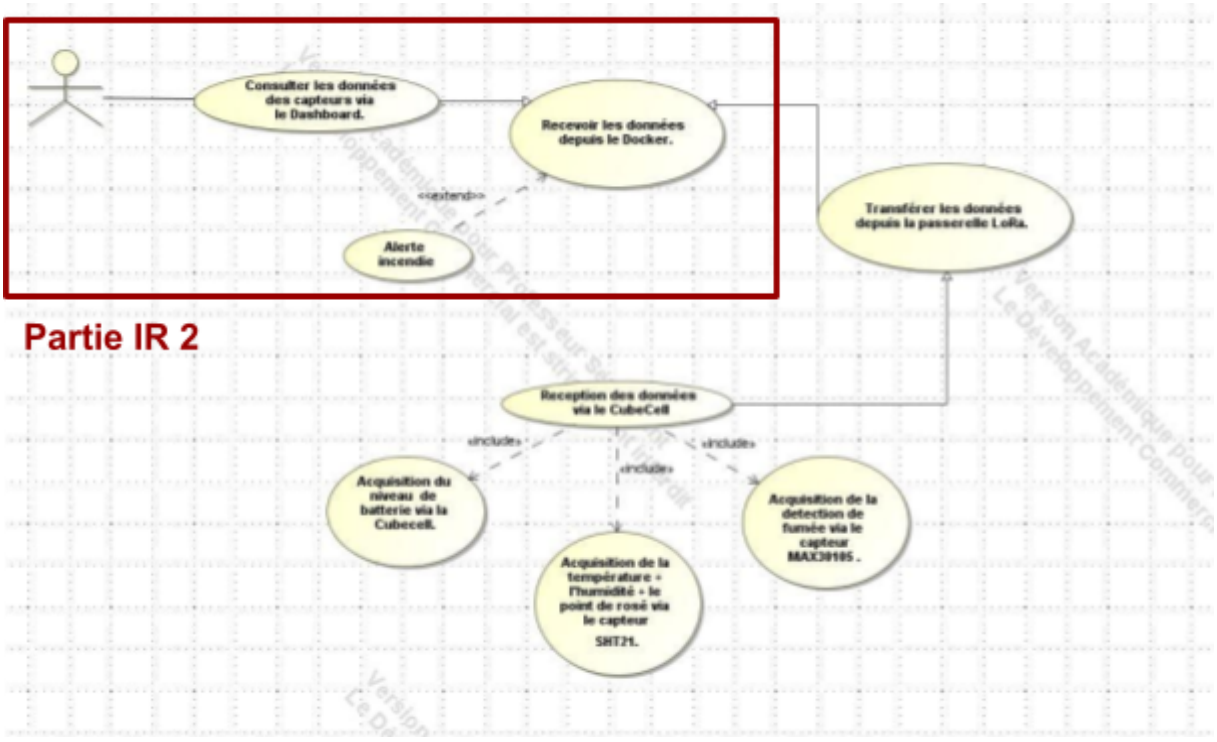
JEAN Romain

Partie IR 2 : JEAN Romain

➢ Objectifs

- Mettre en service la passerelle LoRa Multitech.
- Conversion de la trame de la CubeCell en trame JSON.
- Installer Mosquitto dans un conteneur Docker.
- Tester la communication avec un client MQTT.
- Concevoir/Coder/Tester deux Dashboards (NodRed et QT).

➢ Diagramme Cas d'utilisation



L'utilisateur peut consulter les données depuis le graphique (Dashboard), montrant chaque valeurs des capteurs. Les données sont stockées dans le docker, données qui sont elles-même reçues via la CubeCell et transférées par la passerelle LoRa.

Partie IR 2 : JEAN Romain

➢ LoRa

LoRa qui signifie Long-Range est un système de communication sans-fil longue portée. Il est très peu gourmand en énergie et il vise à être utilisable dans des appareils alimentés par batteries.

Le terme LoRa est communément utilisé pour désigner deux couches réseaux distinctes du modèle OSI :

- La couche physique.
- La couche liaison.

La couche physique LoRa permet des communications à longue portée, à faible puissance et à faible débit. Elle fonctionne sur les bandes ISM de 433, 868 ou 915 MHz, selon la région où elle est déployée (868 MHz en Europe).

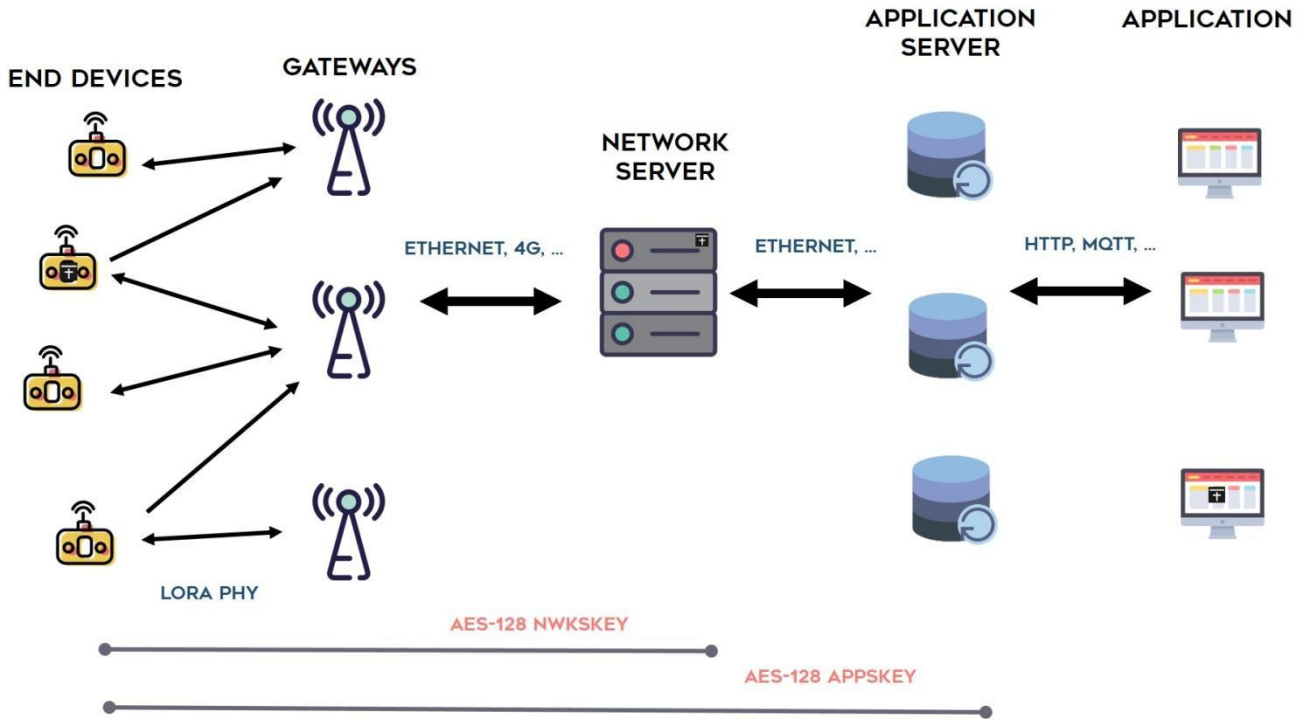
La charge utile (= payload) de chaque transmission peut varier de 2 à 255 octets, et le débit de données peut atteindre 50 Kbps lorsque l'agrégation de canaux est utilisée.

La couche de liaison permet à des appareils basse consommation de communiquer avec des serveurs réseaux via des passerelles qui relaient des paquets radio LoRa qu'il reçoit ou envoie sur des réseaux traditionnels (3G ou Ethernet).

Partie IR 2 : JEAN Romain

➢ LoRa

Un réseau LoRa typique possède une topologie en étoile et comporte 4 types d'équipements.



1) Les **end-devices/nodes/motes** qui représentent les appareils basse-consommation.

2) Les **gateways** qui relayent les paquets des end-devices dont ils reçoivent ou envoient au serveur réseaux (= network server).

3) Le **network server** qui décode les paquets LoRa provenant des end-devices puis les envoie aux applications finales, encode les réponses des applications pour les envoyer aux end-devices.

4) **L'application server** qui héberge les applications traitant les données des end_devices.

Partie IR 2 : JEAN Romain

➢ LoRa

Dans cette partie nous allons relever tous les points importants de LoRa.Nous allons donc relever la fréquence, la longueur d'onde et la durée d'émission.

LoRa peut avoir une fréquence qui varie en fonction des pays elle sont de 433, 868 ou 915 MHz.

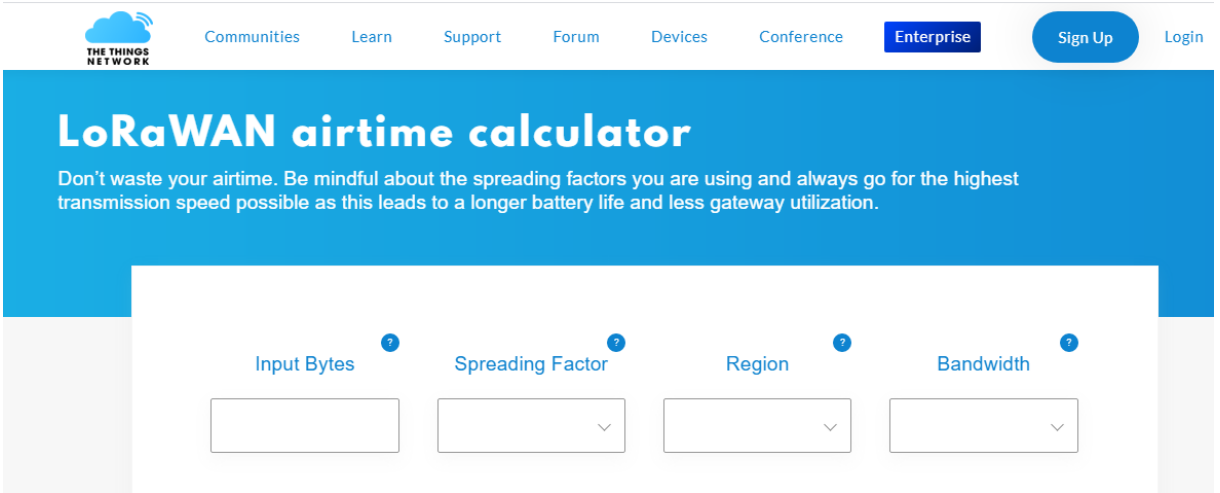
Dans notre cas la fréquence est à 868 MHz (ce qui est la fréquence en Europe), une fois cela fait nous allons calculer la longueur d'onde en nous servant de la fréquence obtenue plus haut.

Longueur d'onde:

$$\lambda = c / f = 3.10^8 / 868.10^6 = 0.3456 \approx 0.346 \text{ m}$$

Après ça nous allons calculer sa durée d'émission grâce à ce site internet:

https://www.thethingsnetwork.org/airtime-calculator

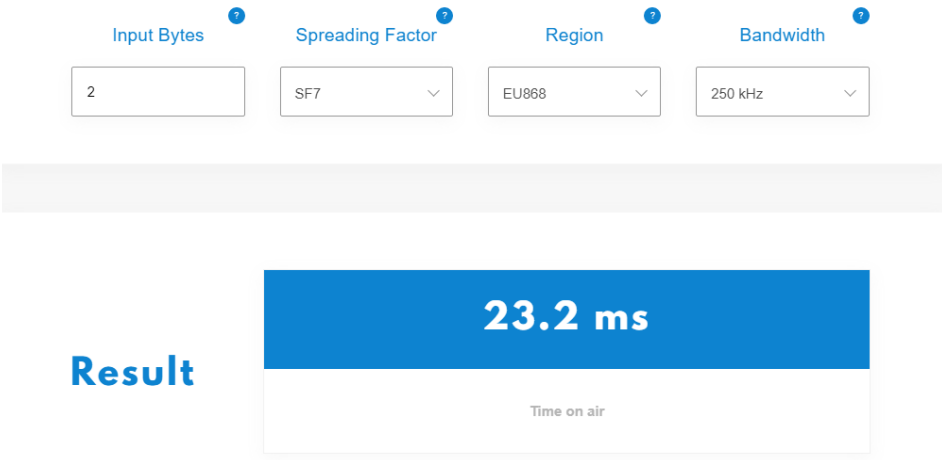


Partie IR 2 : JEAN Romain

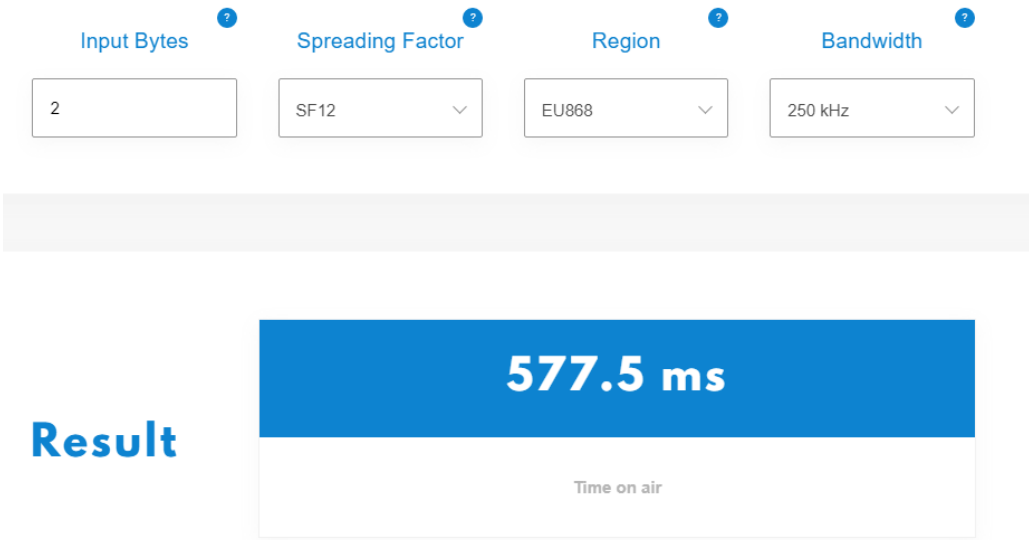
➢ LoRa

Dans notre cas, le spreading factor est automatiquement ajusté par le Protocol Lorawan en fonction de la force du signal donc nous calculeront le SF (spreading factor) minimum et maximum.

SF minimum (SF 7):



SF maximum (SF 12):

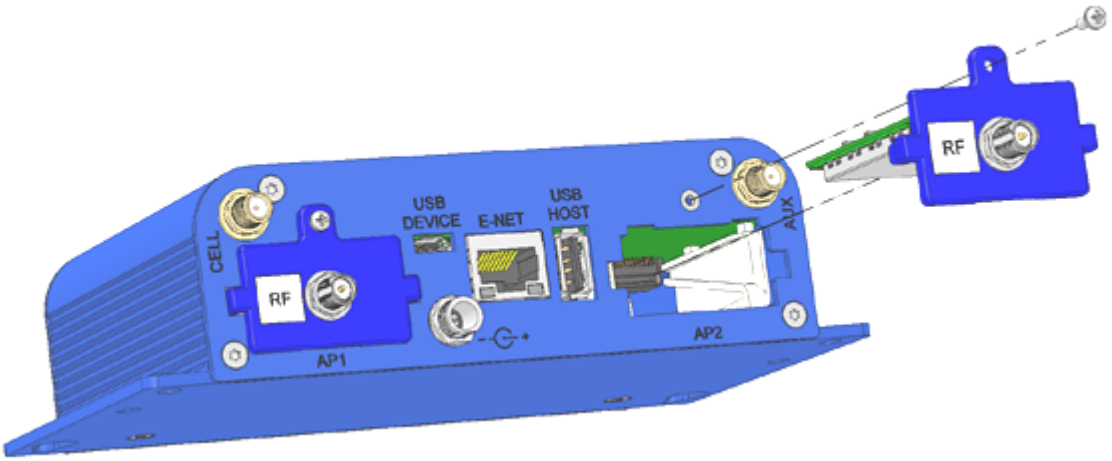


Partie IR 2 : JEAN Romain

➢ Mise en place de la Passerelle LoRa

Branchement de la passerelle :

Lors du branchement de la passerelle nous utiliserons simplement les connectiques Ethernet et RF ainsi que l'alimentation.



RF : Permet la communication par radio fréquence avec d'autres dispositifs (dans notre cas le module m-dot ou la carte CubeCell).

Ethernet : Permet de l'identifier sur notre réseau, la passerelle nous laisse accéder à son adresse IP pour la configurer.

Accès à la passerelle :

Après avoir branchée la passerelle nous devons y accéder depuis un navigateur web pour la configurer.

Pour cela nous devons d'abord trouver son adresse ip sur le réseau notamment grâce à des logiciels tel que NMAP ou Angry IP Scanner.

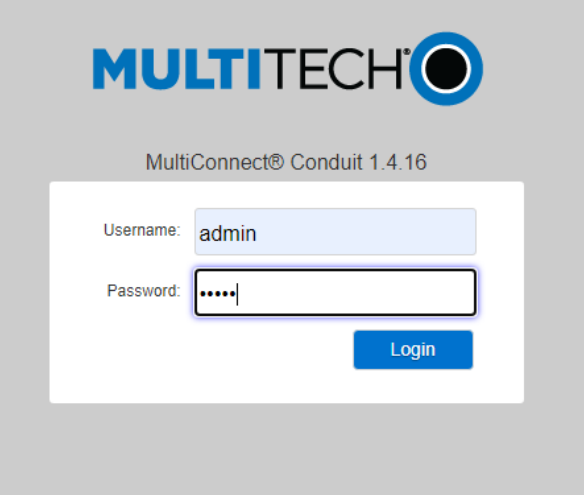
Après avoir trouvé son adresse (dans notre cas 192.168.4.79) aller dans un navigateur et taper la pour ouvrir la page de connexion à la passerelle

Partie IR 2 : JEAN Romain

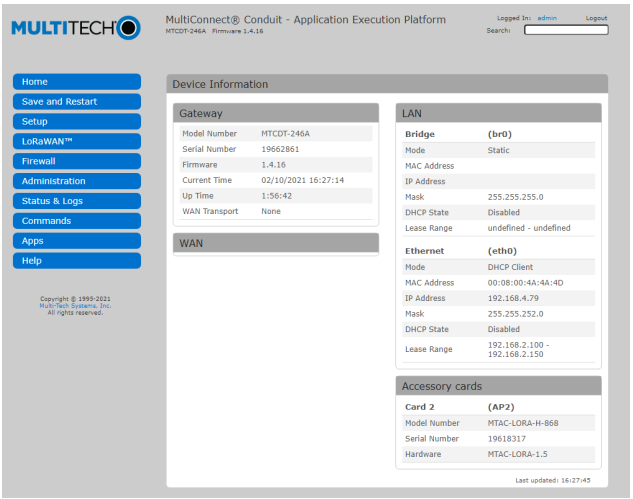
➢ Mise en place de la Passerelle LoRa

Configurer la passerelle :

Une fois sur la page de configuration vous devrez vous connecter avec le mot de passe et login de base sont admin.



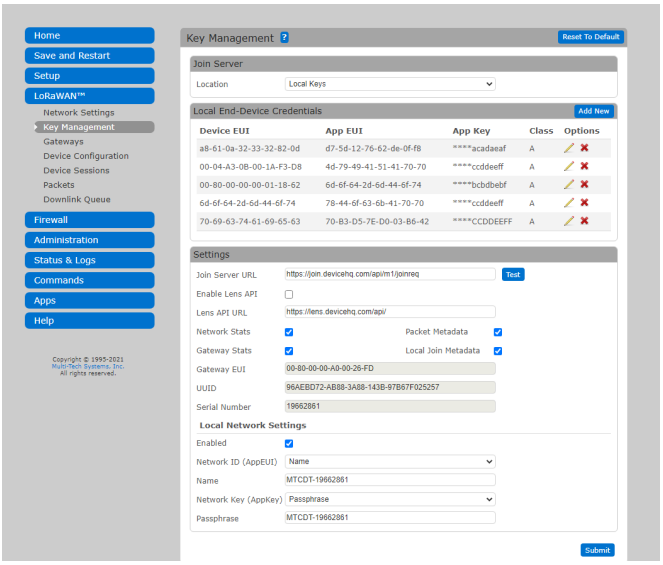
Après la connexion la page d'accueil de la passerelle s'ouvre nous laissant la configurer.



Beaucoup d'options de configuration nous sont données mais nous allons simplement nous intéresser à l'onglet LoRaWAN et allons ouvrir le sous onglet Key Management.

Partie IR 2 : JEAN Romain

➢ Mise en place de la Passerelle LoRa



Cela nous permet de configurer l'accès à notre passerelle par d'autre module.

