Projet API-GTC

Weisseldinger Thomas

Salmon Robin

Gonzalez Thomas

Gestion Technique Centralisée d'un Eclairage Public avec communication XBee

SOMMAIRE

PARTIE COMMUNE **1**

Contexte 3

Présentation du projet 4

Solution avancée 5

Cas d'utilisation 6

Matériels/logiciels 8

Schéma synoptique 14

Diagramme des exigences 15

Diagramme de séquence 16

PARTIE INDIVIDUELLE ..... **20 - 93**

EC 22

IR2 44

IR1 72

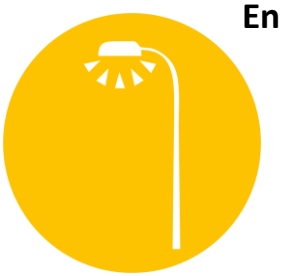
PARTIE ANNEXE **94**

PLAN DE DEVELOPPEMENT **100**

Contexte

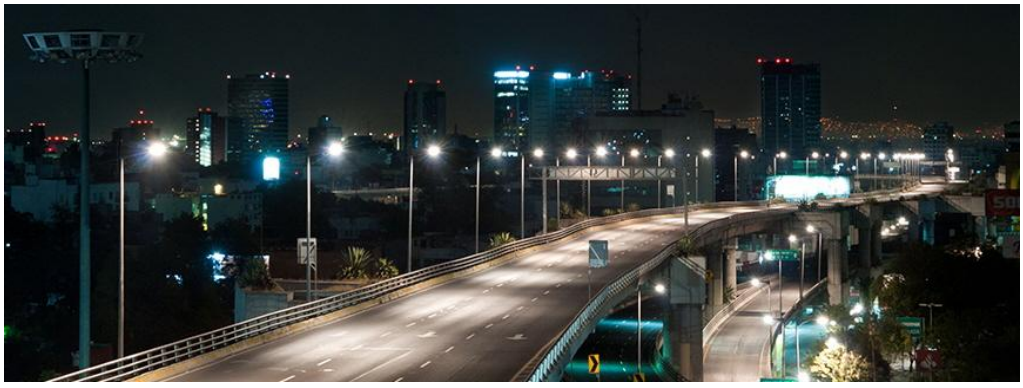
Dans les villes, l'éclairage public offre une sécurité publique, une grande aide à la perception d'obstacles, à la convivialité ainsi qu'à la possibilité de disposer de différentes ambiances pour embellir les patrimoines de la ville en question.

Malheureusement le système d'éclairage actuel dispose d'inconvénients majeurs lorsqu'il s'agit notamment du budget. En effet l'éclairage nocturne est un gouffre pour le budget des villes.



En France 9 millions de lampes fonctionnent entre 3500 et 4300 heures par an pour un total d'une puissance d'environ 1260 MW. Cette énergie représente 18% de toute l'énergie consommée par toutes les collectivités territoriales.

Les collectivités, vont devoir alors investir dans du matériel afin de pallier à ce problème budgétaire, tout en disposant d'une solution efficace, afin de pouvoir garder les points positifs de l'éclairage public tout en réduisant la consommation global d'électricité, ce qui va engendrer une réduction de la consommation de CO^2 général pour la commune.