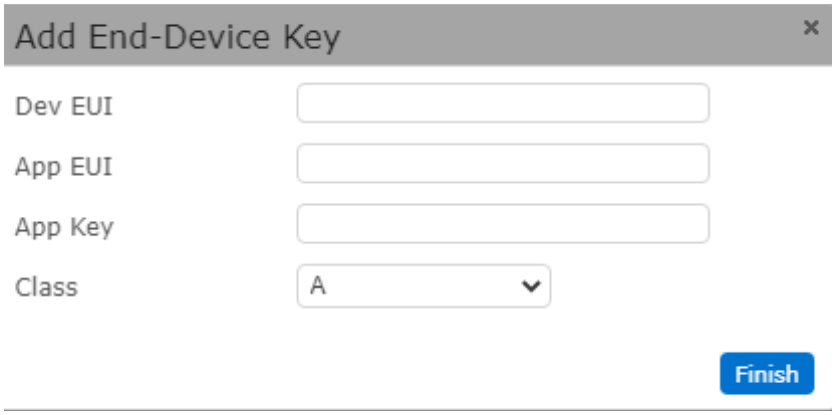
Dans notre cas nous allons simplement ajouter un nouveau Module avec l'option Add New.

Un module est généralement un appareil qui communique avec la passerelle LoRa, dans notre cas ce sera notre CubeCell.

L'ajout du nouveau module nous permettra d'ajouter notre CubeCell et de la lier à notre passerelle Lora afin qu'ils puissent communiquer.

Partie IR 2 : JEAN Romain

➢ Mise en place de la Passerelle LoRa



Pour ajouter ce nouveau module nous devons remplir les champs suivant:

Dev EUI: Le Dev EUI fonctionne comme un identifiant au Module en langage hexadécimal.

App EUI: App EUI sert a nommé le Module en langage hexadécimal.

App Key: Autorise le module à communiquer si l'app key est différent, aucune communication n'est établie.

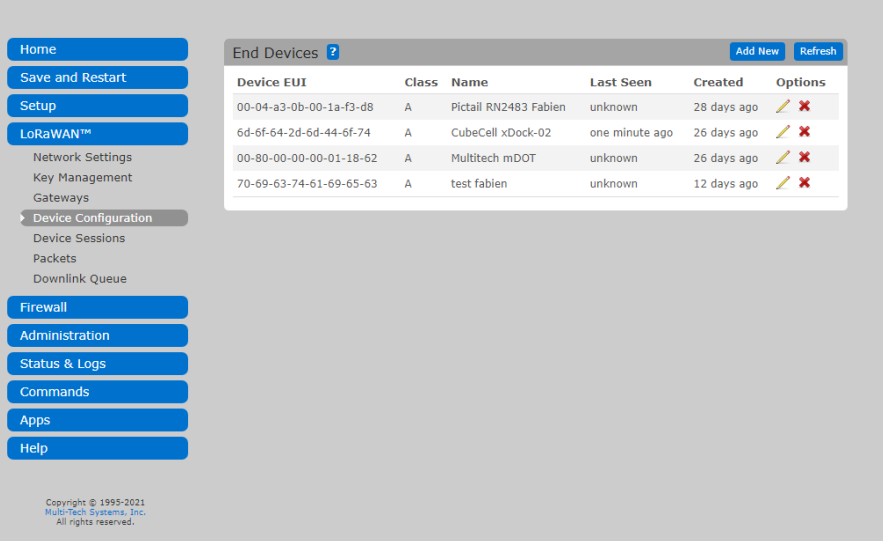
Class: définit dans quel groupe appartient le module

La classe nous importe peu dans notre cas et les valeurs hexadécimals sont définies dans le code de la CubeCell.

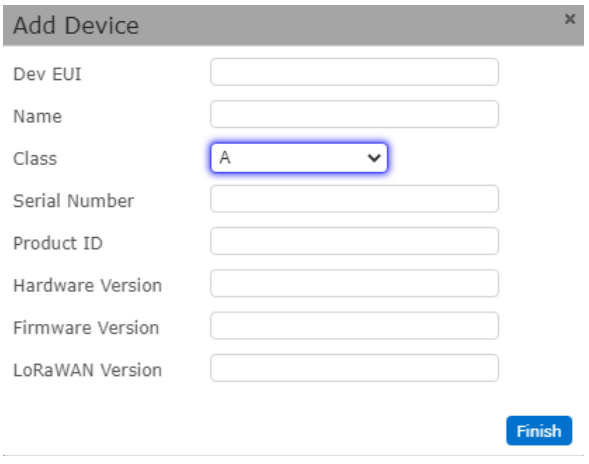
Partie IR 2 : JEAN Romain

➢ Mise en place de la Passerelle LoRa

Après avoir ajouté l'accès du module à la passerelle nous devons configurer le module pour que la passerelle le reconnaisse, pour cela nous devons aller dans Device Config.



Cela nous permet de configurer un Module, cliquer sur add new.



Entrer l'identifiant du Module (Dev EUI), pour le Nom du module, pour la classe laisser la valeur A.

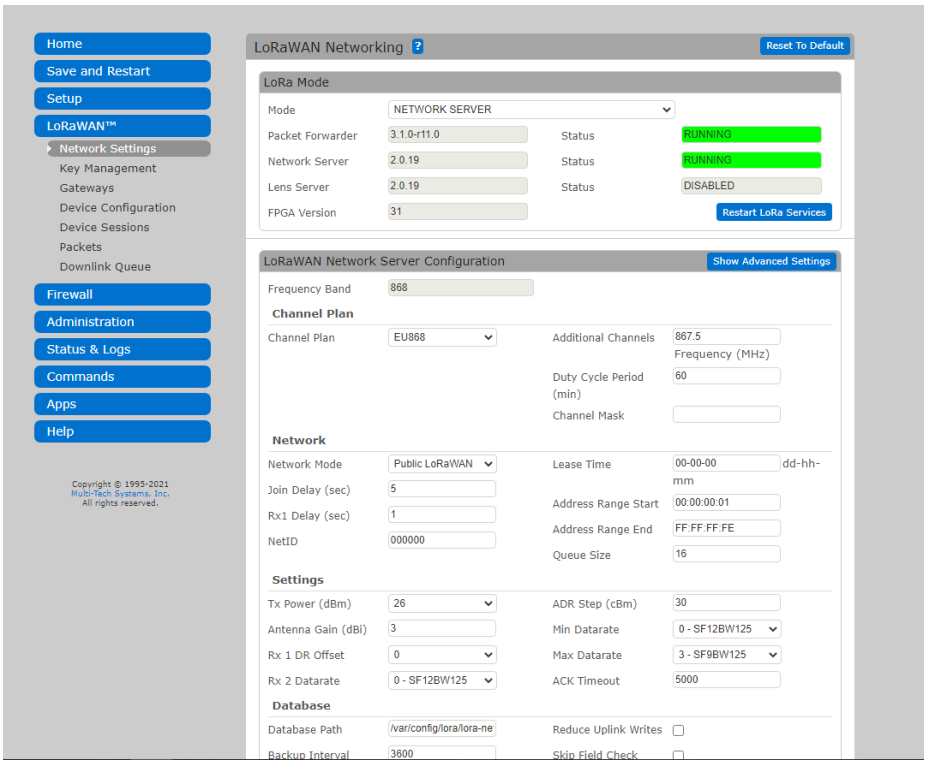
Partie IR 2 : JEAN Romain

➢ Mise en place de la Passerelle LoRa

ATTENTION:

Après chaque modification, il faut relancer les Service LoRa pour pouvoir prendre les modifications en compte.

Pour cela aller dans Network Setting et cliquer sur Restart LoRa Services.



Partie IR 2 : JEAN Romain

➢ Nod-Red

Node-RED est un outil de programmation basé sur les flux, qui est une manière de décrire le comportement d'une application comme un réseau de boîtes noires, ou «nœuds» comme on les appelle dans Node-RED.

Les nœuds correspondent à du code préfait, ce qui facilite la programmation en la rendant plus rapide et facile d'accès.

Par exemple une node "lora": 

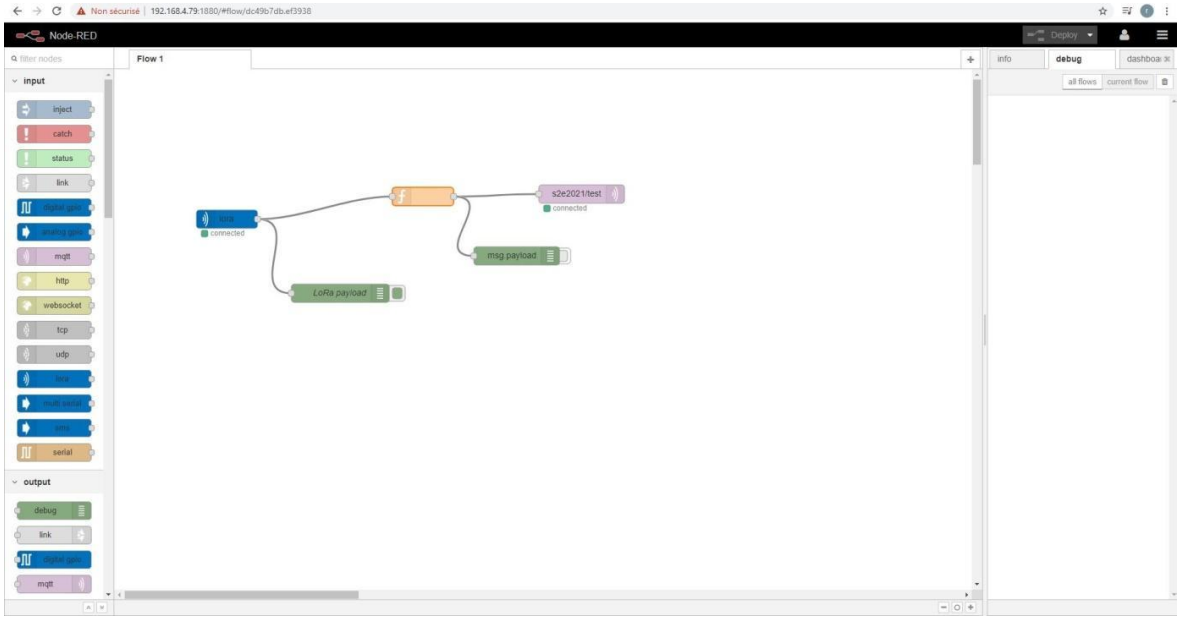
Elle aura pour principe de se connecter automatiquement à la passerelle, ce qui nous facilite grandement la programmation.

C'est un logiciel que l'on peut installer sur différent support tel qu'une passerelle LoRa, une Raspberry ou bien un serveur. On peut créer une application en faisant glisser les nœuds de notre palette dans un espace de travail et commencer à les relier. En un seul clic, l'application est de nouveau déployée dans l'environnement d'exécution où elle est exécutée.

Partie IR 2 : JEAN Romain

➢ Conversion de la trame de la CubeCell en trame JSON

La trame reçue par la CubeCell doit être convertie en une trame JSON pour pouvoir comprendre les infos reçues. Pour cela nous allons utiliser NodRed, pour que la trame reçue par la passerelle soit directement transformée en trame JSON et envoyer au docker. Pour cela, nous avons assembler plusieurs noeux comprenant (LoRa, debug function et MQTT).



Après cela nous devons coder le nœud fonction pour permettre la conversion de la trame en JSON.

Partie IR 2 : JEAN Romain

➢ Conversion de la trame de la CubeCell en trame JSON

```
function computeDataPayload() {
   var ts = msg.payload.readInt16LE(0); // IMPORTANT : configurer le payload du noeud LoRa en tant
que "bytes" plutôt que "utf-8"
   var t;
   var h = msg.payload.readInt8(2);
   var dp;
   var bat = msg.payload.readInt8(3);

   if (ts & 0x8000) {         //recup temp
     t = null;
   } else {
      t = ts & 0x1fff;
      if( ts & 0x1000) {
         t = (-1 & ~0x1fff) | t;
      }
      t /= 10;
   }

if( (t !== null) && (h !== null)) {
      dp =
243.04*(Math.log(h/100)+((17.625*t)/(243.04+t)))/(17.625-Math.log(h/100)-((17.625*t)/(243.04+t)));
   } else {
      dp = null;
   }

   msg.payload = {
      _dt: Math.round(Date.now() / 1000),    //arrondie la valeur et la divise pour la rendre lisible
      temperature : t,
      humidity : h,
      dewpoint : parseFloat(dp.toFixed(1)),
      battery : bat,
   }

   msg.topic = "crossdock/data/" + msg.deveui;

}
```

Partie IR 2 : JEAN Romain

➢ Conversion de la trame de la CubeCell en trame JSON

```
function computeAlertPayload() {
    var s = msg.payload.readInt8(0);

    msg.payload = {
        _dt: Math.round(Date.now() / 1000),
        smoke : s ? true : false
    }

    msg.topic = "crossdock/alert/" + msg.deveui;

}

switch(msg.port) {
    case 1 : // Message périodique pour les mesures de capteur
        computeDataPayload();
        break;
    case 2 : // Message asynchrone pour les alertes
        computeAlertPayload();
        break;
}

return msg;
```

Ce code sert à convertir les valeurs des capteurs de température, d'humidité, de fumée et de batterie pour les rendre compréhensibles par l'humain tout ça en lisant simplement le « payload ».

Il envoie également la fumée sur un port différent pour être prévenu immédiatement.

Payload = endroit où les données sont présentes.

Partie IR 2 : JEAN Romain

➢ Docker

En termes simples, Un Docker est une plate-forme logicielle qui simplifie le processus de création, d'exécution, de gestion et de distribution d'applications. Pour ce faire, il virtualise le système d'exploitation de l'ordinateur sur lequel il est installé et en cours d'exécution.

C’est donc une solution de virtualisation nommée docker.

➢ Docker Mosquitto

Nous allons commencer par installer le Docker (dans notre cas il servira à stocker les données reçues par la passerelle LoRa) pour cela nous devons nous rendre sur le site de docker (https://www.docker.com/) et télécharger le .exe d’installation.

Après l’avoir installé, nous allons ajouter Mosquitto au Docker.

Mosquitto implémente les versions de protocole MQTT 5.0, 3.1.1 et 3.1, il est léger et convient à tous les appareils, des ordinateurs basse consommation aux serveurs complets.

Pour cela il faut entrer la commande suivante sur windows PowerShell (exécutez en mode admin).

```
% docker run -it --name mosquitto -p 1883:1883 eclipse-mosquitto
```

Cela permet la création de Mosquitto dans notre Docker qui se lancera à chaque démarrage du docker.

Nous utiliserons le logiciel MQTTFX pour visualiser les données sous MQTT.

Partie IR 2 : JEAN Romain

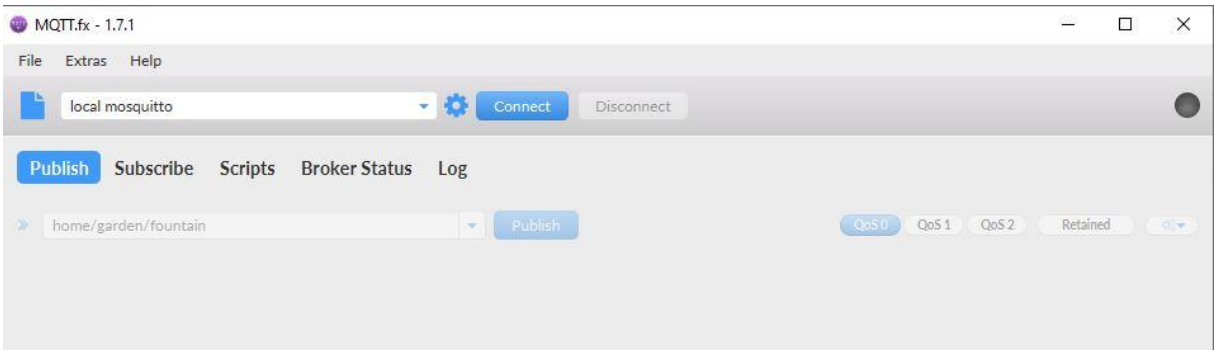
➢ Protocole MQTT

MQTT est un protocole de messagerie basé sur le système publish-subscribe. Il utilise les protocoles TCP/IP pour fonctionner. Le protocole MQTT utilise différentes systèmes et technologies pour fonctionner :

- **Broker:** Service permettant la mise en œuvre de gestion de flux.
- **Topic:** Objet du broker qui sert au fonctionnement du protocole. Permet la publication de données.
- **Souscription (= subscribe):** S'abonner à un Topic pour récupérer les données depuis le serveur MQTT.
- **Publication (= publish):** Envoyer des données sur un Topic du serveur MQTT.

➢ Visualisation des données

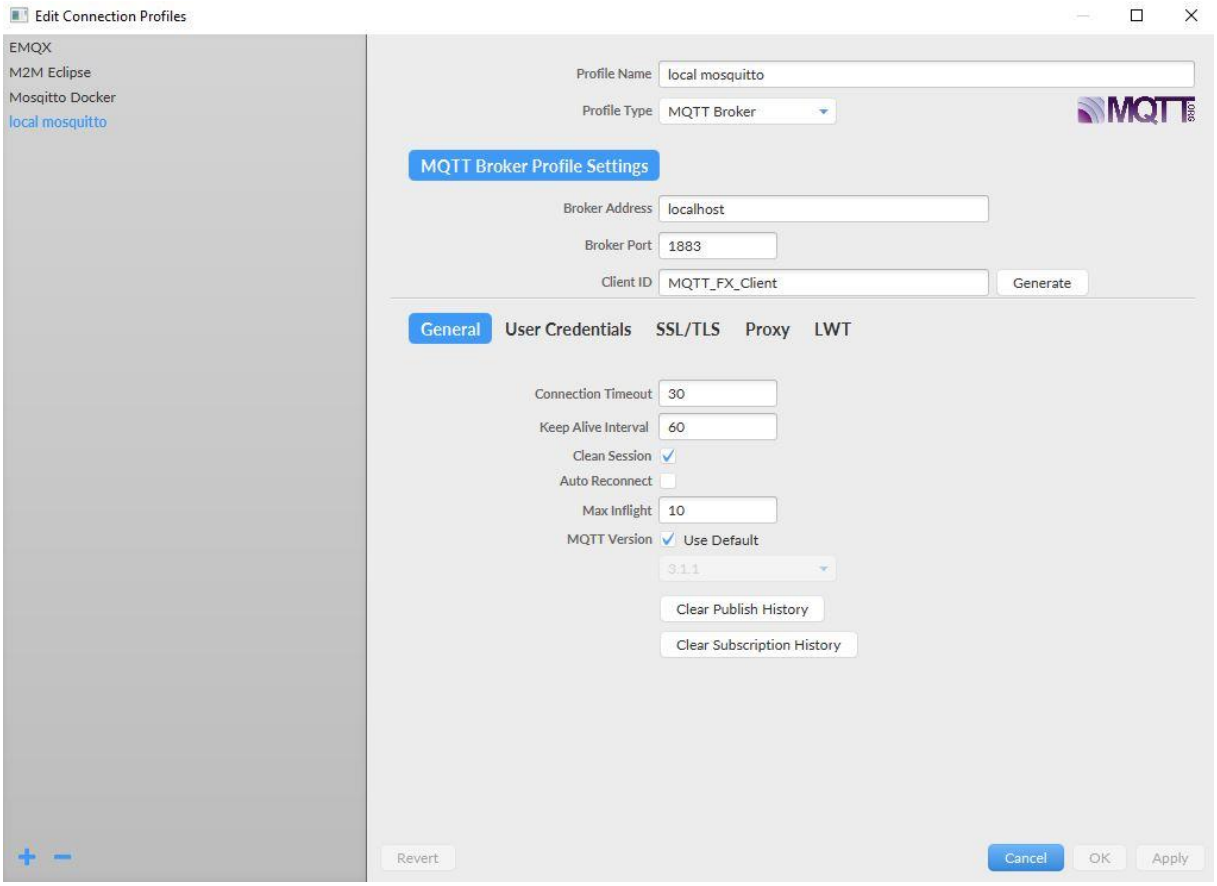
Pour visualiser les données nous allons utiliser le logiciel MQTTFX.



Sur la page principale de MQTTFX, il faudra d'abord créer le profil de notre Docker Mosquitto pour cela il faudra cliquer sur la roue pour que la page de création apparaît.

Partie IR 2 : JEAN Romain

➢ Visualisation des données



Le point le plus important dans cette page sera le broker adresse et broker port qui serviront à associer notre docker mosquitto a MQTTFX.

Broker adresse: Correspond à l'adresse du serveur, dans notre cas le logiciel Docker.

Broker port: Correspond au port du serveur, dans notre cas le logiciel Docker.

Nous laissons tous les autres paramètres par défaut.

Partie IR 2 : JEAN Romain

➢ Installation et configuration des conteneurs

Après l'avoir installé, nous allons ajouter le conteneur Mosquitto au Docker.

Mosquitto implémente les versions de protocole MQTT 5.0, 3.1.1 et 3.1, il est léger et convient à tous les appareils, des ordinateurs basse consommation aux serveurs complets.

Conteneur Mosquitto:

Pour installer le conteneur Mosquitto il faut entrer la commande suivante sur windows powerShell (exécutez en mode admin).

```
docker run -it --name mosquitto -p 1883:1883 eclipse-mosquitto
```

Voici une explication détaillée de la commande:

docker run: Exécute le conteneur.

-it: Attacher une session de terminal afin que nous puissions voir ce qui se passe.

--name: Donne à la machine un nom local (dans notre cas mosquitto car --name mosquitto).

-p 1883:1883: Connecte le port local 1883 au socket par défaut de MQTT qui est 1883.

Partie IR 2 : JEAN Romain

➢ Installation et configuration des conteneurs

Conteneur Node-RED:

Pour installer le conteneur Node-RED, il faut entrer la commande suivante sur windows powerShell (exécutez en mode admin).

```
docker run -it -p 1880:1880 -v node_red_data:/data --name mynodered nodered/node-red
```

docker run: Exécute le conteneur.

-it: Attacher une session de terminal afin que nous puissions voir ce qui se passe.

-p 1880:1880: Connecte le port local 1880 au socket par défaut de MQTT qui est 1880.

-v node_red_data:/data: Monter le répertoire hôte node_red_data dans le répertoire container / data afin que toutes les modifications apportées aux flux soient conservées.

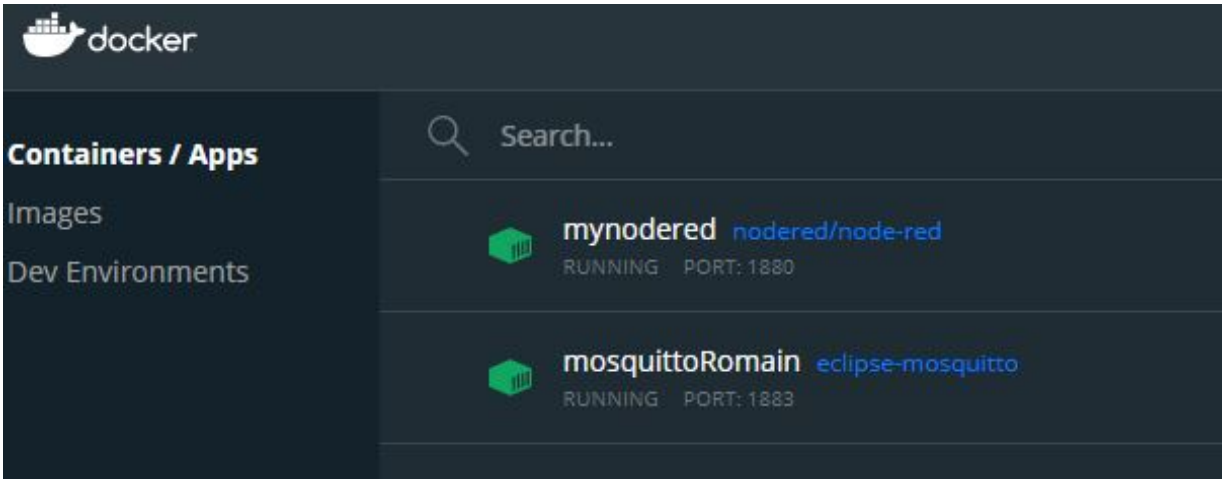
--name: Donne à la machine un nom local (dans notre cas mynodered car --name mynodered).

nodered/node-red: L'image sur laquelle il se base actuellement Node-RED v1.2.0.

Partie IR 2 : JEAN Romain

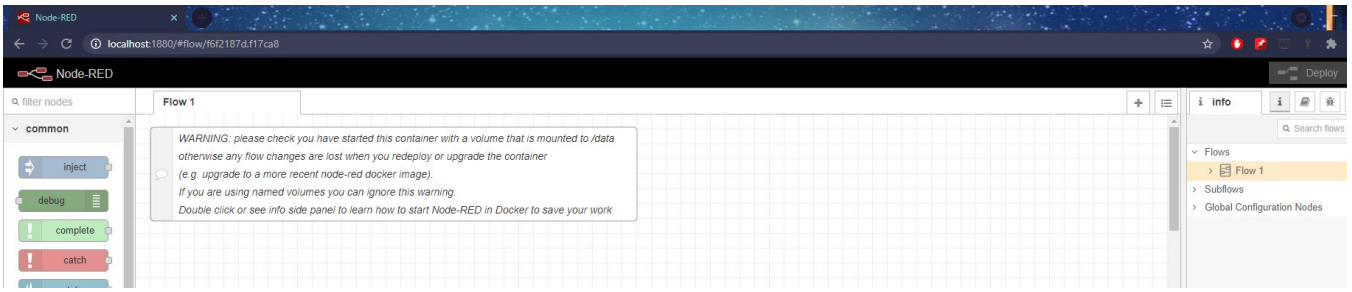
➢ Visualisation graphique

Une fois ces 2 commandes effectuées, l'installation des conteneurs sur notre docker est terminée.



Nous allons ensuite visualiser graphiquement nos données grâce au conteneur Node-RED.

Pour cela nous devons lancer Node-RED en localhost depuis internet, pour cela ouvrir un navigateur internet et taper localhost:port de Node-RED (dans notre cas localhost :1880).

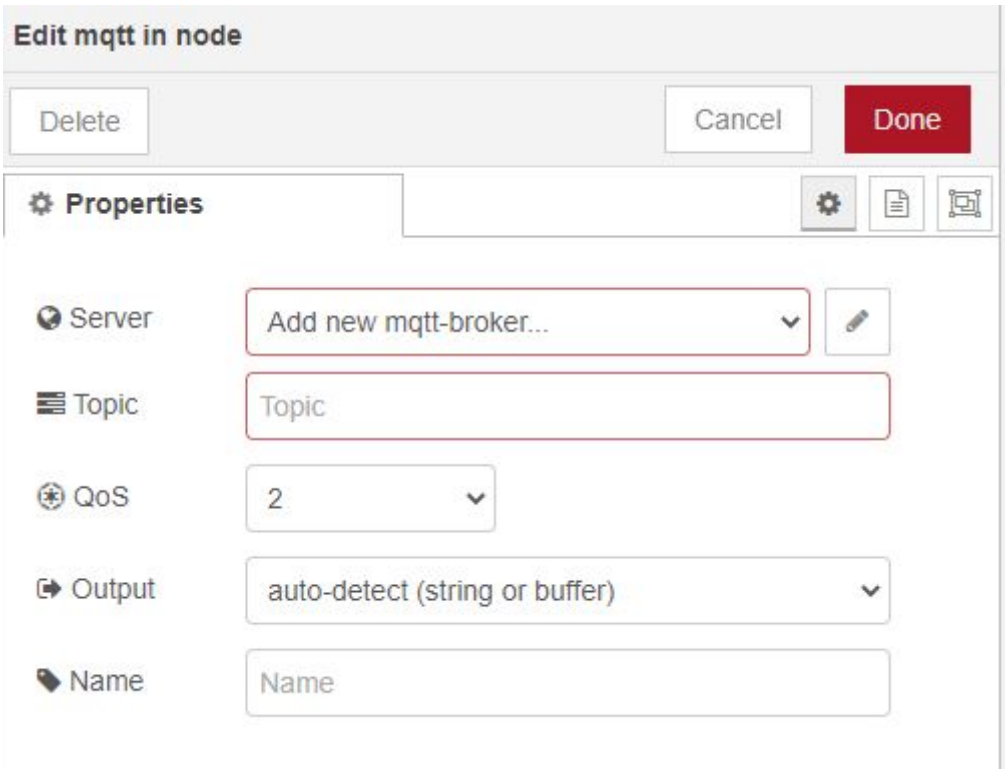


Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Une fois node-REd ouvert nous allons commencer à relier le conteneur Mosquitto au Conteneur Node-RED, ce qui permettra la communication entre les 2 (les données pourront alors atteindre le conteneur Node-RED).

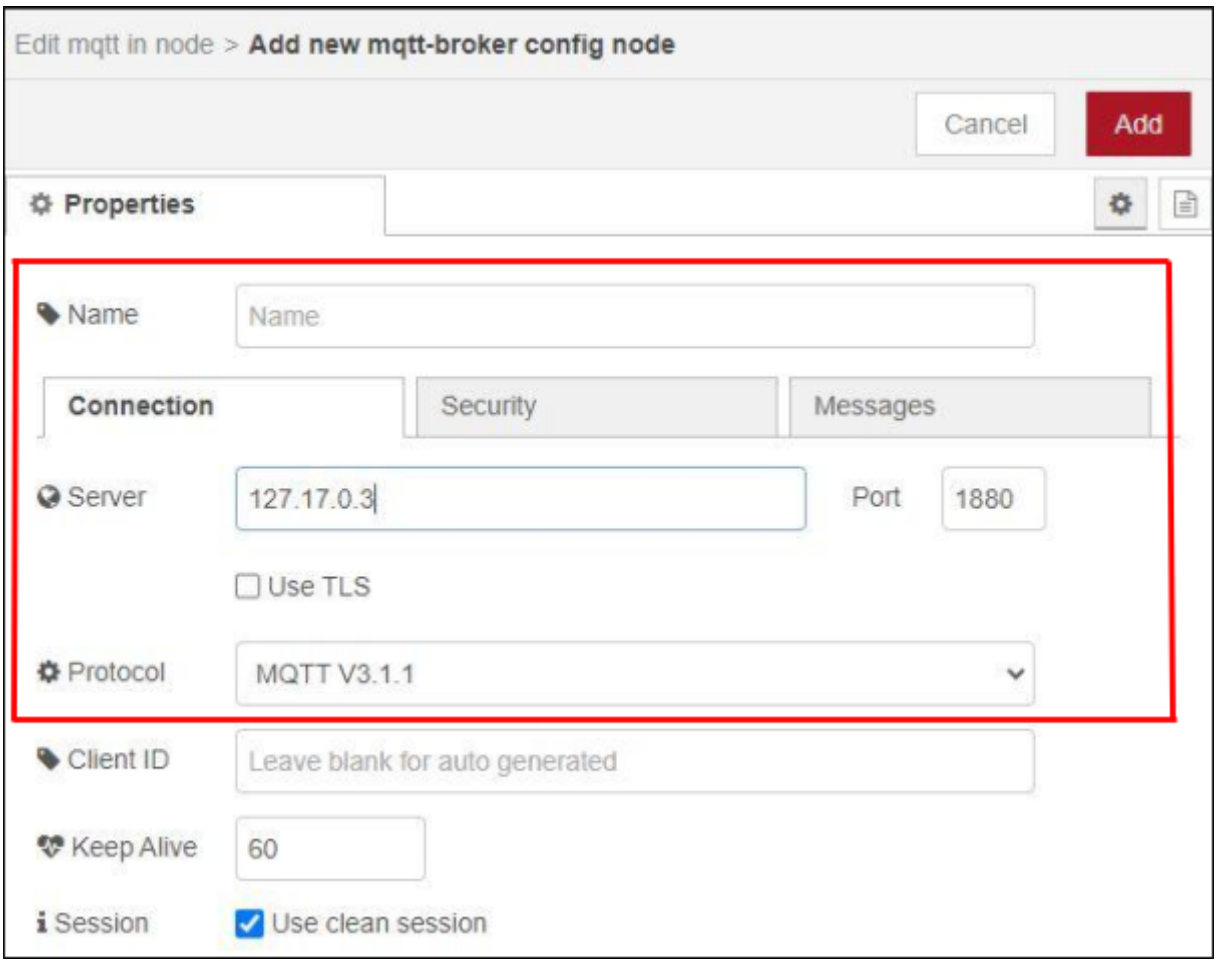
Pour cela il faut déplacer le node mqtt-in sur la grille et double cliquer dessus pour la configurer.



Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Une fois sur l'onglet serveur, mettre add new mqtt-broker et cliquer sur l'icône du stylo à côté.



Name: Nom donné à votre serveur.

Server: Mettre l'IP de votre serveur dans notre cas celui du conteneur Mosquitto.

Protocol: Protocole à utiliser.

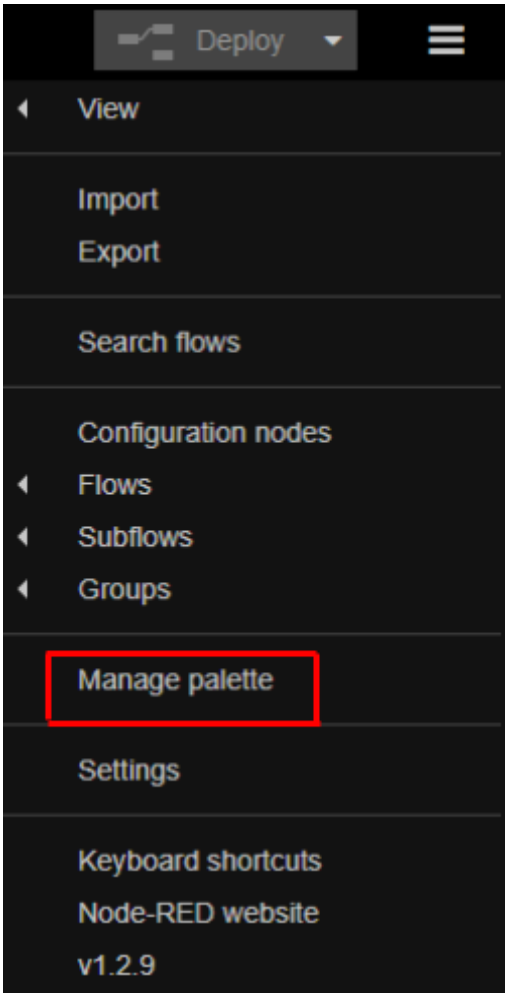
Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Une fois ceci fait les conteneurs sont connectés, nous devons ensuite installer l'extension dashboard du conteneur pour pouvoir créer l'interface graphique.

Pour cela sur le conteneur Node-RED dans votre navigateur, aller en haut à droite de l'écran sur l'icône des 3 barres horizontale et cliquer sur manage palette.

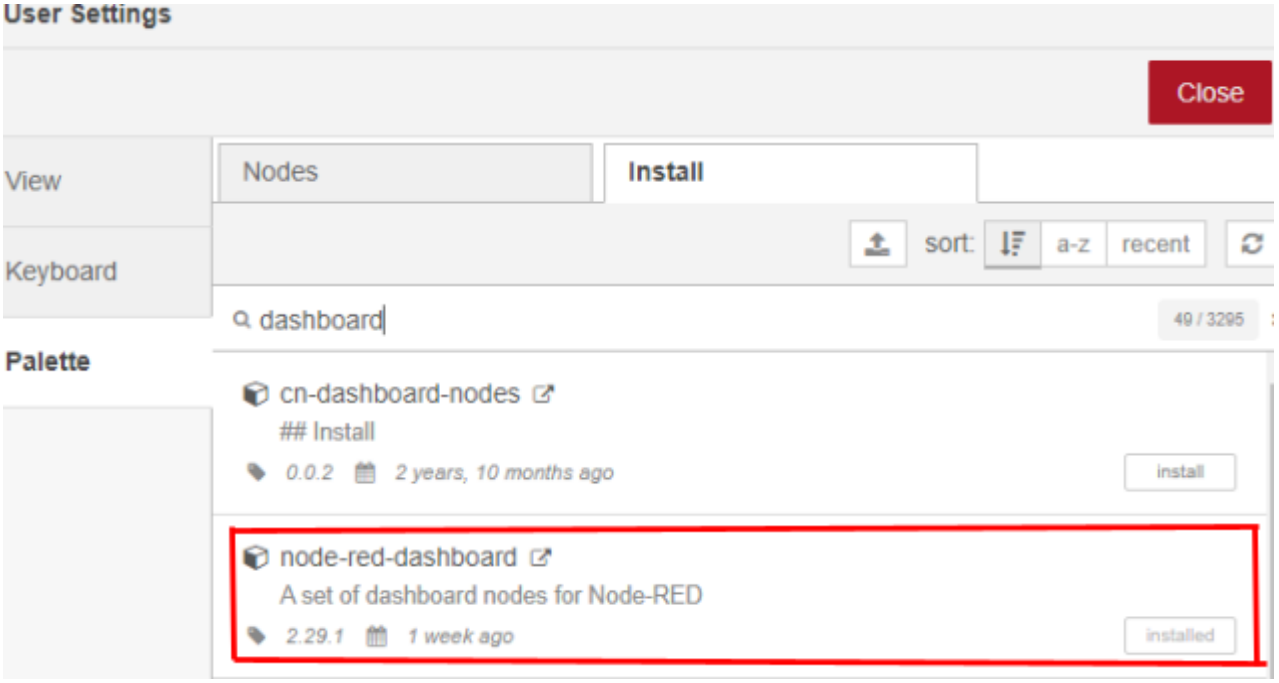
Une fenêtre s'ouvre pour aller sur install et taper dashboard.



Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Une fois cela fait selection Node-RED dashboard et installer le.

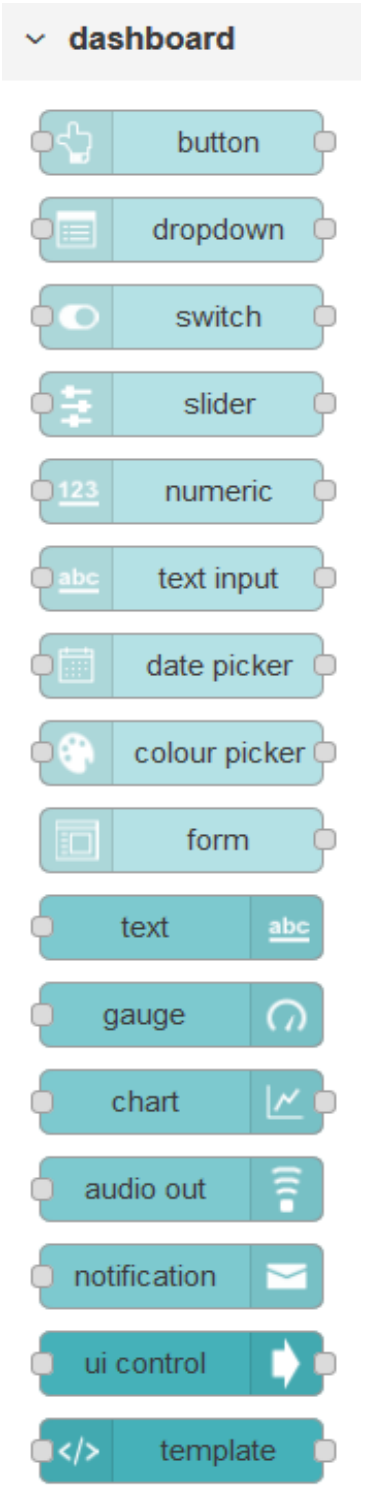


Une fois la librairie installée, il se peut qu'il faut actualiser la page appuyez donc sur F5 pour la recharger.

Partie IR 2 : JEAN Romain

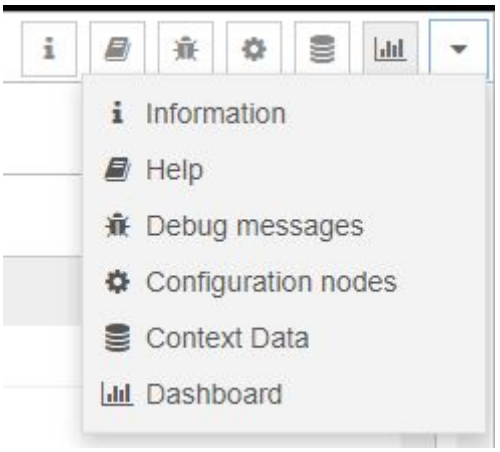
➢ Visualisation graphique

Une nouvelle palette s'offre donc à nous, nous permettant donc de créer une interface graphique.

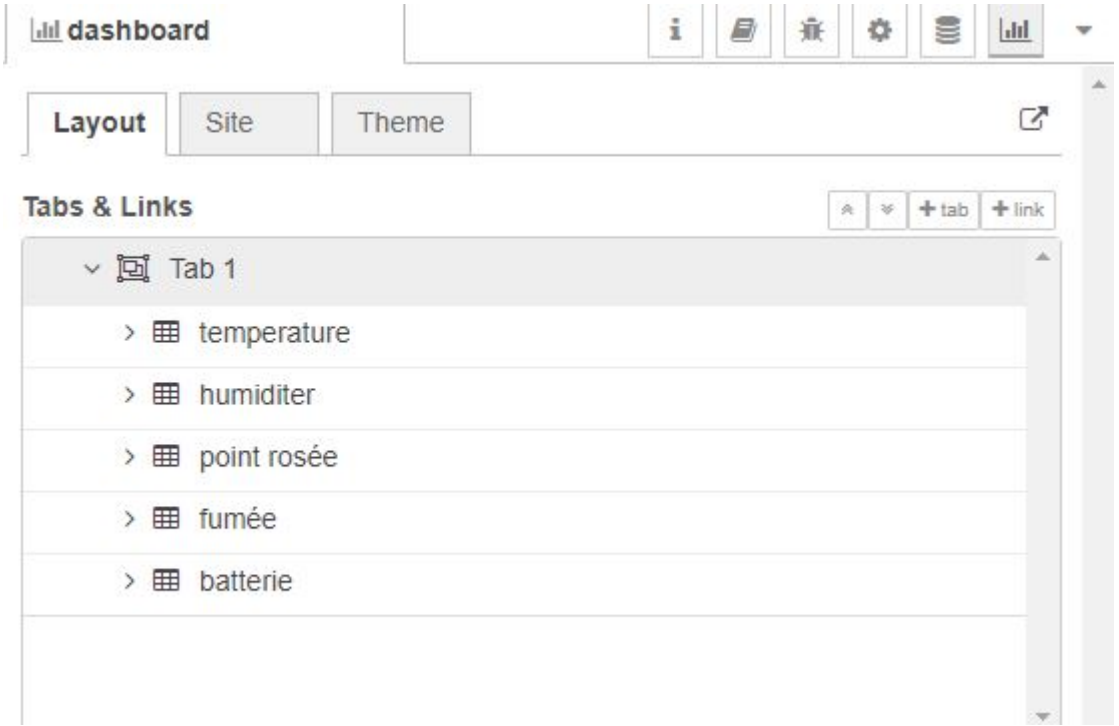


➢ Visualisation graphique

Une fois l'extension installée, recharger la page et cliquer sur l'icône de la flèche à droite de l'écran et sélectionner dashboard.