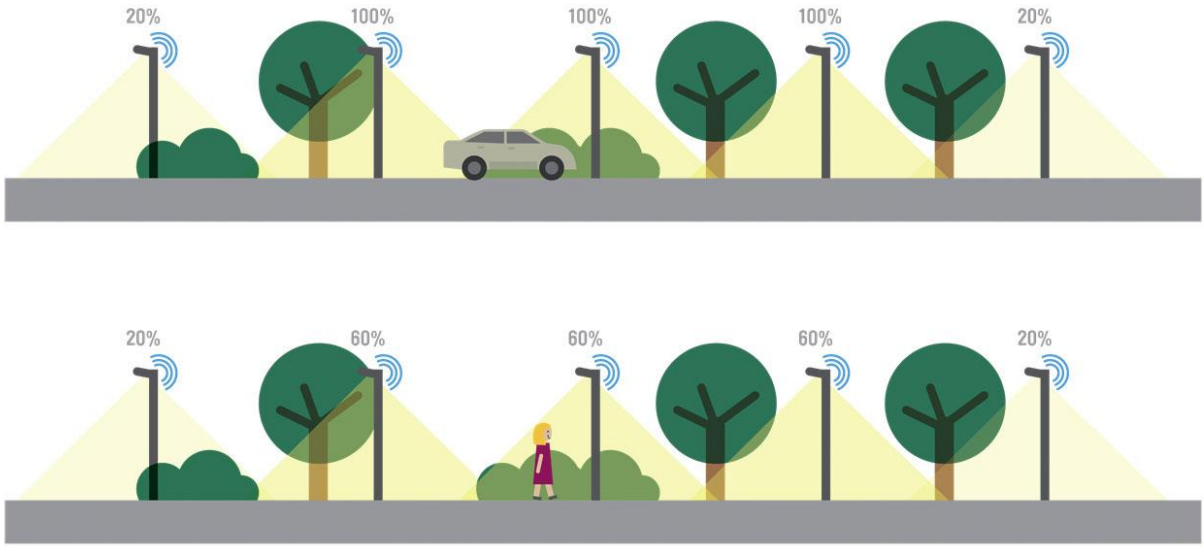
Exemple d'éclairage intelligent à Mexico

PRESENTATION DU PROJET

L'entreprise API située à Vedène et spécialisée dans la conception d'assemblage d'équipements de contrôle des processus industriels, a été sollicitée par plusieurs communes environnantes afin de réaliser un éclairage public plus économe et intelligent. API a donc fait appel au BTS SN pour la réalisation de cette tâche.

Le projet va consister à développer la solution vue précédemment. Il s'agira d'une solution a base de la technologie XBee™ développée par DIGI.

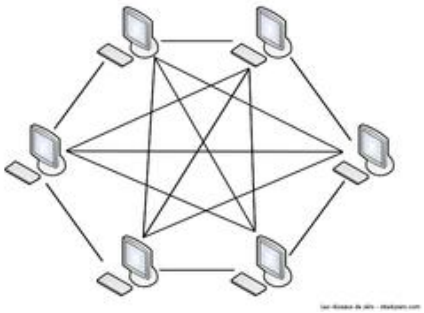
Afin de pouvoir créer un réseau de lampadaires intelligents et connectés qui permettrons de pouvoir modifier un a un l'éclairage de chacun d'eux avec une intensité choisi afin de ne pas ébouillir les piétons ou automobilistes sur place.



SOLUTION AVANCEE

La solution retenue fut la suivante : un système de télégestion au point lumineux. Cela permet d'utiliser un système de communication de type Intranet (dans notre cas, avec la technologie ZigBee) afin de piloter à distance individuellement ou collectivement les lampadaires connectés.

Dans une optique de communication entre lampadaires, nous avons opté pour une topologie réseau mesh (maillé) qui permet de transmettre les informations entre chaque lampadaire sans revenir au coordinateur (dans notre cas, le maître).