Une autre fenêtre s'ouvre elle nous permet de créer des groupe (les groupes servent de repère à une node exemple la node graph de la température appartient au groupe température).

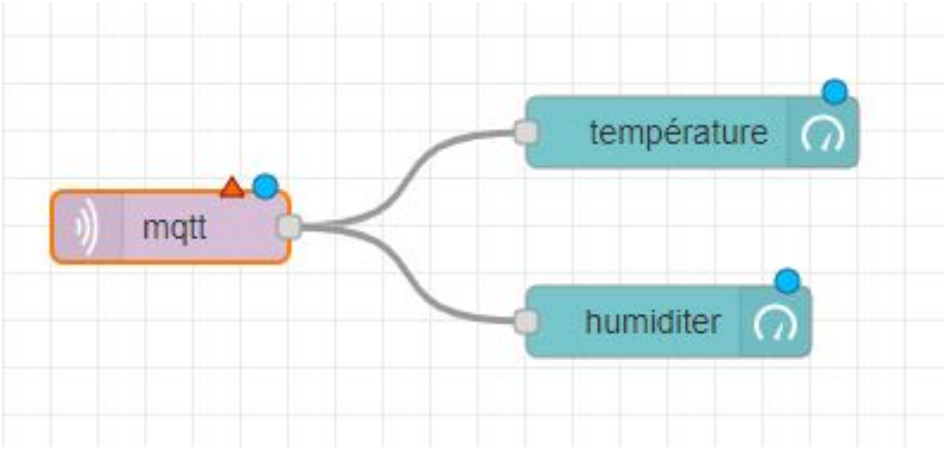


Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Une fois les groupes créés nous pouvons utiliser les nodes graphiques de dashboard.

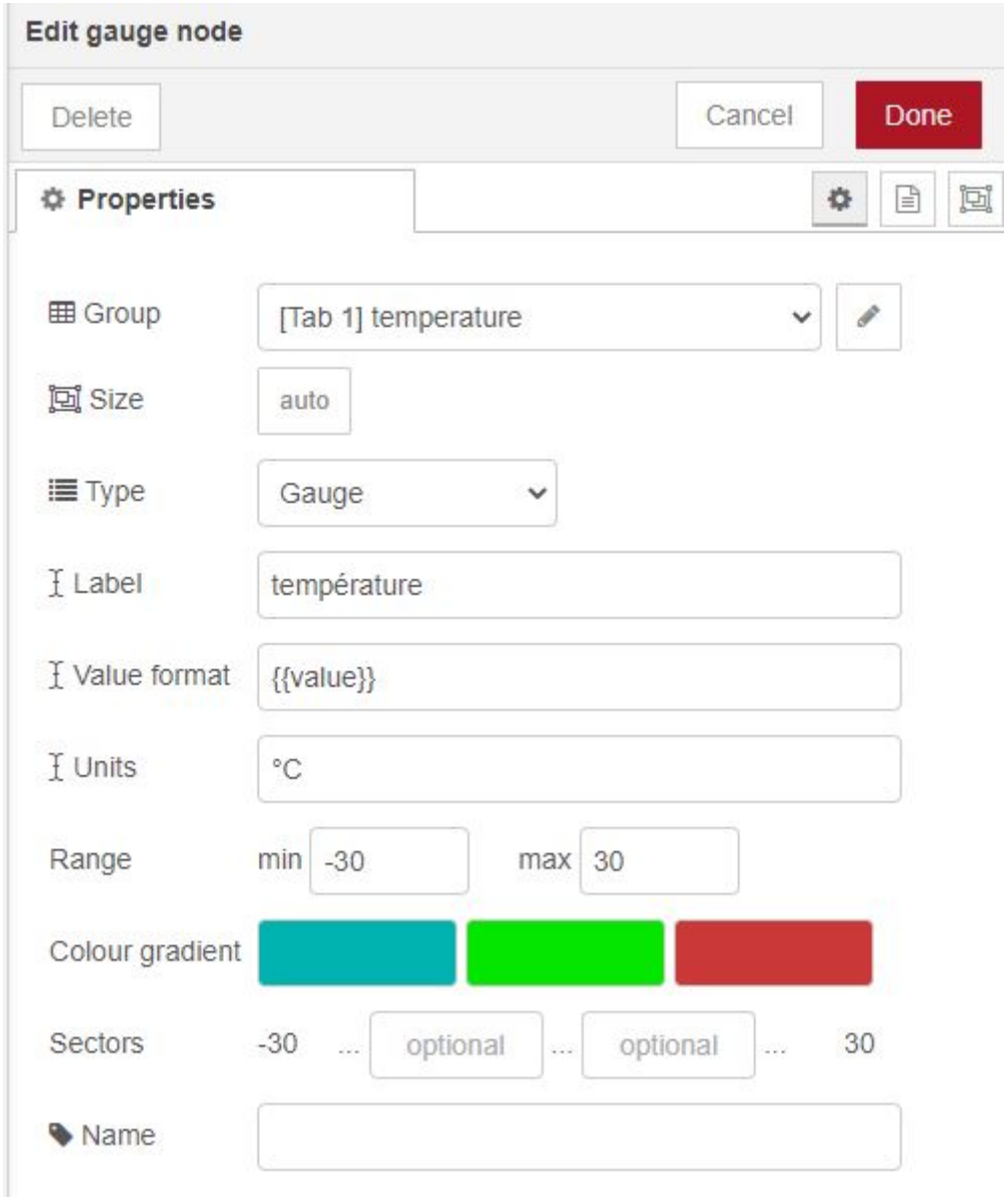
Dans notre cas les node seront utiliser dans cette configuration, bien que actuellement juste la température et humidité sont configurés.



Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Nous allons donc configurer la node température, pour cela double cliquer dessus et un menu apparaît.



Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Group: Assigner la node a un groupe particulier dans notre cas le groupe température.

Size: Taille du graphique.

Type: Type de jauge voulue.

Label: Nom de la node.

Value format: Format de l'unité souhaitée.

Units: Unité de la valeur.

Range: Portée minimale et maximale du graphe.

Coulour gradient: Selection de couleur indiquant un certain seuil.

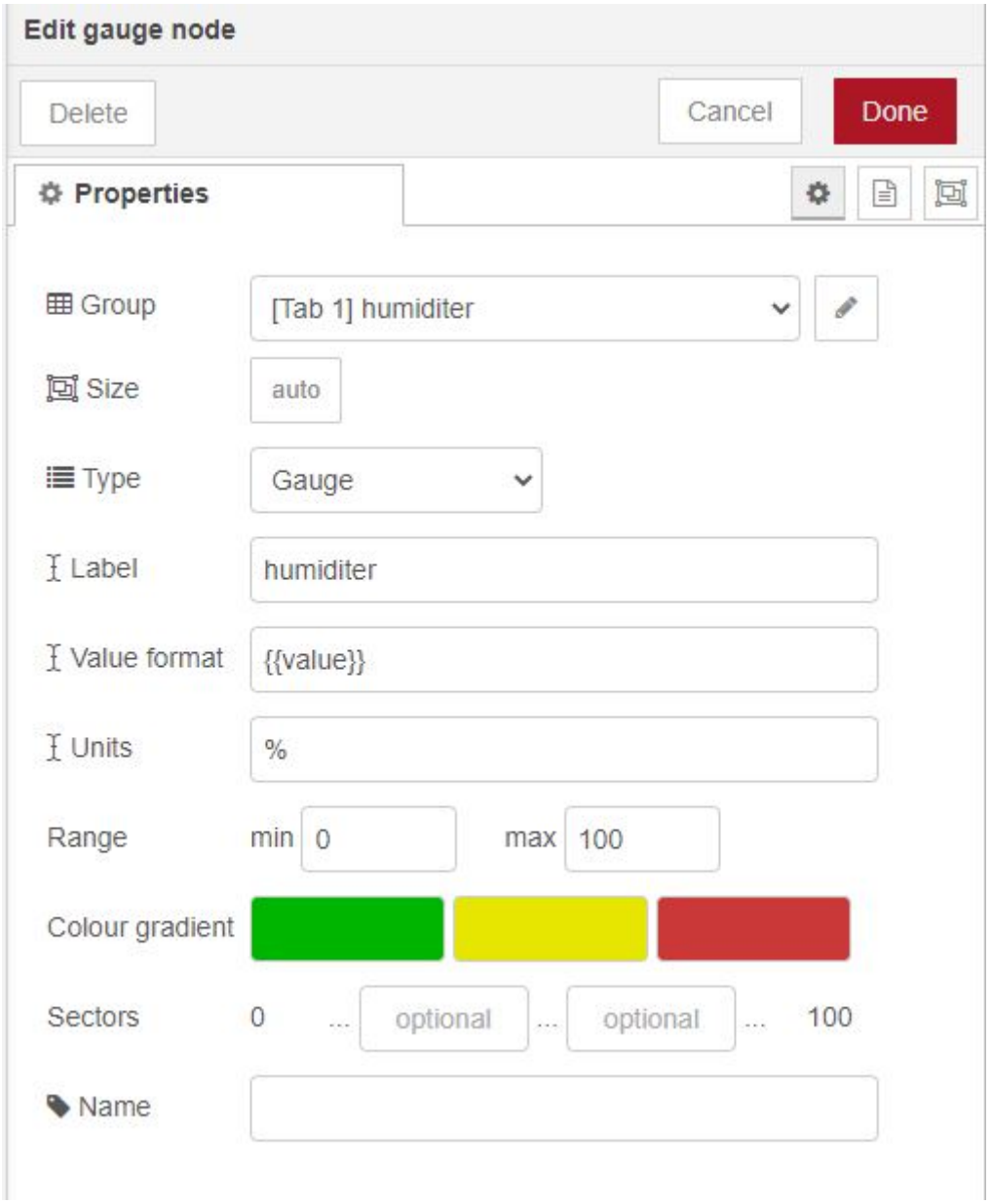
Sector: Secteur de définition des seuil.

Name: Nom du graph.

Une fois la configuration établie nous allons répéter le même procédé pour l'humidité.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique



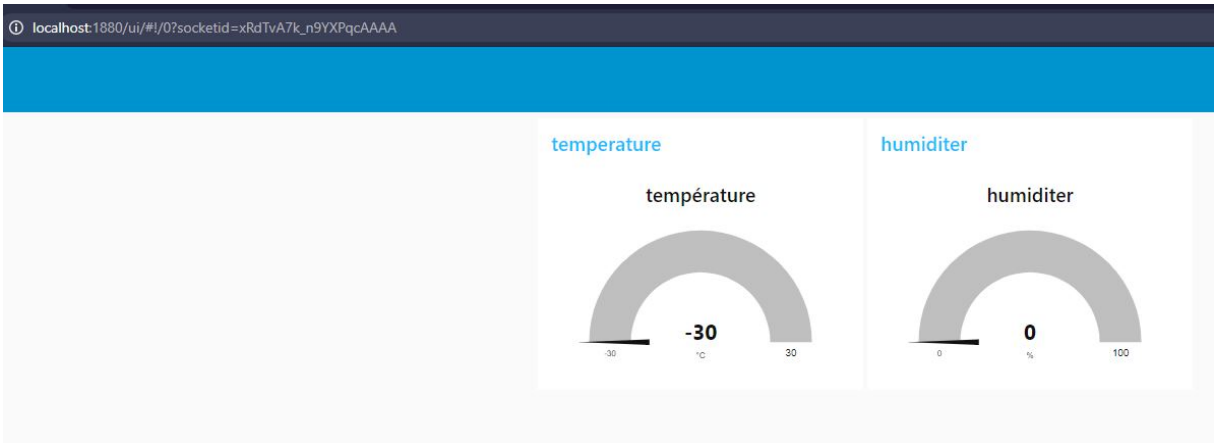
Une fois cela fait le graphe devient visible lors des test (pensez à cliquer sur deploy pour sauvegarder).

Pour vérifier tout cela taper dans la barre de recherche de votre navigateur localhost1880/ui.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

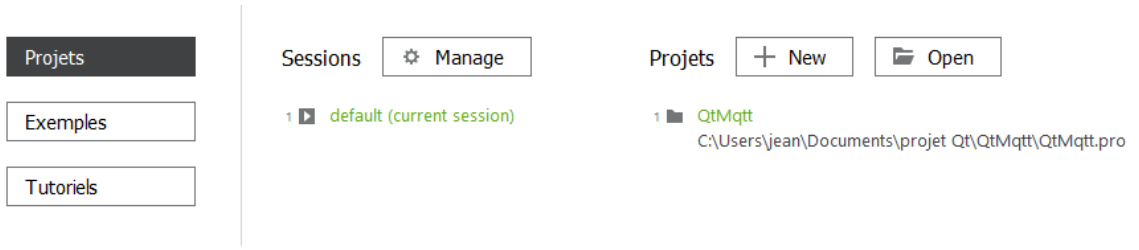
Cela ouvrira une page web contenant vos nodes graphiques.



Une fois cela fait, nous allons nous pencher sur notre interface graphique sur Qt creator.

Nous allons donc créer un nouveau projet Qt (nous utiliserons la version 5.14.2 de Qt).

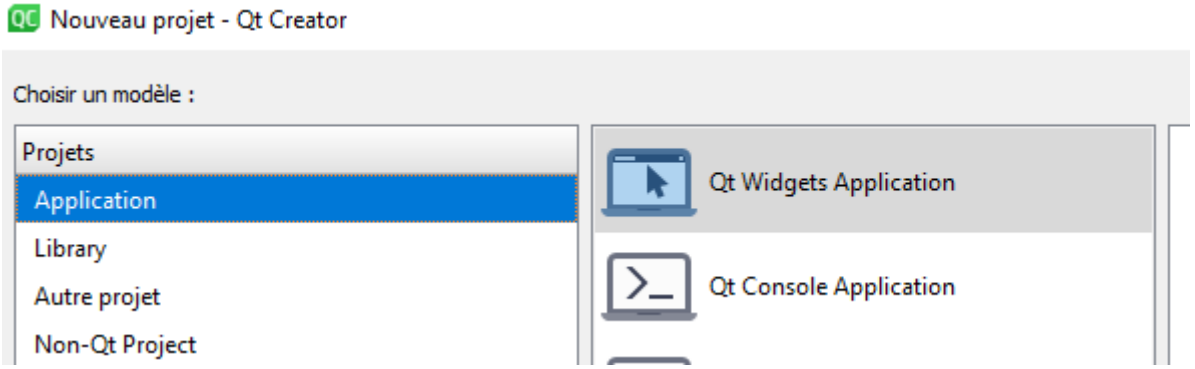
Cliquer sur new:



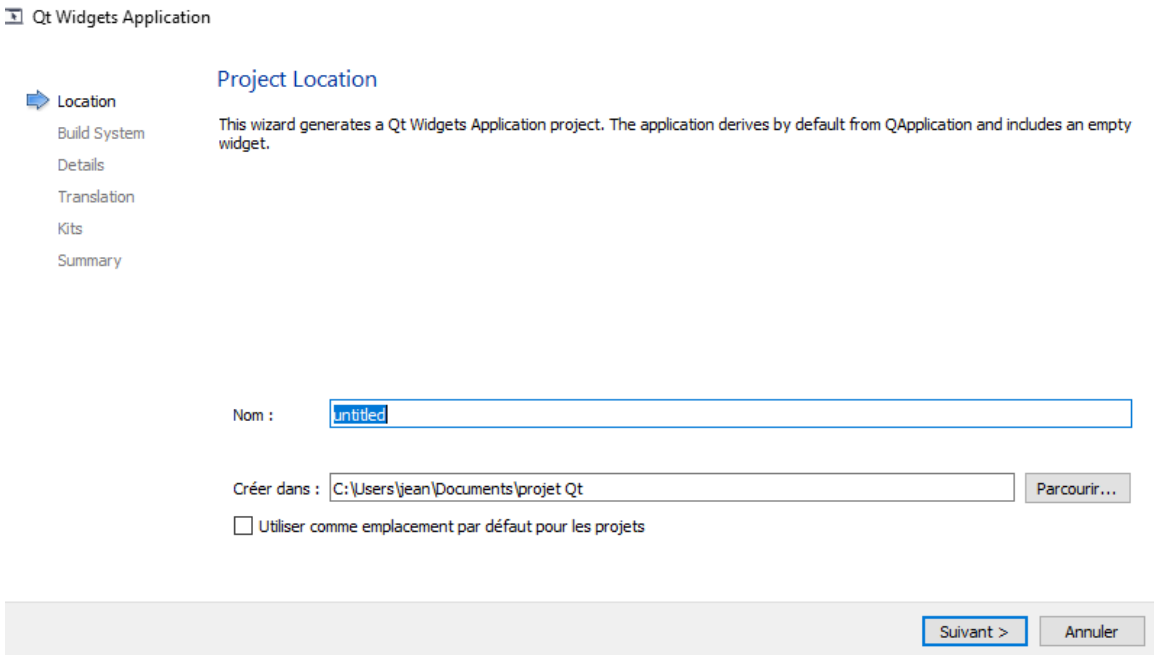
Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Créer une application Qt en cliquant sur Qt Widgets application.



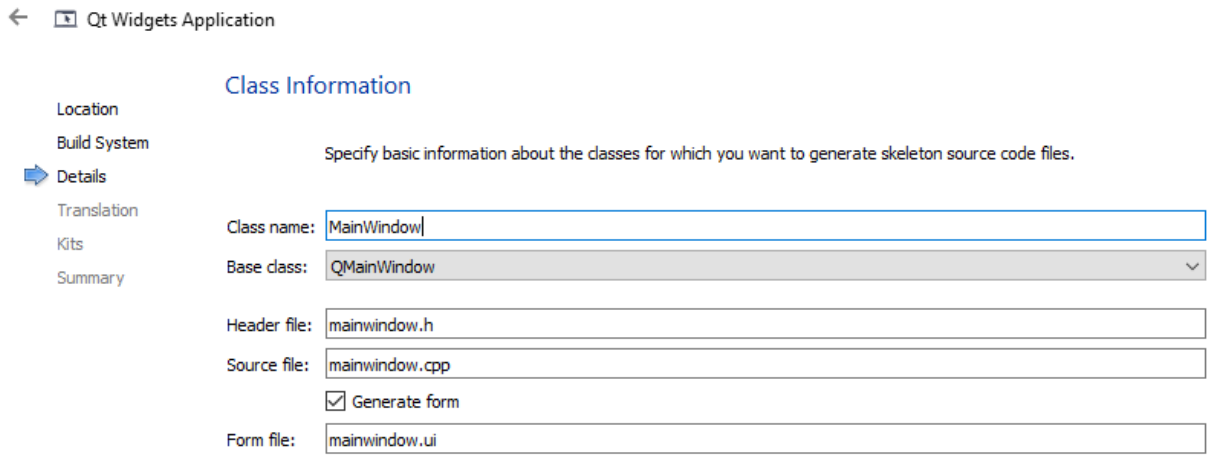
Ensuite il faudra nommer le projet, dans notre cas le projet se nommera QtMqtt.



On prendra le build system qmake, puis après ça nous devons nommer nos fichier, dans notre cas nous laisserons les noms de base.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique



Après ça, ignorer “translation file” et cliquez sur suivant, faite de même pour “kit sélection” et une fois à “project management” cliquer sur “terminer”.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>


/****************** Connexion a mqtt *************************/

MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  m_client = new QMqttClient(this);
  m_client->setHostname("127.0.0.1");
  m_client->setPort(1883);

  connect(m_client, &QMqttClient::stateChanged, this, &MainWindow::stateChange); //a
comprendre
}


MainWindow::~MainWindow()
{
  delete ui;
}
```

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>


/****************** Connexion a mqtt ************************/

MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  m_client = new QMqttClient(this);
  m_client->setHostname("127.0.0.1");
  m_client->setPort(1883);

  connect(m_client, &QMqttClient::stateChanged, this, &MainWindow::stateChange); //a
comprendre
}


MainWindow::~MainWindow()
{
  delete ui;
}
```

Cette partie du code permet de se connecter à mqtt en indiquant les paramètres du serveur (qui dans notre cas est le Docker).
Il y a aussi un destructeur pour cette classe.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

```
void MainWindow::on_pbConnexion_clicked()  // action se déroulant au moment ou on appuis sur le bouton connexion
{
  if (m_client->state() == QMqttClient::Disconnected) {
        ui->pbConnexion->setText(tr("Deconnexion"));
        m_client->connectToHost();

    } else {
        ui->pbConnexion->setText(tr("Connexion"));
        m_client->disconnectFromHost();
    }
} //fin pbConnexion


void MainWindow::stateChange() {   //state change est une déclaration de <QMqttClient>, il nous informeras de l'état de notre demande

  QString message;
     switch (m_client->state())
     {
        case 0 :message = "Déconnecté";break;
        case 1 :message = "En cours de connexion";break;
        case 2 :message = "Connecté";
        QString myTopic = "test";        //test est le nom du topics ou les données sont envoyer et reçue donc quand on s'abonne a test on reçoit toute les données

        auto subscription = m_client->subscribe(myTopic); // s'abonne au topic pour recevoir les valeurs
                if (!subscription) {
                   QMessageBox::critical(this, "Erreur", "Impossible de souscrire au topic\n"+myTopic);
                   return;
                }
     }
} //fin stateChange

/****************** FIN Connexion a mqtt ***************************/
```

Cette partie du code indique qu'au moment où le bouton connexion est enclenché, le programme se connecte au serveur et nous affiche un rendu de l'état de la connexion ou d'un échec si impossible.

Ceci marque la fin de la connexion par mqtt.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

```
/****************** Reception des messages ***************************/

void MainWindow::receivedMessage(const QByteArray &message, const QMqttTopicName &topic)
//affiche les valeur reçue et convertie par la passerelle.
{
  QString unit;
  If (topic.name()=="test")
  {
     unit = " °C";
     ui->temp->setText(message+unit);    //affiche la valeur de température en °C
  }
  If (topic.name()=="test")
  {
     unit = " %";
     ui->hum->setText(message+unit);     //affiche la valeur d'humidité en %
  }

  {
     unit = " °C";
     ui->pr->setText(message+unit);     //affiche la valeur du point de rosée en °C
  }

  {
     unit = " %";

 ui->batt->setText(message+unit);    //affiche la valeur de la batterie en %
  }

  {
     ui->smoke->setText(message+unit);   //affiche la valeur de la fumée
  }
}// fin receivedMessage

/******************FIN Reception des messages ***************************/
```

Cette partie du code sert à recevoir les données du serveur, ce qui fait que pour chaque valeur demandée nous avons une unité et les valeurs pour chaque capteur.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

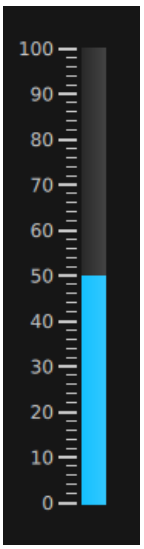
```
import QtQuick 2.0
import QtQuick.Extras 1.4
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4

tempGauge {   //type circular gauge
    maximumValue : 60
   minimumValue: -30
   anchors.left: parent
// donner saisie automatique grâce à réception dans mainwindows.cpp
}

humidité {   //type circular gauge
    maximumValue : 100
   minimumValue: 0
   anchors.centerIn: parent
// donner saisie automatique grâce à réception dans mainwindows.cpp
}

Gauge {
   minimumValue: 0
   //value: donner saisie automatique grâce à réception dans mainwindows.cpp
   maximumValue: 100
    anchors.right: parent.right
}
```

Dans cette partie du code nous avons le “.qml”, qui sert à créer nos interface graphique donnant ce genre de résultat:



Partie IR 2 : JEAN Romain

➢ Visualisation graphique

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QMqttClient>
#include <QMessageBox>
#include <QObject>
#include <QQuickItem>



QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();


private slots:
  void on_pbConnexion_clicked();
  void stateChange();
  void brokerDisconnected();
  void receivedMessage(const QByteArray &message, const QMqttTopicName &topic);

private:
  Ui::MainWindow *ui;
  QMqttClient *m_client;
};
#endif // MAINWINDOW_H