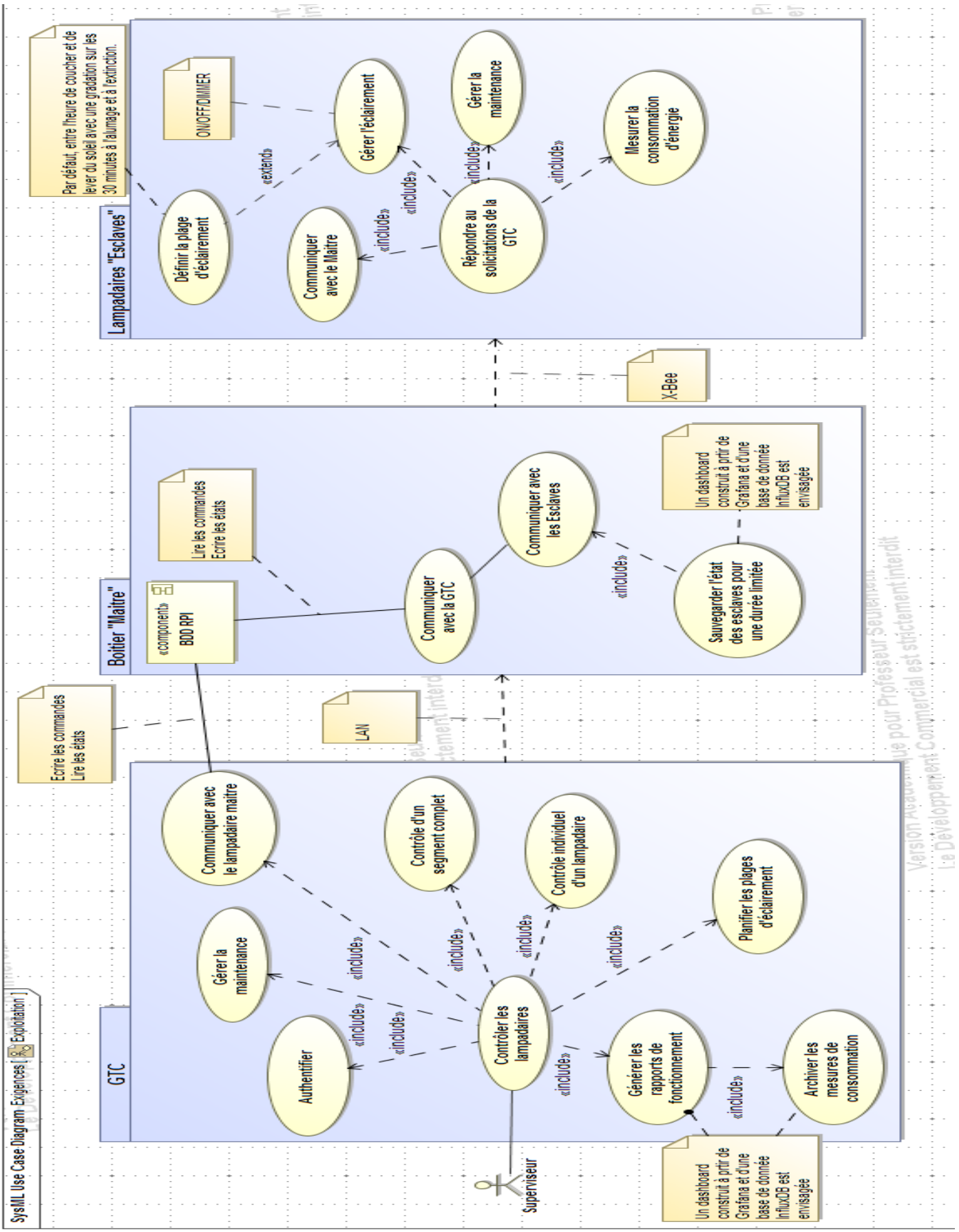


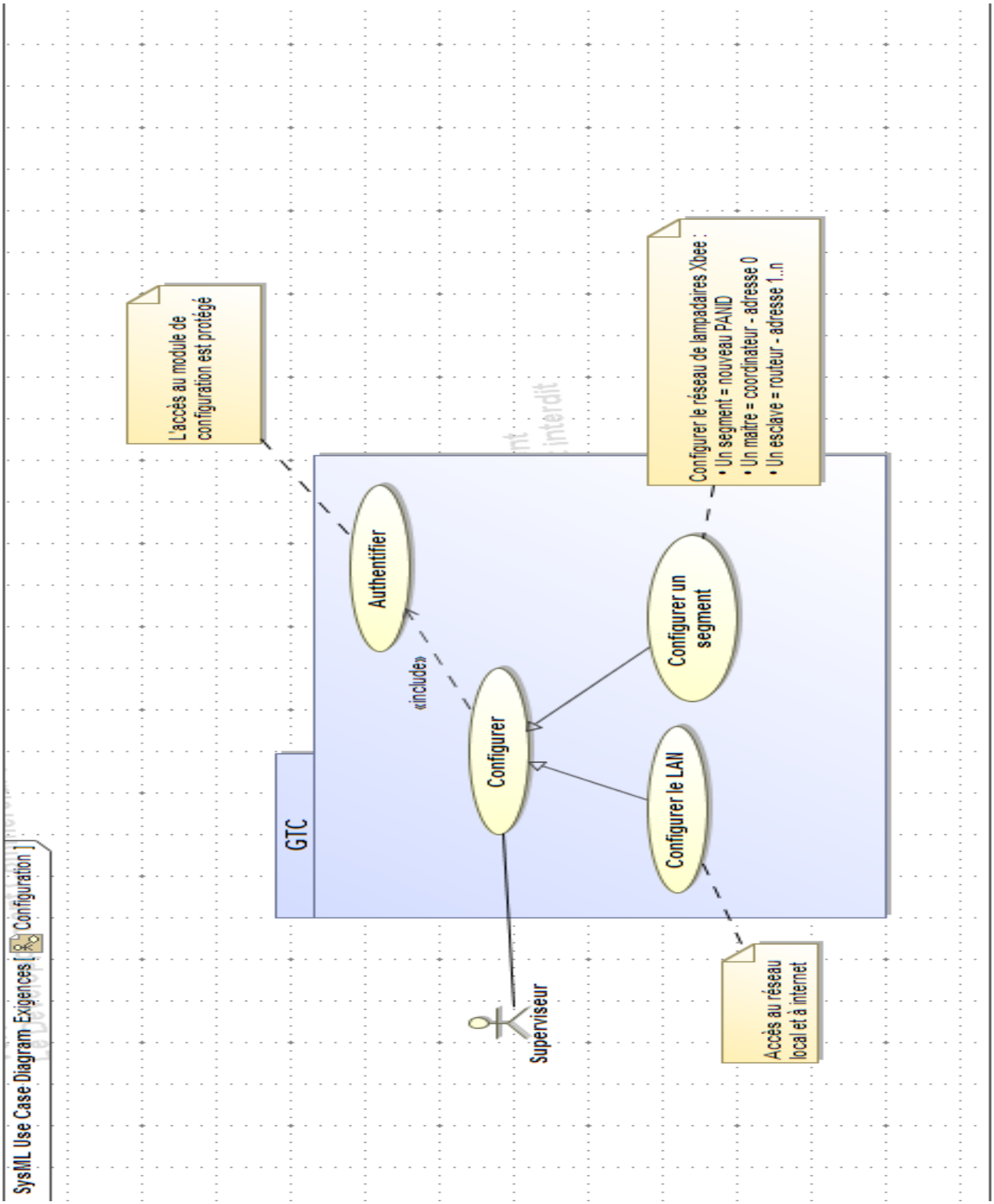
Cette topologie dispose de trois avantages :

- **le message est propagé le long du chemin en sautant d'un nœud à un autre jusqu'à atteindre sa destination**
- **il détermine automatiquement si un ou plusieurs nœuds du réseau sont manquants et répare automatiquement les routes endommagées**
- **créer le réseau sans aucune intervention humaine**

Il faut identifier deux types de nœuds : le coordinateur (le lampadaire maître) et les routeurs (les lampadaires esclaves). La notion de maître/esclave est nécessaire afin d'identifier quel est le rôle d'un lampadaire. En effet, le maître est celui qui va récupérer l'information transmise par l'utilisateur et l'envoyer au lampadaire sélectionné qui le transmettra, si nécessaire, au prochain et ainsi de suite.