class tempGauge:
```

Cette partie du code concerne la déclaration des classes présente dans le .cpp et le “.qml”.

Il sert à définir le fonctionnement des classes, par exemple la classe mainwindows instaure l’action du clic sur le bouton connexion, l'état de changement et la déconnexion du broker ainsi que la réception des messages.

Partie IR 2 : JEAN Romain

➢ Visualisation graphique

Manquant de temps la suite n'est que théorique mais pour avoir l'interface graphique il faut simplement initialiser les classe du .qml (tempGauge, humidité et gauge) dans le .h pour faire fonctionner l'interface graphique et récupérer automatiquement les valeurs.

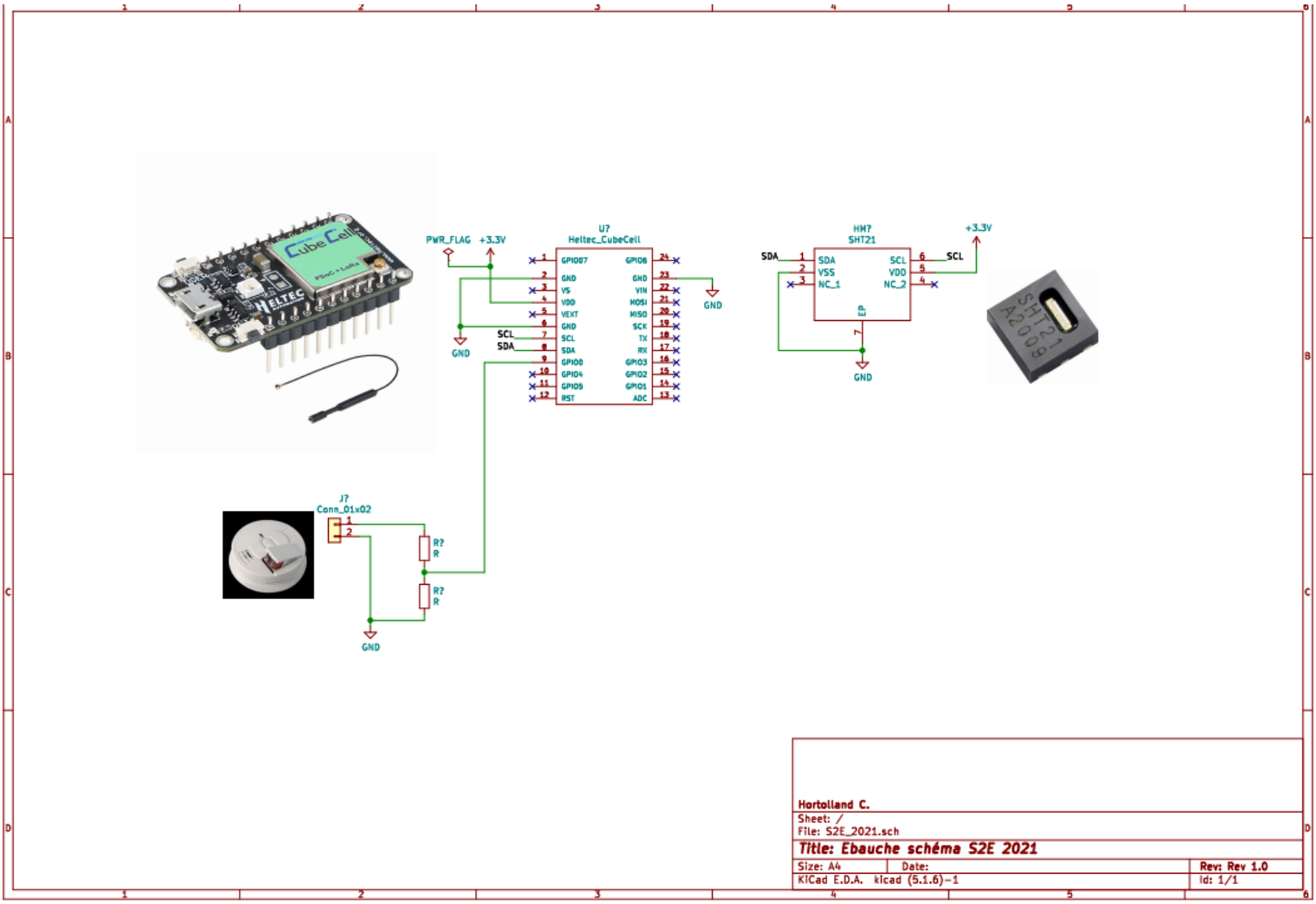
Partie EC1:

DELORME Valentin

Partie EC1 : Delorme Valentin

L’objectif est de créer une carte électronique ainsi qu’un boîtier adapté en utilisant conjointement une carte CubeCell un détecteur thermos-hydrométrique ainsi qu’un détecteur de fumée affin de communiqué les données relatives à l’humidité, la chaleur et la possibilité d’incendie au système de «surveillance d’environnement pour entrepôt».

Les valeurs doivent être régulièrement envoyée à un docker en LoRa. Mr.Bijou souhaite que les cartes aient la plus longue autonomie possible.



(Schéma de principe, comprenant un capteur SHT21, un détecteur de fumée et une carte CubeCell.)

Partie EC1 : Delorme Valentin

➢ Choix/Détail des détecteurs de fumée

Prix Unité en euros	5,95
Complexité du circuit	Simple
Vitesse de déclenchement	moyen

[Lifedom (20KF01) **]** *Sélectionné*

Prix Unité en euros	15
Complexité du circuit	moyen
Vitesse de déclenchement	Lente

[X-SENSE (SD13) **]**

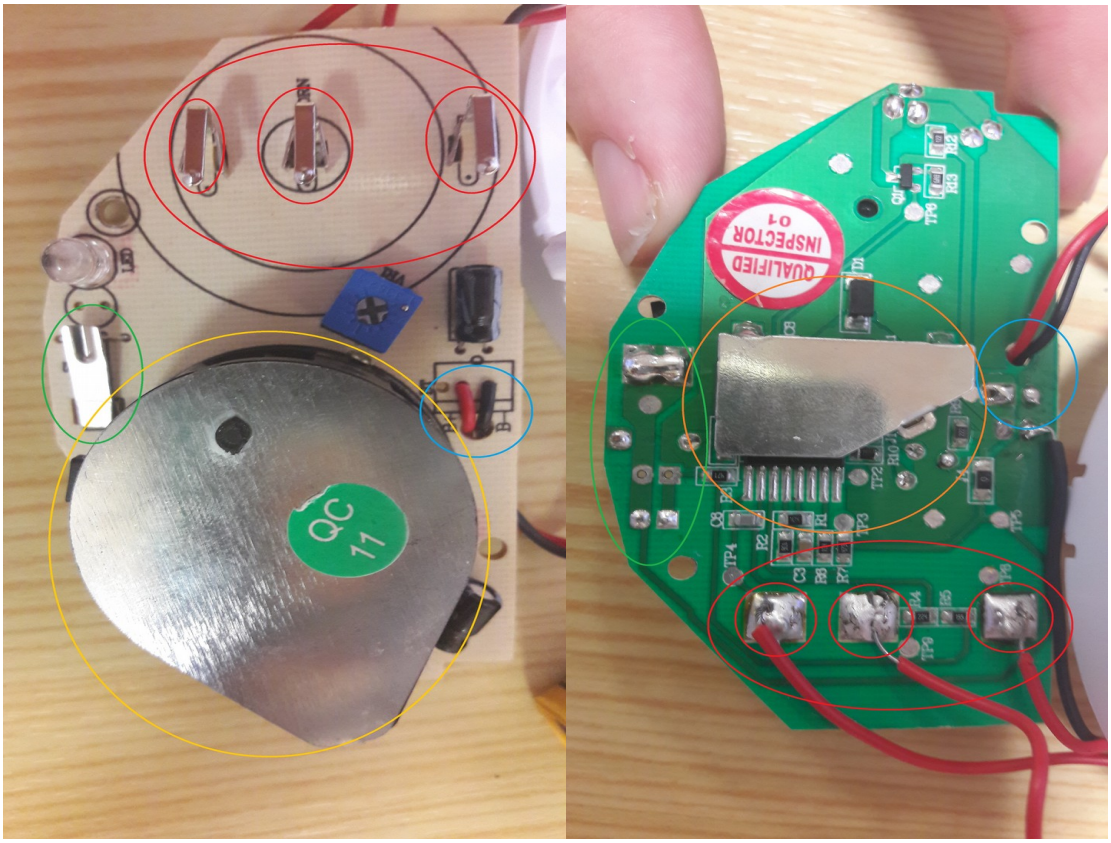
Prix Unité en euros	6,45
Complexité du circuit	Élevée
Vitesse de déclenchement	Rapide

[Angel-Eye (GN2465/R2) **]**

Partie EC1 : Delorme Valentin

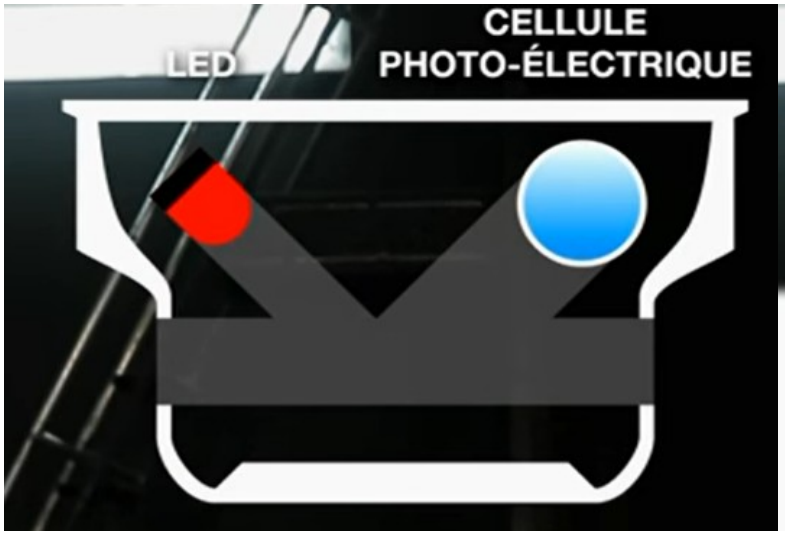
➢ *[Lifedom (20KF01)]*

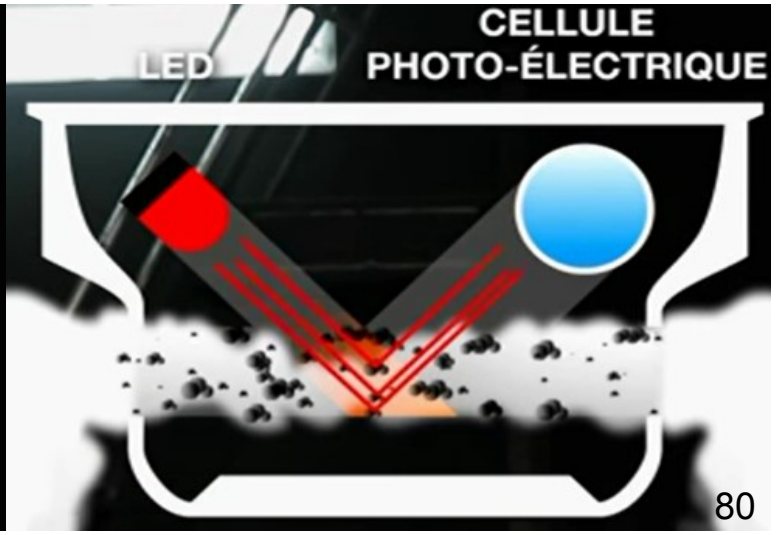
Le système du Lifedom fonctionne grâce à l'association d'un capteur de fumée optique et d'un circuit (le tout alimenté par une pile 9V) qui déploie une tension alternative sur un ensemble de 3 pattes métallique vibrant contre une cymbale, provoquant un bruit strident chaque fois que la zone est enfumée à proximité.



Rouge : Pattes vibrant contre un cymbale à sa fréquence de résonance (Sonnerie), les pattes sont alimentés en 9 V alternatif
Jaune : Détecteur optique de fumée et son circuit
Bleue : Alimentation, pile 9 V

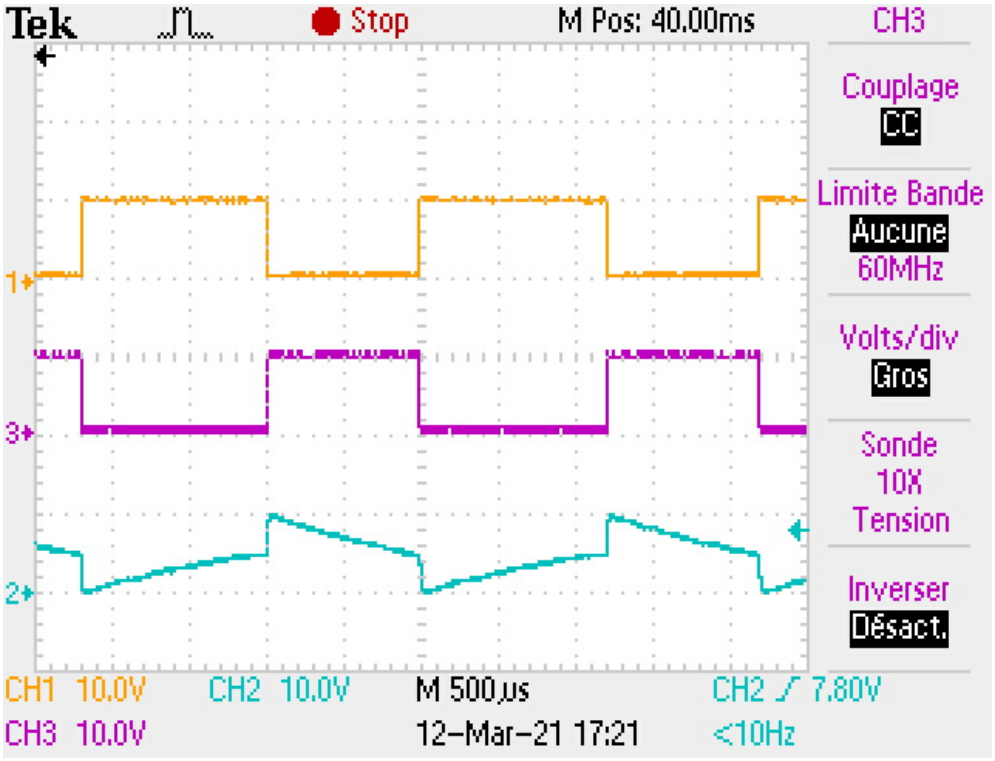
Vert : Bouton test, basiquement un bouton poussoir



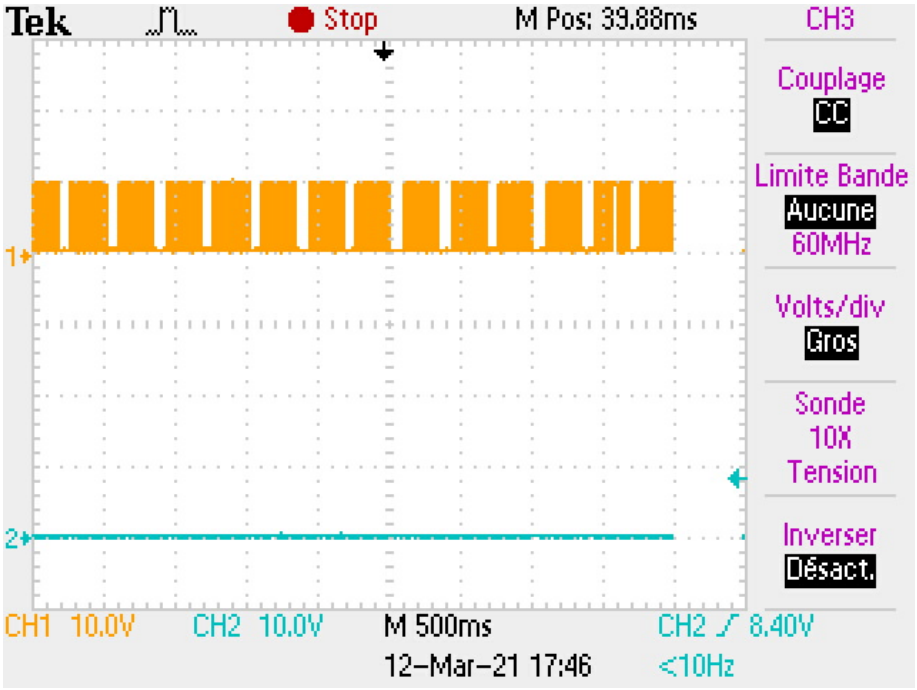


Partie EC1 : Delorme Valentin

➢ ***[Lifedom Signaux pattes I]***



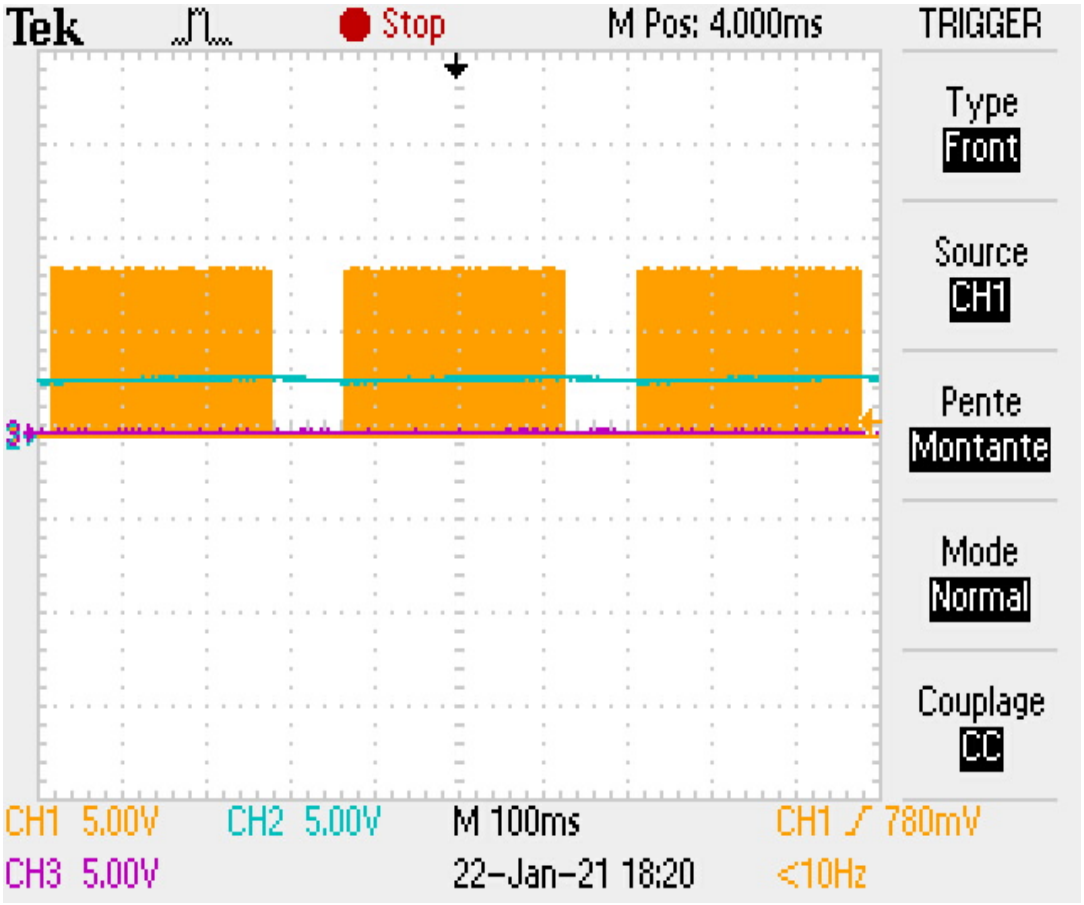
L'ensemble des signaux de chaque pattes, ici les deux pattes périphérique reçoivent un signal carré en opposition de phase, la patte centrale semble recevoir un signal à mi-chemin entre carré et sinusoïdal, ce signal n'apparaît cependant pas lorsqu'il est pris en solitaire, on peut donc en conclure qu'il la synthèse des deux autres.



Le signal d'une des pattes périphérique pris à une échelle de temps bien supérieur, on comprend qu'il s'agit de salve, la capture suivante montrera que chaque salve dure 230 ms avec un intervalle de 80 ms, soit 310 ms du début à la fin.

Partie EC1 : Delorme Valentin

➢ ***[Lifedom Signaux pattes II]***



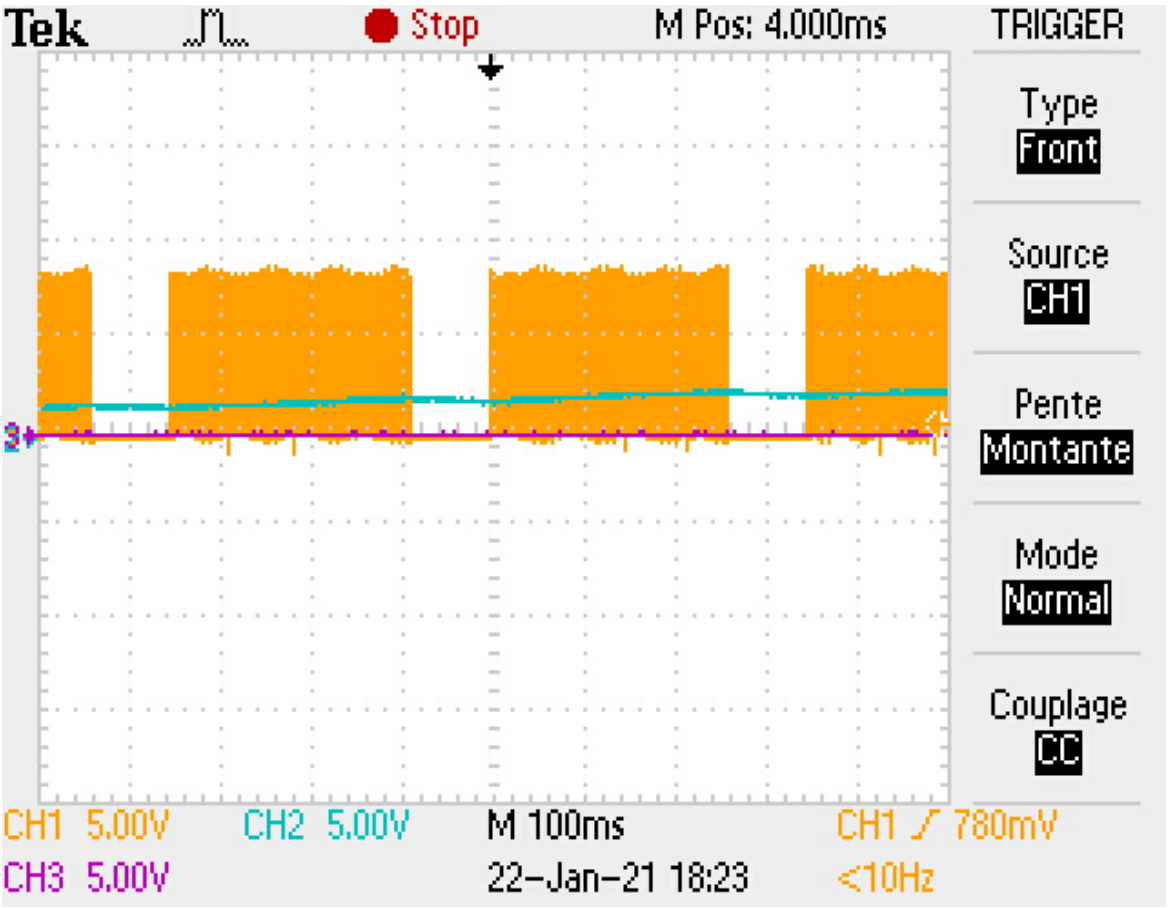
La carte prévue pour recevoir le signal est une CubeCell (3,3V), ils faut donc un signal continue, supérieur à 2 V en pin I/O pour obtenir un état haut (Vcc*0,6), les premiers tests ont été fait en créant un filtre passe-bas analogique à partir d'une résistance de 100K ohms et d'un condensateur de 10 micro farad, ce montage sera modifié car n'envoyant qu'un signal à peine détectable et fût plus tard remplacé pour une résistance de 1M ohms pour 1 micro farad.

$$f_c = \frac{1}{2\pi RC} \text{ ou } \omega_c = \frac{1}{RC}$$

En filtrant les fréquence supérieur à un 0,15 Hz on obtiens un signal quasi-continu.

Partie EC1 : Delorme Valentin

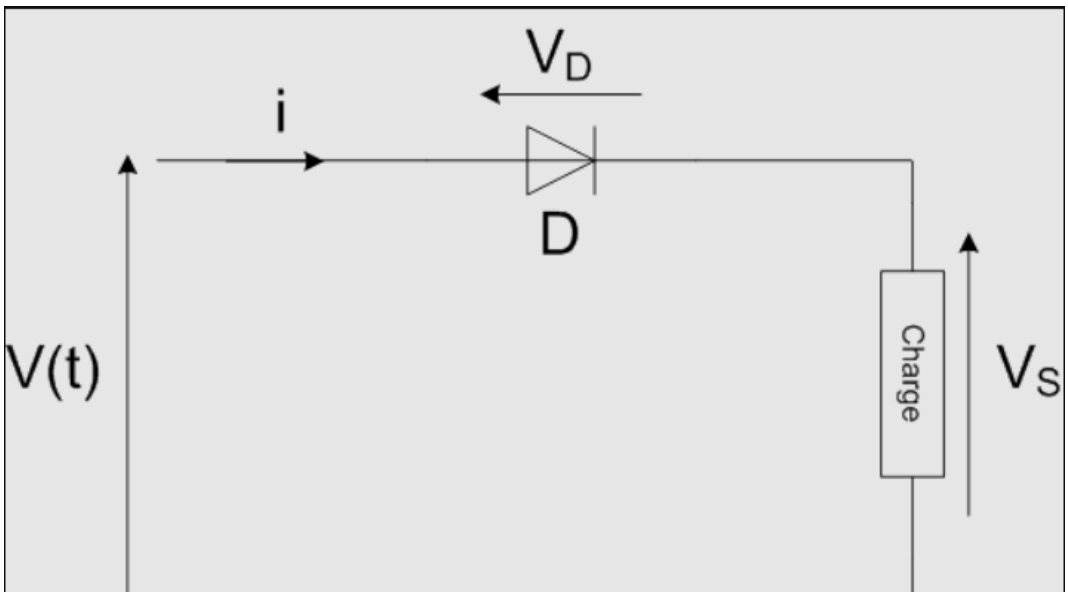
➢ **[Lifedom Signaux pattes III]**



Le montage filtre passe-bas ne sera finalement pas retenue du fait qu'avec le capot surviens une perte de tension l'amenant à un 2V fluctuant légèrement, 2V étant le seuil inférieur de l'état haut et le seuil supérieur d'un état indéterminé.

De plus la charge du condensateur est particulièrement longue.

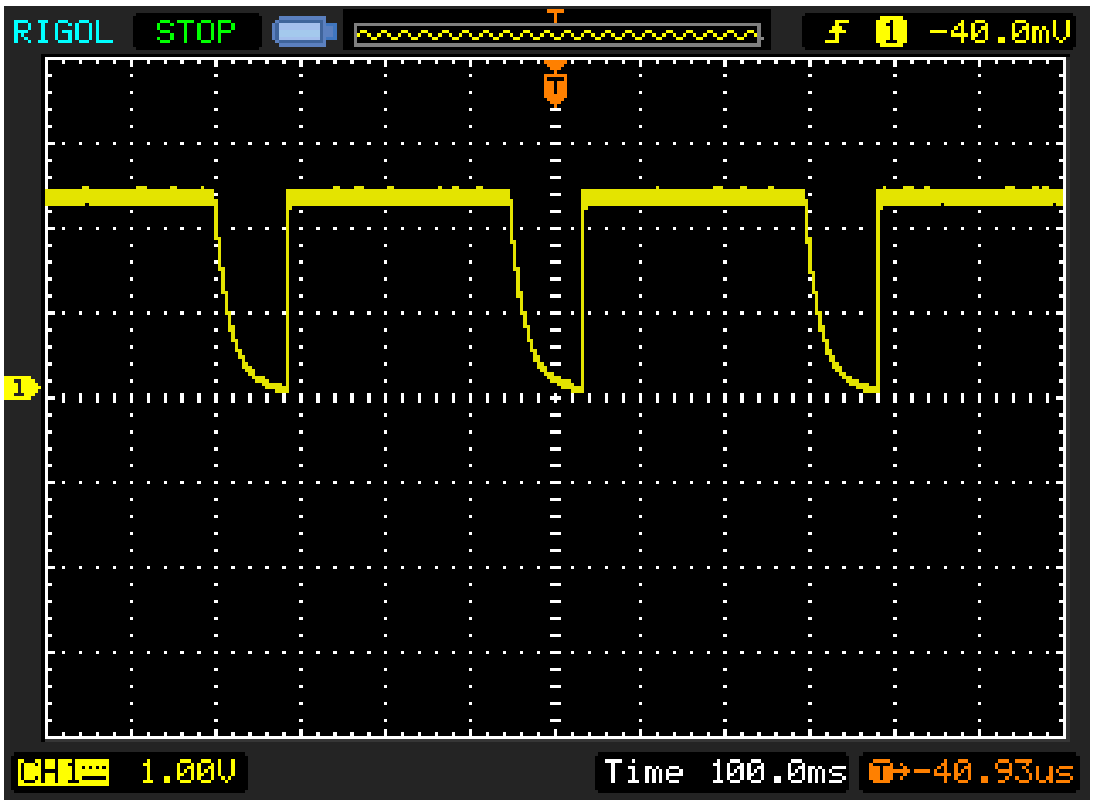
En remplacement du montage filtre passe-bas, un montage basée sur un redresseur et un pont diviseur de tension seront utilisés à la place.



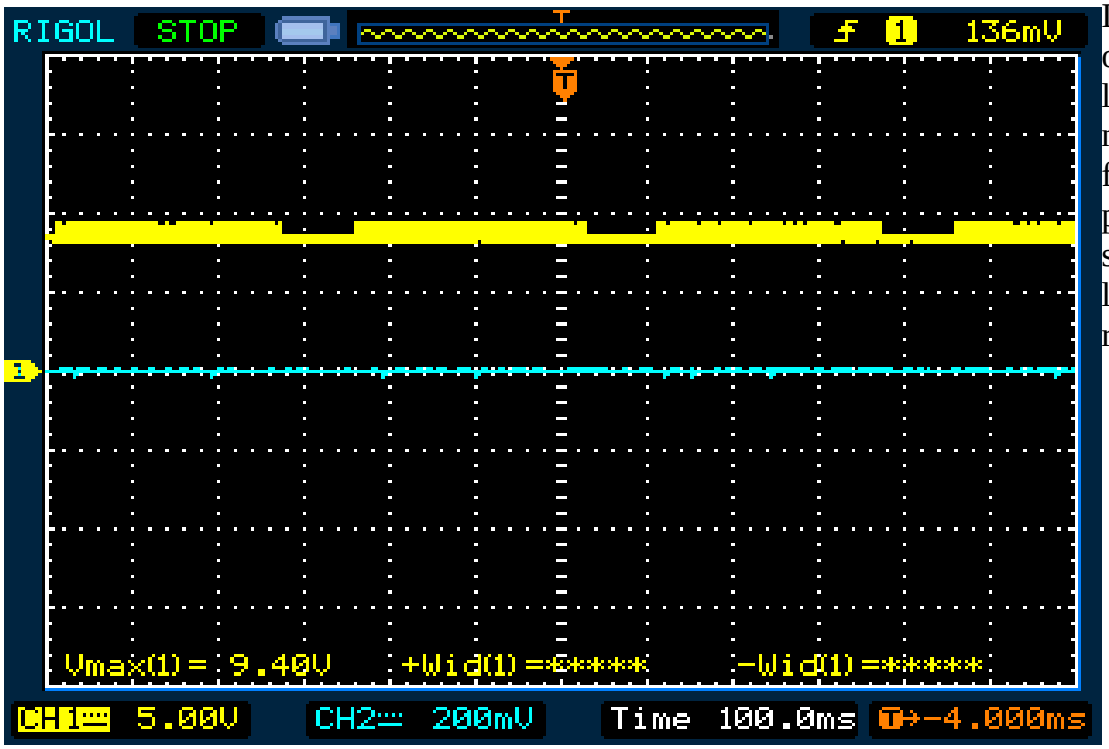
Redresseur de tension

Partie EC1 : Delorme Valentin

➢ [Lifedom Signaux pattes IV]



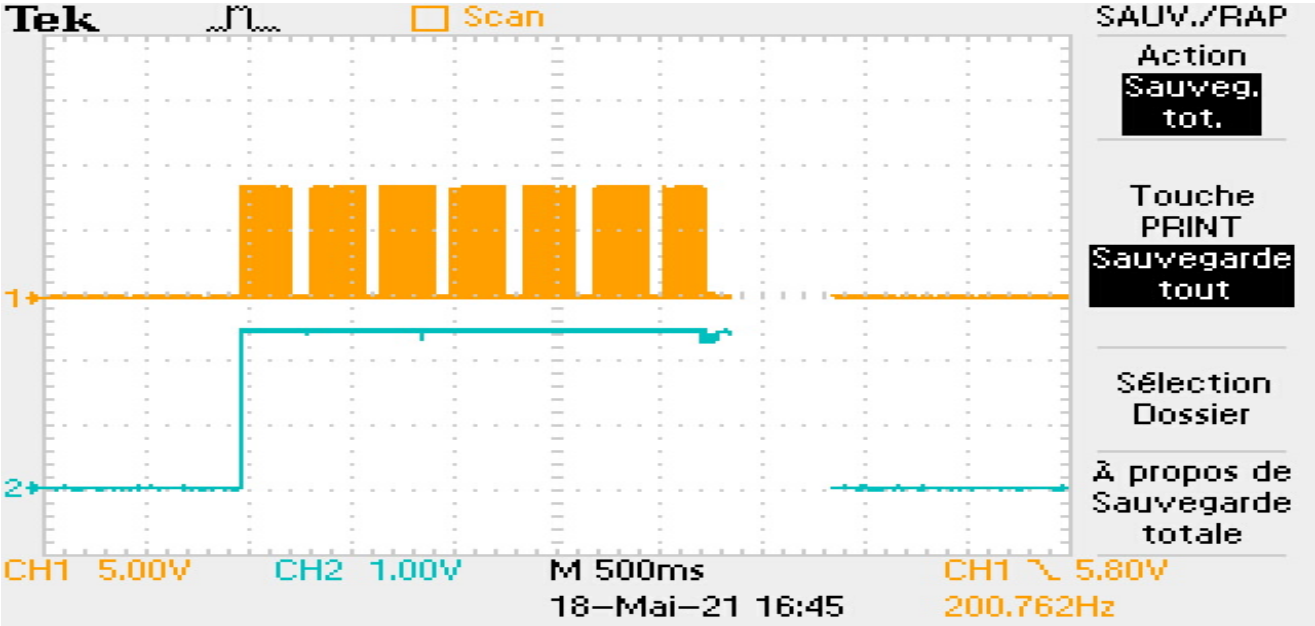
Le montage utilisant le pont diviseur de tension et le redresseur de tension s'avéra ne pas être viable, même si on arrive à ne saisir que l'enveloppe du signal la partie IR a manifester la volonté de n'avoir qu'un seul front ascendant suivie d'un signal en continue. Pour répondre à cette demande un composant monostable déclenchable, le NE555 a été sélectionné.



Le signal obtenu avec le NE555 n'était finalement pas satisfaisant, le CD4538 le remplacera

Partie EC1 : Delorme Valentin

➢ [Lifedom Signaux pattes V]



Le CD4538 fonctionne correctement, les salves en alternatives sont «convertie» en continue, un état haut permanent en quelque sorte.

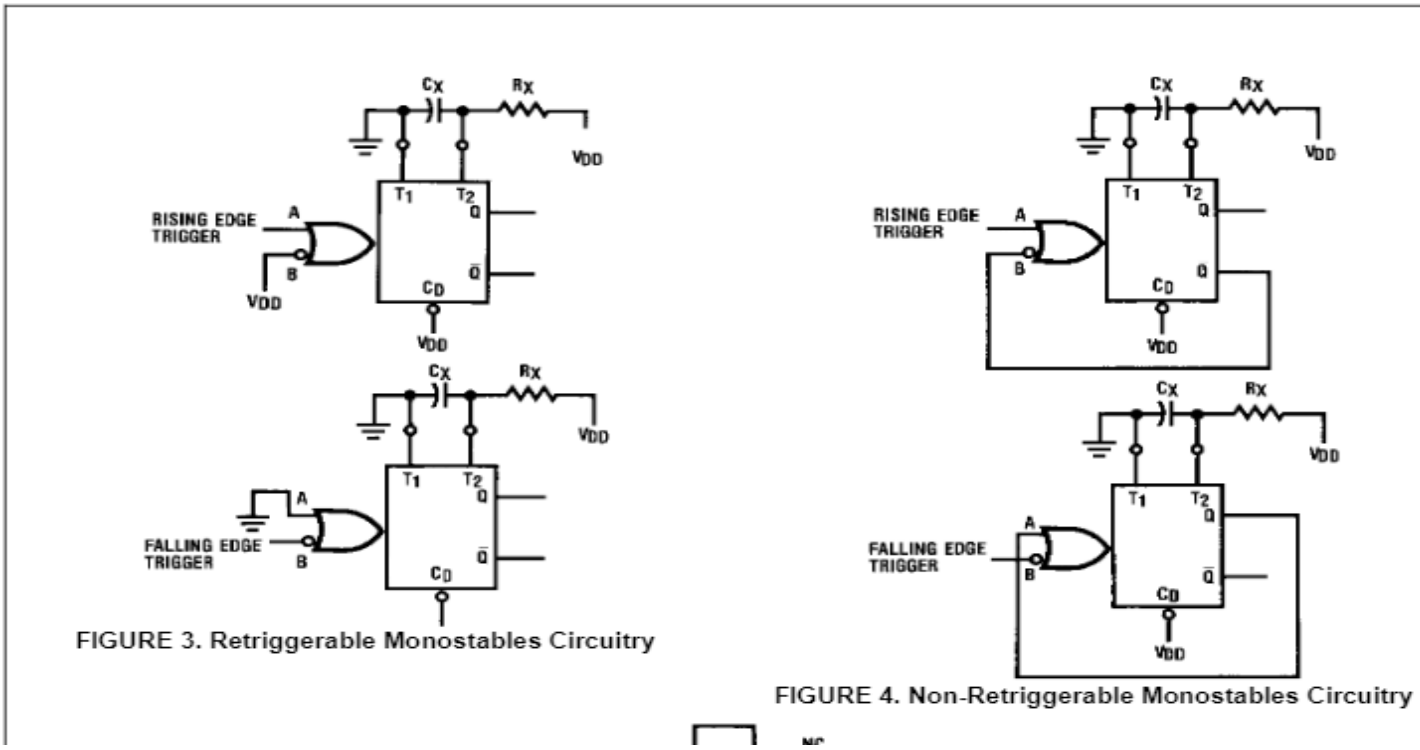


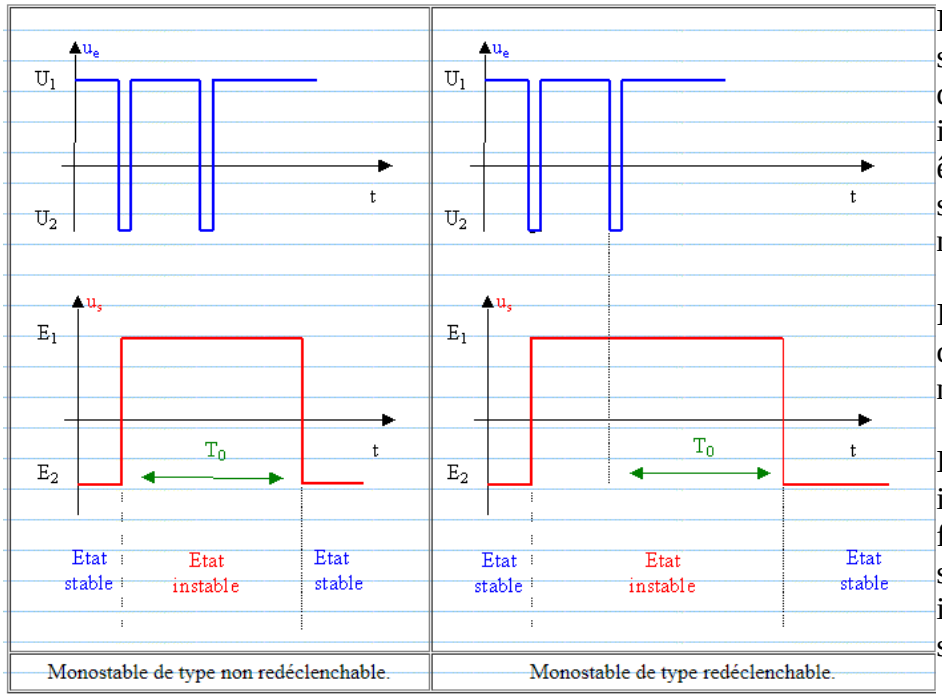
FIGURE 3. Retriggerable Monostables Circuitry

FIGURE 4. Non-Retriggerable Monostables Circuitry

(Le CD4538 en re-déclenchable et non re-déclenchable sur front montant et front descendant)

Partie EC1 : Delorme Valentin

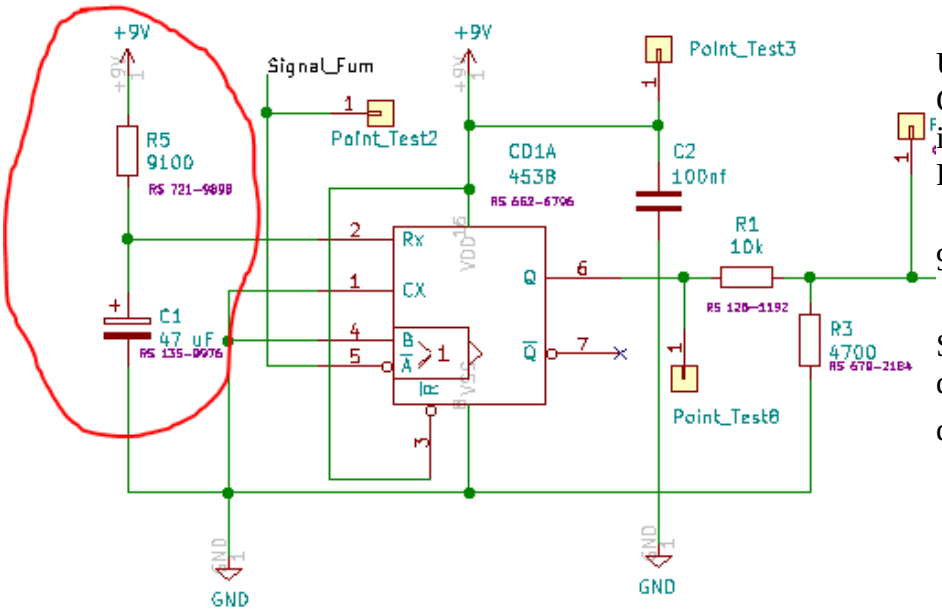
➢ [Monostable CD4538]



Les monostables sont des structures possédant deux états, dit état stable et instable, l'état instable est temporaire et peut être provoquer par une stimulation, sa durée dépend du montage.

Ils peuvent être divisé en deux catégories, re-déclenchable et non re-déclenchable.

En non re-déclenchable, l'état instable ne se déclenche qu'une fois avant de revenir à un état stable. En re-déclenchable, l'état instable se déclenche à chaque stimulation.



Un extrait du schéma utilisant le CD4538, la durée de son état instable est le produit direct de R5 et C1.

9100x0.000047 = 0,4277

Soit environ 0,4 secondes, la durée totale de chaque salves est de 0,31 secondes.

Partie EC1 : Delorme Valentin

➢ [Monostable CD4538 – Caractéristiques électriques]

Absolute Maximum Ratings (Notes 1 and 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

DC Supply Voltage (V_{DD})	−0.5 to +18 V_{DC}
Input Voltage (V_{IN})	−0.5V to V_{DD} + 0.5 V_{DC}
Storage Temperature Range (T_S)	−65°C to +150°C
Power Dissipation (P_D)	
Dual-In-Line	700 mW
Small Outline	500 mW
Lead Temperature (T_L) (Soldering, 10 seconds)	260°C

Recommended Operating Conditions (Note 2)

DC Supply Voltage (V_{DD})	3 to 15 V_{DC}
Input Voltage (V_{IN})	0 to V_{DD} V_{DC}
Operating Temperature Range (T_A)	
CD4538BM	−55°C to +125°C
CD4538BC	−40°C to +85°C

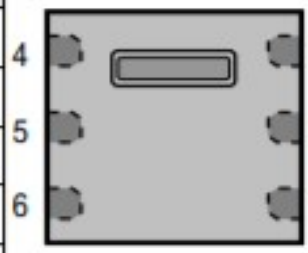
DC Electrical Characteristics CD4538BM (Note 2)

Symbol	Parameter	Conditions	−55°C		+25°C			+125°C		Units
			Min	Max	Min	Typ	Max	Min	Max	
I_{DD}	Quiescent Device Current	V_{DD} = 5V, V_{IH} = V_{DD}, V_{IL} = V_{SS}, All Outputs Open		5		0.005	5		150	μA
		V_{DD} = 10V, V_{IH} = V_{DD}, V_{IL} = V_{SS}, All Outputs Open		10		0.010	10		300	μA
		V_{DD} = 15V, V_{IH} = V_{DD}, V_{IL} = V_{SS}, All Outputs Open		20		0.015	20		600	μA
V_{OL}	Low Level Output Voltage	V_{DD} = 5V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$		0.05		0	0.05		0.05	V
		V_{DD} = 10V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$		0.05		0	0.05		0.05	V
		V_{DD} = 15V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$		0.05		0	0.05		0.05	V
V_{OH}	High Level Output Voltage	V_{DD} = 5V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	4.95		4.95	5		4.95		V
		V_{DD} = 10V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	9.95		9.95	10		9.95		V
		V_{DD} = 15V, $\|I_O\| < 1\ \mu A$, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	14.95		14.95	15		14.95		V
V_{IL}	Low Level Input Voltage	$\|I_O\| < 1\ \mu A$, V_{DD} = 5V, V_O = 0.5V or 4.5V		1.5		2.25	1.5		1.5	V
		$\|I_O\| < 1\ \mu A$, V_{DD} = 10V, V_O = 1.0V or 9.0V		3.0		4.50	3.0		3.0	V
		$\|I_O\| < 1\ \mu A$, V_{DD} = 15V, V_O = 1.5V or 13.5V		4.0		6.75	4.0		4.0	V
V_{IH}	High Level Input Voltage	$\|I_O\| < 1\ \mu A$, V_{DD} = 5V, V_O = 0.5V or 4.5V	3.5		3.5	2.75		3.5		V
		$\|I_O\| < 1\ \mu A$, V_{DD} = 10V, V_O = 1.0V or 9.0V	7.0		7.0	5.50		7.0		V
		$\|I_O\| < 1\ \mu A$, V_{DD} = 15V, V_O = 1.5V or 13.5V	11.0		11.0	8.25		11.0		V
I_{OL}	Low Level Output Current (Note 3)	V_{DD} = 5V, V_O = 0.4V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	0.64		0.51	0.88		0.36		mA
		V_{DD} = 10V, V_O = 0.5V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	1.6		1.3	2.25		0.9		mA
		V_D = 15V, V_O = 1.5V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	4.2		3.4	8.8		2.4		mA
I_{OH}	High Level Output Current (Note 3)	V_{DD} = 5V, V_O = 4.6V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	−0.64		−0.51	−0.88		−0.36		mA
		V_{DD} = 10V, V_O = 9.5V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	−1.6		−1.3	−2.25		−0.9		mA
		V_D = 15V, V_O = 13.5V, $V_{IH} = V_{DD}$, $V_{IL} = V_{SS}$	−4.2		−3.4	−8.8		−2.4		mA
I_{IN}	Input Current, Pin 2 or 14	V_{DD} = 15V, V_{IN} = 0V or 15V		±0.02		$\pm 10^{-5}$	±0.05		±0.5	μA
I_{IN}	Input Current Other Inputs	V_{DD} = 15V, V_{IN} = 0V or 15V		±0.1		$\pm 10^{-5}$	±0.1		±1.0	μA

Partie EC1 : Delorme Valentin

➢ SHT21 : Caractéristiques

Pin	Name	Comment
1	SDA	Serial Data, bidirectional
2	VSS	Ground
5	VDD	Supply Voltage
6	SCL	Serial Clock, bidirectional
3,4	NC	Not Connected



Le SHT21 est capteur de température et d'humidité à six broches mesurant 3mm par 3mm, qui fonctionne en I2C et utilise la technologie CMOS. Sa résolution peut varier de 8/12 bits à 12/14 pour l'humidité relative et la température.

Relative Humidity

Parameter	Condition	min	typ	max	Units
Resolution [1]	12 bit		0.04		%RH
	8 bit		0.7		%RH
Accuracy tolerance [2]	typ		±2.0		%RH
	max	see Figure 2			%RH
Repeatability			±0.1		%RH
Hysteresis			±1		%RH
Nonlinearity			<0.1		%RH
Response time [3]	τ 63%		8		s
Operating Range	extended [4]	0		100	%RH
Long Term Drift [5]	normal		< 0.5		%RH/yr

Temperature

Parameter	Condition	min	typ	max	Units
Resolution [1]	14 bit		0.01		°C
	12 bit		0.04		°C
Accuracy tolerance [2]	typ		±0.3		°C
	max	see Figure 3			°C
Repeatability			±0.1		°C
Operating Range	extended [4]	-40		125	°C
Response Time [7]	τ 63%	5		30	s
Long Term Drift			< 0.04		°C/yr

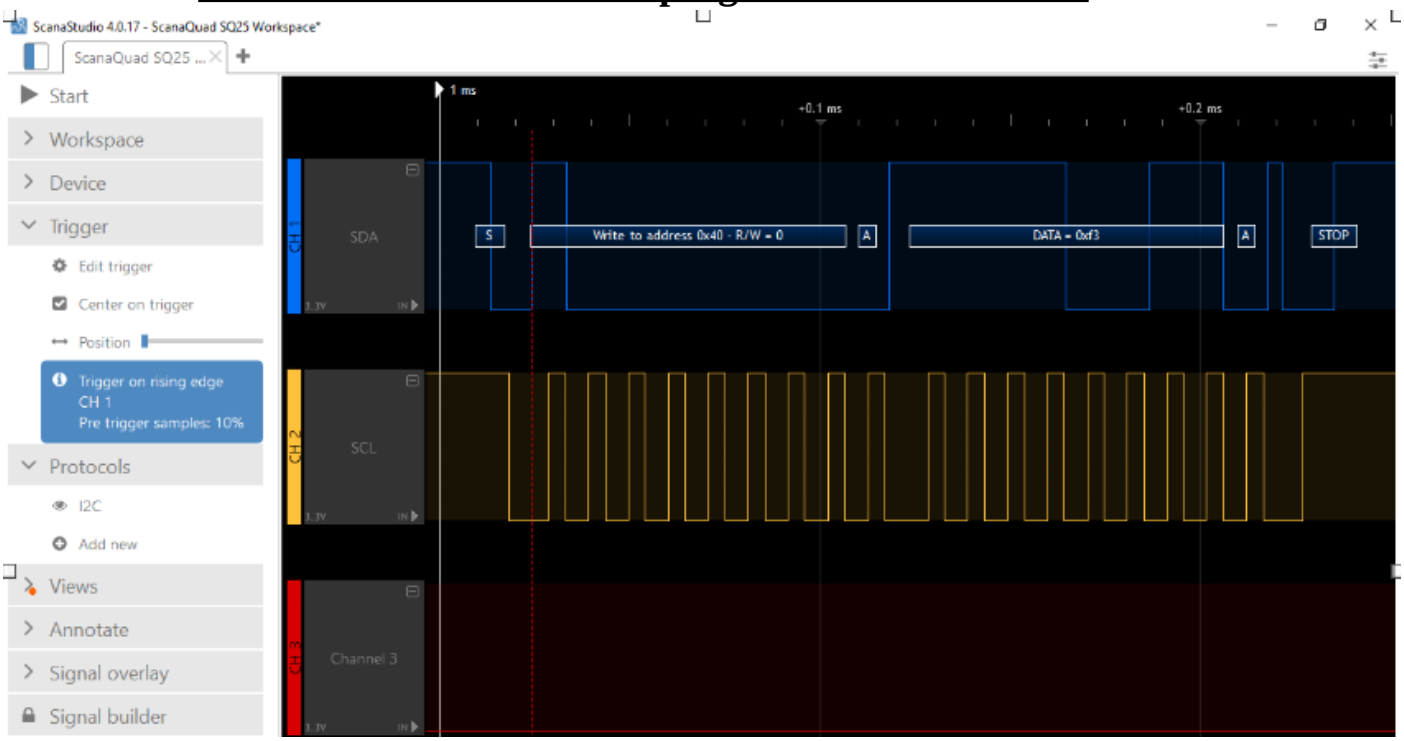
Electrical Specification

Parameter	Condition	min	typ	max	Units
Supply Voltage, VDD		2.1	3.0	3.6	V
Supply Current, IDD [6]	sleep mode		0.15	0.4	µA
	measuring	200	300	330	µA
Power Dissipation [6]	sleep mode		0.5	1.2	µW
	measuring	0.6	0.9	1.0	mW
	average 8bit		3.2		µW
Heater	VDD = 3.0 V	5.5mW, ΔT = + 0.5-1.5°C			
Communication	digital 2-wire interface, I²C protocol				

Sa tension typique est de 3 V et son intensité lors de mesure est 300 micro Ampère.

Partie EC1 : Delorme Valentin

➢ SHT21 fonctionnement du programme et trame I2C



En premier lieu on prépare la connexion en déterminant le débit en bit par seconde, on détermine également une broche d'entré. En second lieu on initialise les valeurs ST, SRH, mesureT, mesureRH.

Ensuite, la CubeCell démarre la transmission à l'esclave (SHT21) d'adresse 0x40, la CubeCell envoie 0xF3 puis demande le MSB et le LSB dans un tableau à l'esclave.

La valeur ST est déterminé par ce calcul : ST = (int)data[0]*256 + (int)(data[1]&0b11111100)

Cette valeur sera utilisé dans ce calcul pour déterminé la température en degré :

mesureT = (-46,85 + 175,72*ST/65536)

Pour l'humidité, remplacer 0xF3 par 0xF5 et ST par SRH, le calcul pour SRH est :

SRH = (int)data[0]*256 + (int)(data[1]&0b11110000)

Comme pour ST, SRH est utilisé dans le calcul suivant afin de déterminer l'humidité relative

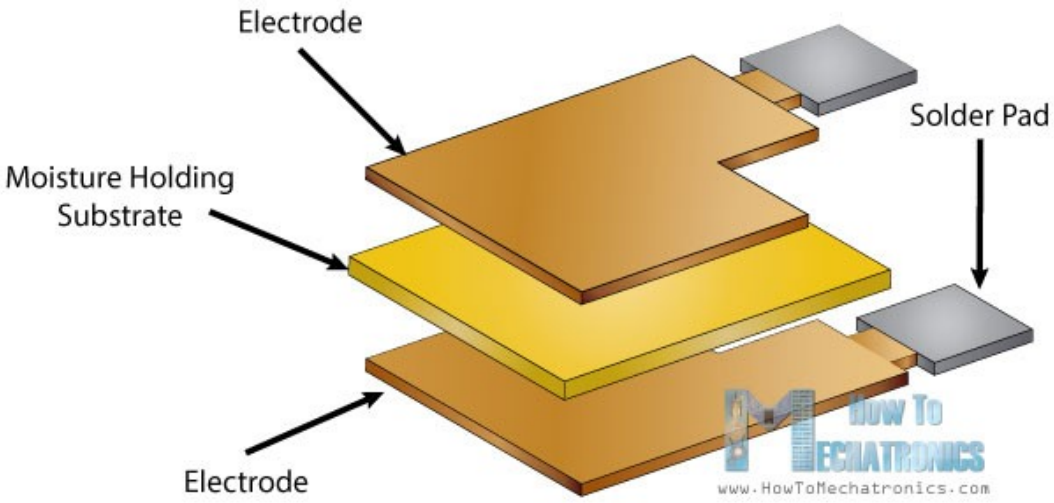
mesureRH= (-6.00 + 125.00*SRH/65536) | Les calculs proviennent de la doc du SHT21. (65536 fait référence à la résolution du SHT21, 16 bit)

Partie EC1 : Delorme Valentin

➢ SHT21 : Principes physiques

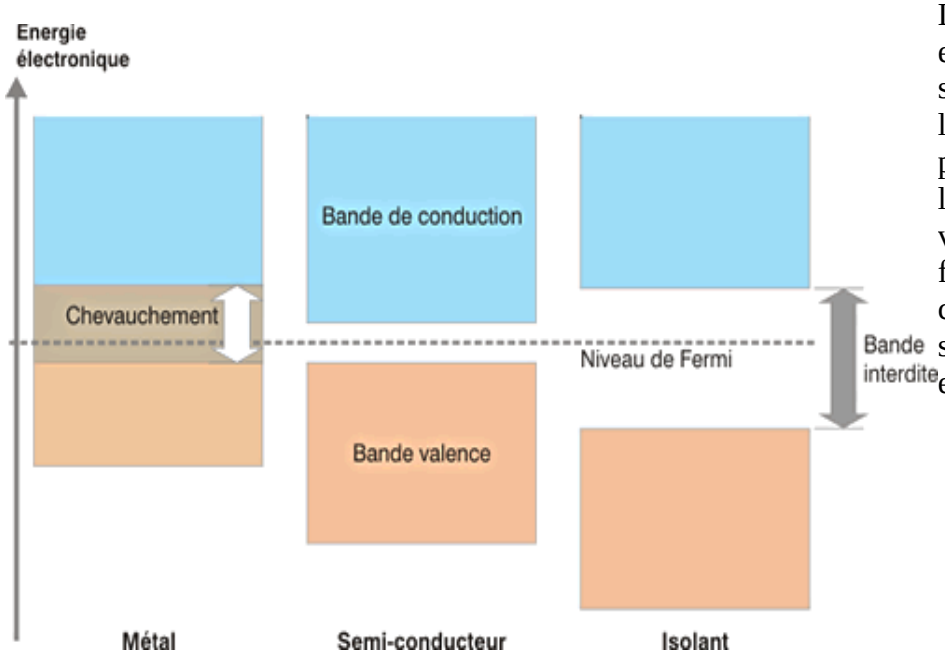
Le SHT21 utilise un élément capteur capacitif pour mesurer l'humidité, tandis que la température est mesurée par un capteur à bande interdite.

Humidité :



Le capteur d'humidité est capacitif, il mesure l'humidité relative en plaçant une fine bande d'oxyde métallique entre deux électrodes, la capacité électrique de l'oxyde métallique change avec l'humidité relative de l'atmosphère.

Température :

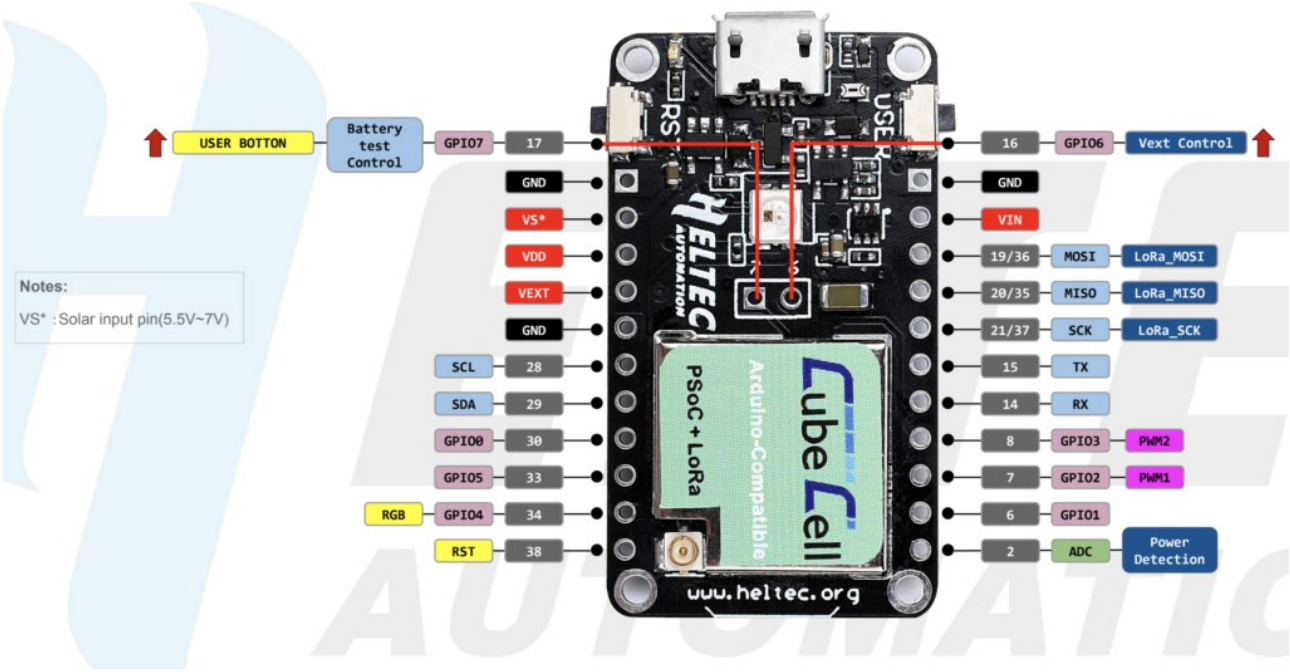


Le capteur de température est à bande interdite en silicium, il est inclus dans le circuit intégré. Le principe du capteur est que l'agitation des électrons de valence par la chaleur, les fera passer dans la zone de conduction. De fait, plus le silicone est chaud et plus il est conducteur.

Partie EC1 : Delorme Valentin

CubeCell

La CubeCell est une carte de développement faite par la société Heltec compatible avec l'IDE Arduino et pensée pour fonctionner en protocole LoRa. La série CubeCell est basée sur ASR605x (ASR6501, ASR6502), ces puces sont déjà intégrées aux microcontrôleurs de la série PsoC 4000 (ARM Cortex M0 + Core) et SX1262

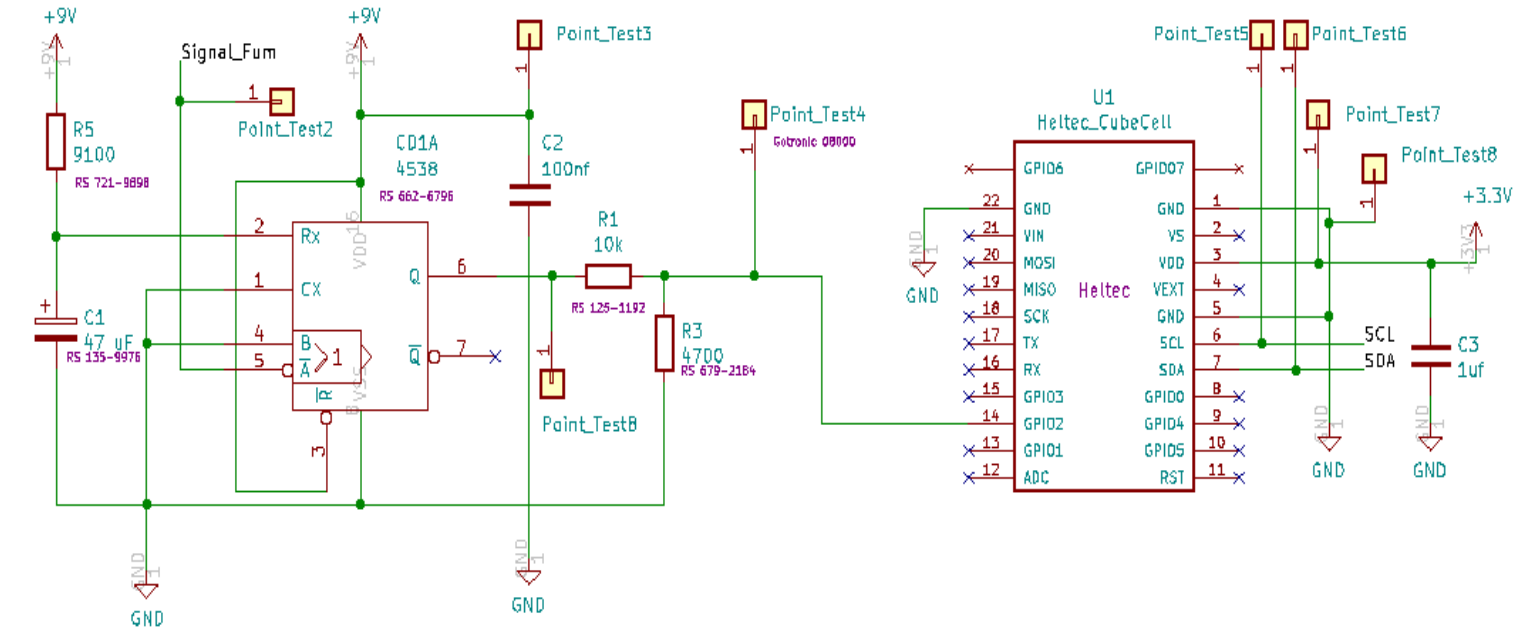


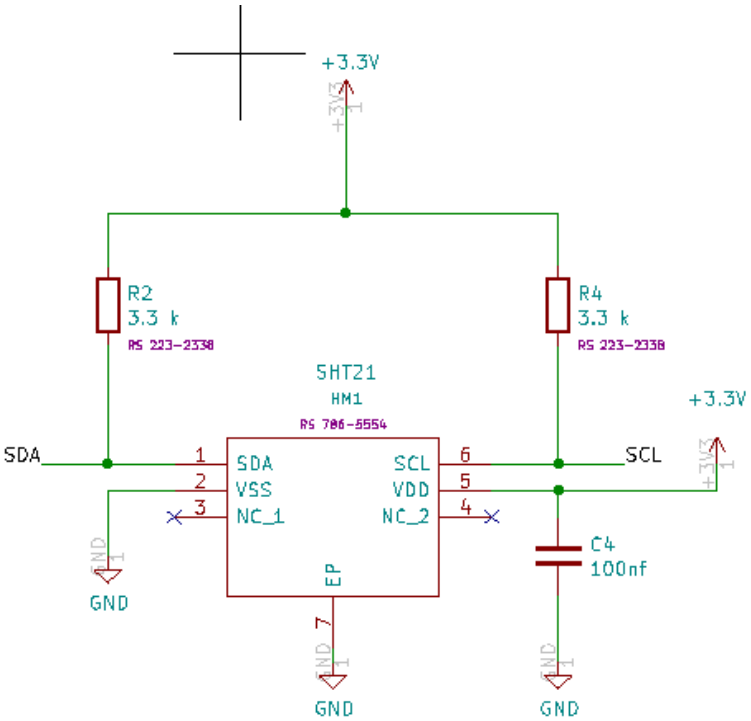
Voici un extrait de la documentation, les caractéristiques électriques de la carte change en fonction des conditions, elle peut donc être utilisé dans différentes situations.

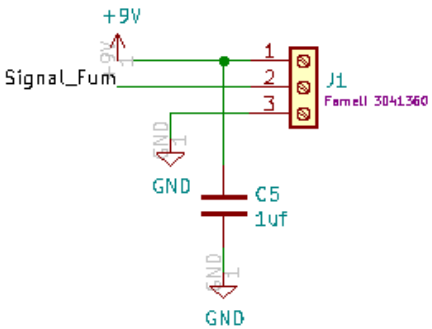
Electrical Features	Condition	Minimum	Typical	Maximum
Power Supply	USB powered (≥500mA)	4.7V	5V	6V
	Lithium powered (≥250mA)	3.3V	3.7V	4.2V
	3.3V (pin) powered (≥150mA)	2.7V	3.3V	3.5V
	5V (pin) powered (≥500mA)	4.7V	5V	6V
Power Consumption(mA)	LoRa Rx Mode		10mA	
	LoRa 10dB output		70mA	
	LoRa 14dB output		90mA	
	LoRa 17dB output		100mA	
	LoRa 20dB output		105mA	
	Sleep Mode (USB powered)		9.6mA	
	Sleep Mode (VBAT/battery powered)		11μA	
	Sleep Mode (3.3V header powered)		3.5μA	
Output	3.3V pin output			500mA
	5V pin output (USB powered only)		Equal to the input current	
	External device power control (Vext 3.3V)			350mA

Partie EC1 : Delorme Valentin

➢ Schéma et Routage







La CubeCell avec le CD4538 en haut

et

Le SHT21 avec ses résistance de pull-up et le bornier reliant le détecteur de fumée.

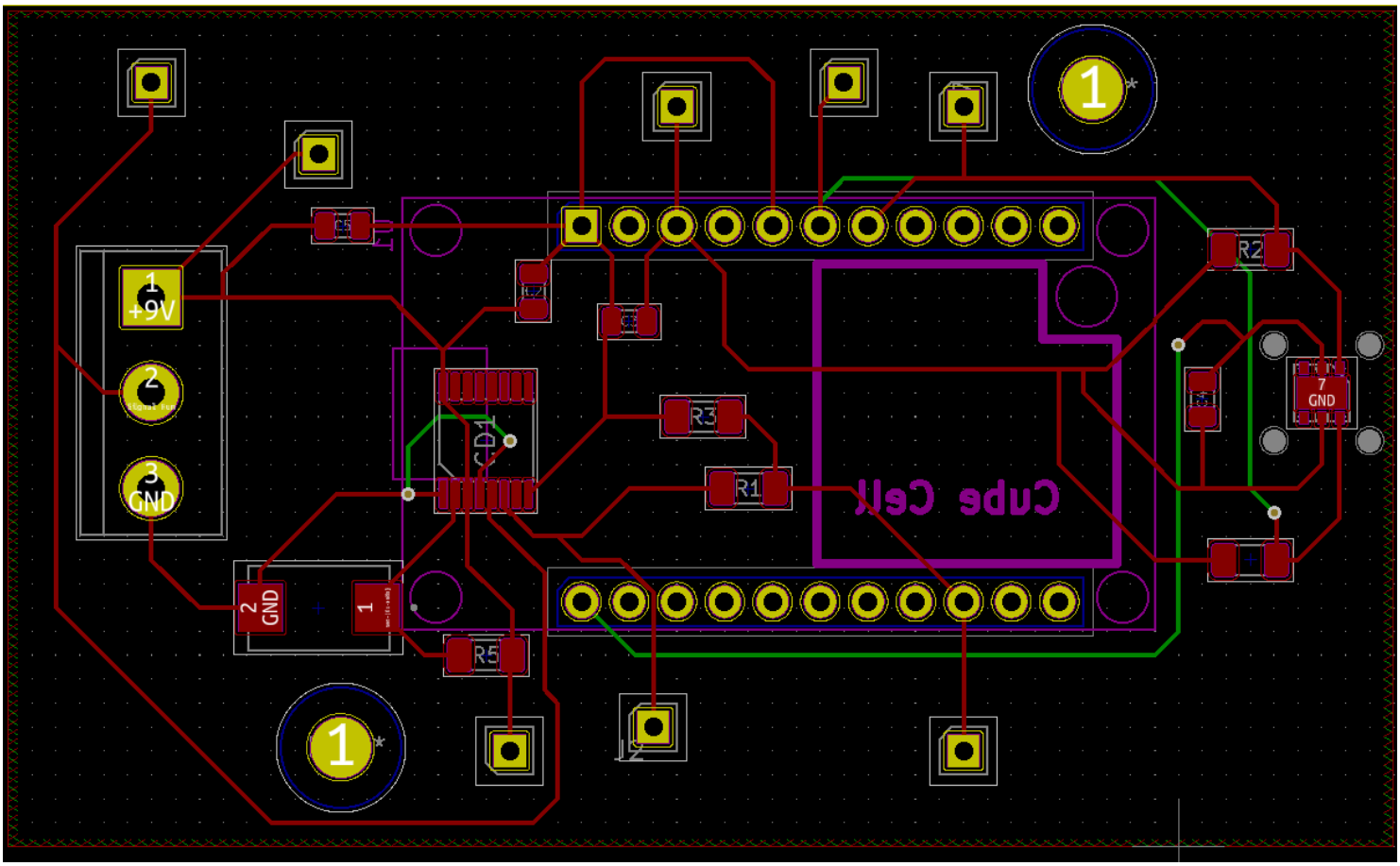
Détail :

Les fonctions analyse de température, humidité relative et calcul du point de rosé sont gérée par la carte CubeCell et le SHT21, ce dernier a besoin de résistance de pull-up car la CubeCell n'en a pas.

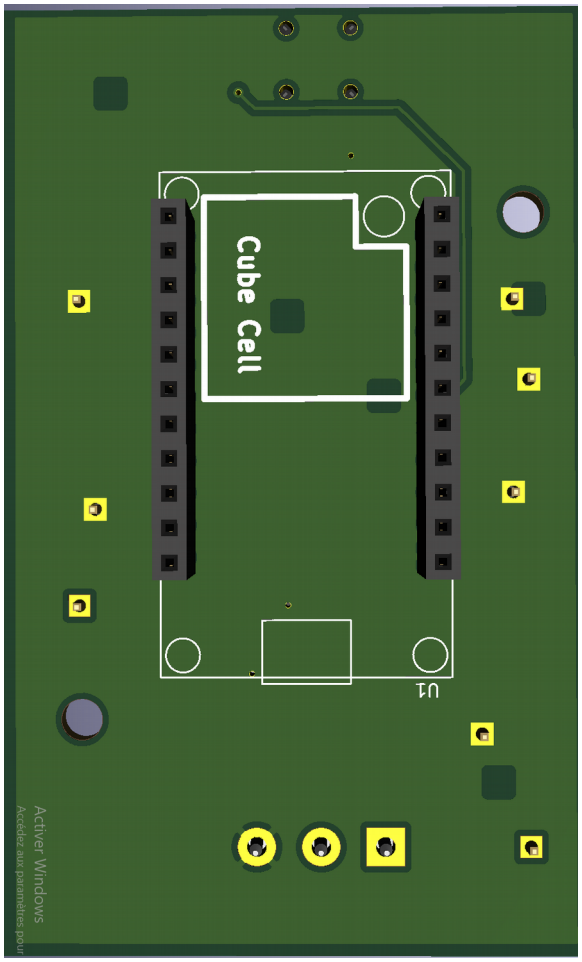
Le CD4538 est relié à la CubeCell il renvoie un signal dont la tension a été réduite par un pont diviseur pour correspondre a plage de la CubeCell. Il est là pour renvoyer un signal continue à partir du signal alternatif reçu par le bornier.

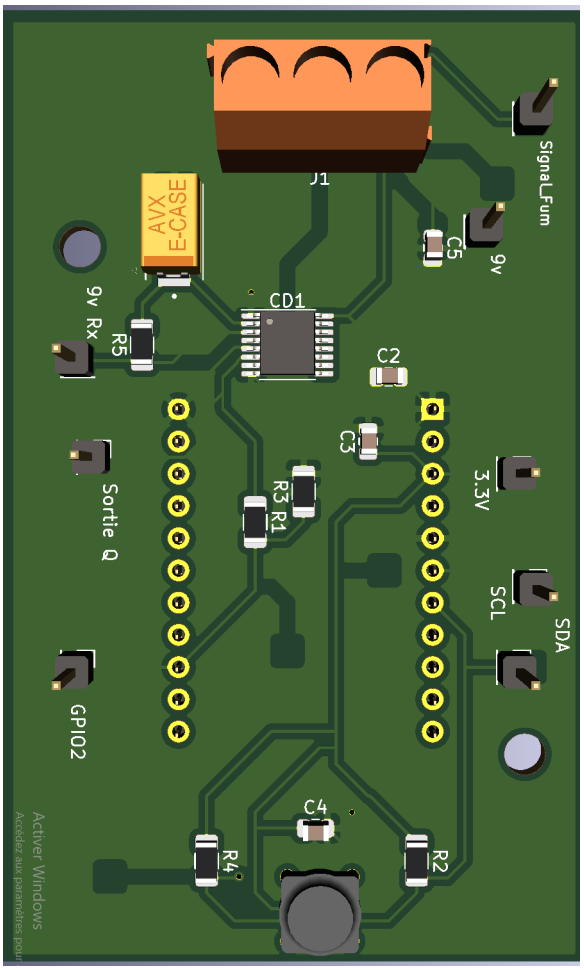
Le bornier reçoit le dit signal du détecteur de fumée qui n'est pas sur la carte.

Voici la première version du routage sur une carte 40x75 mm, deux trou de fixation

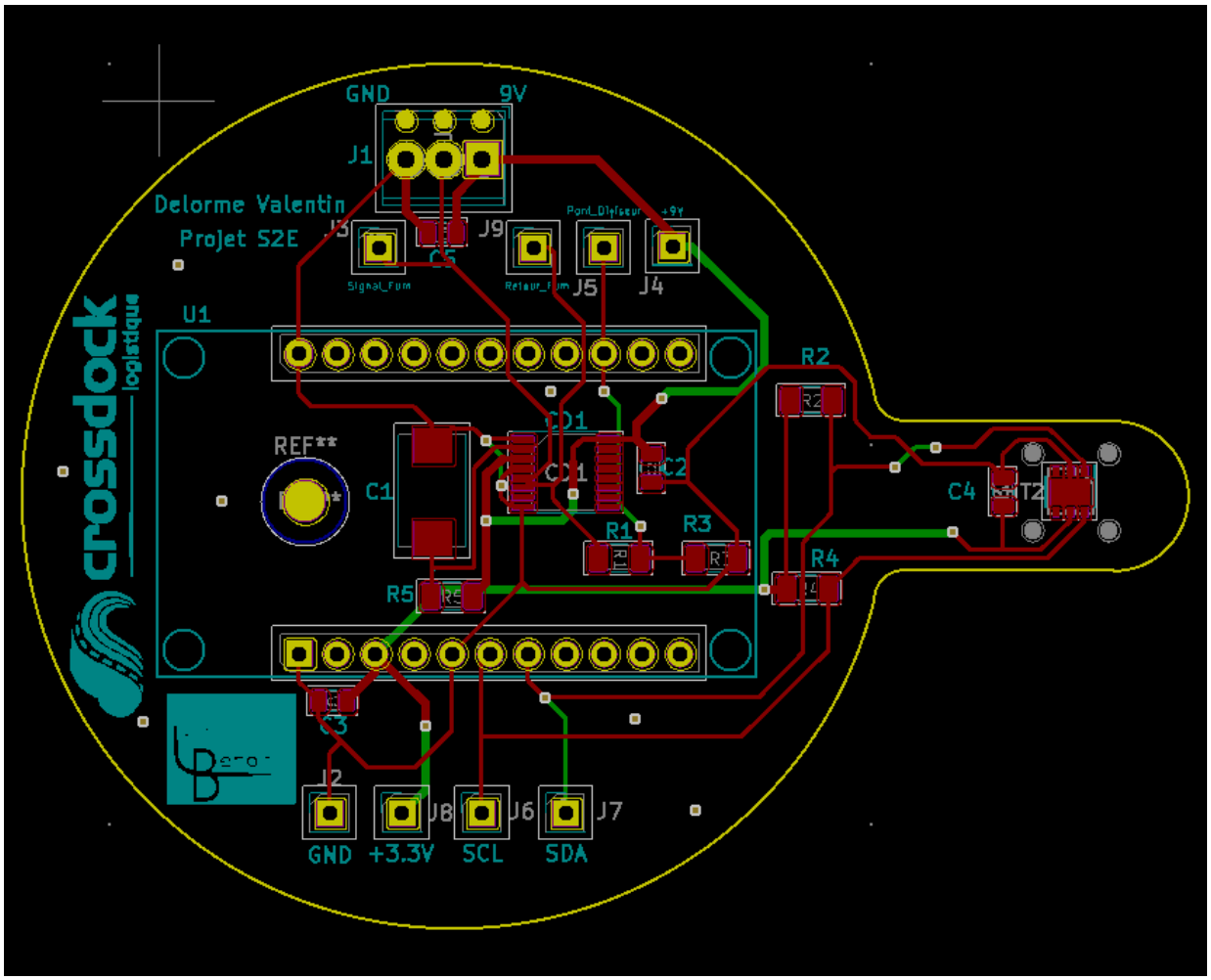


Et son rendu 3D





La version deux, la forme à été changer pour convenir à son boîtier



Rendu 3D n°2