Un logiciel sur PC permettra au poste de conduite GTC de planifier des plages d'éclairement, contrôler un lampadaire ou un segment (c'est-à-dire une zone ou lignée définie de plusieurs lampadaires), mesurer la consommation d'énergie et communiquer avec le lampadaire maître. De l'autre côté, une application destinée aux esclaves permettra de gérer l'éclairement, mesurer la consommation d'énergie et communiquer avec le maître. Enfin, une base de données stockera l'ensemble des informations (mesures etc…). Les modules XBee seront fixés sur le haut de chaque lampadaire.


SysML Use Case Diagram Exigences [Exploitation]
GTC
Communiquer avec le lampadaire maitre
Contrôle d'un segment complet
Contrôle individuel d'un lampadaire
Planifier les plages d'éclairement
Gérer la maintenance
Authentifier
Contrôler les lampadaires
Générer les rapports de fonctionnement
Archiver les mesures de consommation
«include»
Superviseur
Un dashboard construit à prtir de Grafana et d'une base de donnée InfluxDB est envisagée
Ecrire les commandes
Lire les états
LAN
Boitier "Maitre"
«component» BDD RPI
Lire les commandes
Ecrire les états
Communiquer avec la GTC
Communiquer avec les Esclaves
Sauvegarder l'état des esclaves pour une durée limitée
Un dashboard construit à prtir de Grafana et d'une base de donnée InfluxDB est envisagée
X-Bee
Lampadaires "Esclaves"
Par défaut, entre l'heure de coucher et de lever du soleil avec une gradation sur les 30 minutes à l'allumage et à l'extinction.
Définir la plage d'éclairement
ON/OFF/DIMMER
«extend»
Gérer l'éclairement
Communiquer avec le Maitre
«include»
Répondre au solicitations de la GTC
Gérer la maintenance
Mesurer la consommation d'énergie

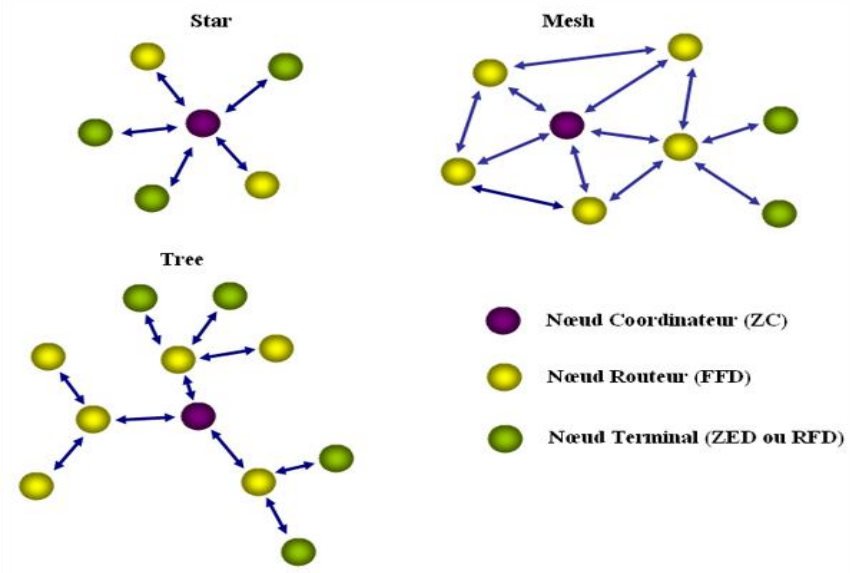

SysML Use Case Diagram Exigences [Configuration]
L'accès au module de configuration est protégé
Configurer le réseau de lampadaires Xbee :
• Un segment = nouveau PANID
• Un maître = coordinateur - adresse 0
• Un esclave = routeur - adresse 1..n
GTC
Authentifier
«include»
Configurer
Configurer un segment
Configurer le LAN
Superviseur
Accès au réseau local et à internet

Matériels

a) Modules XBee et Embarqués

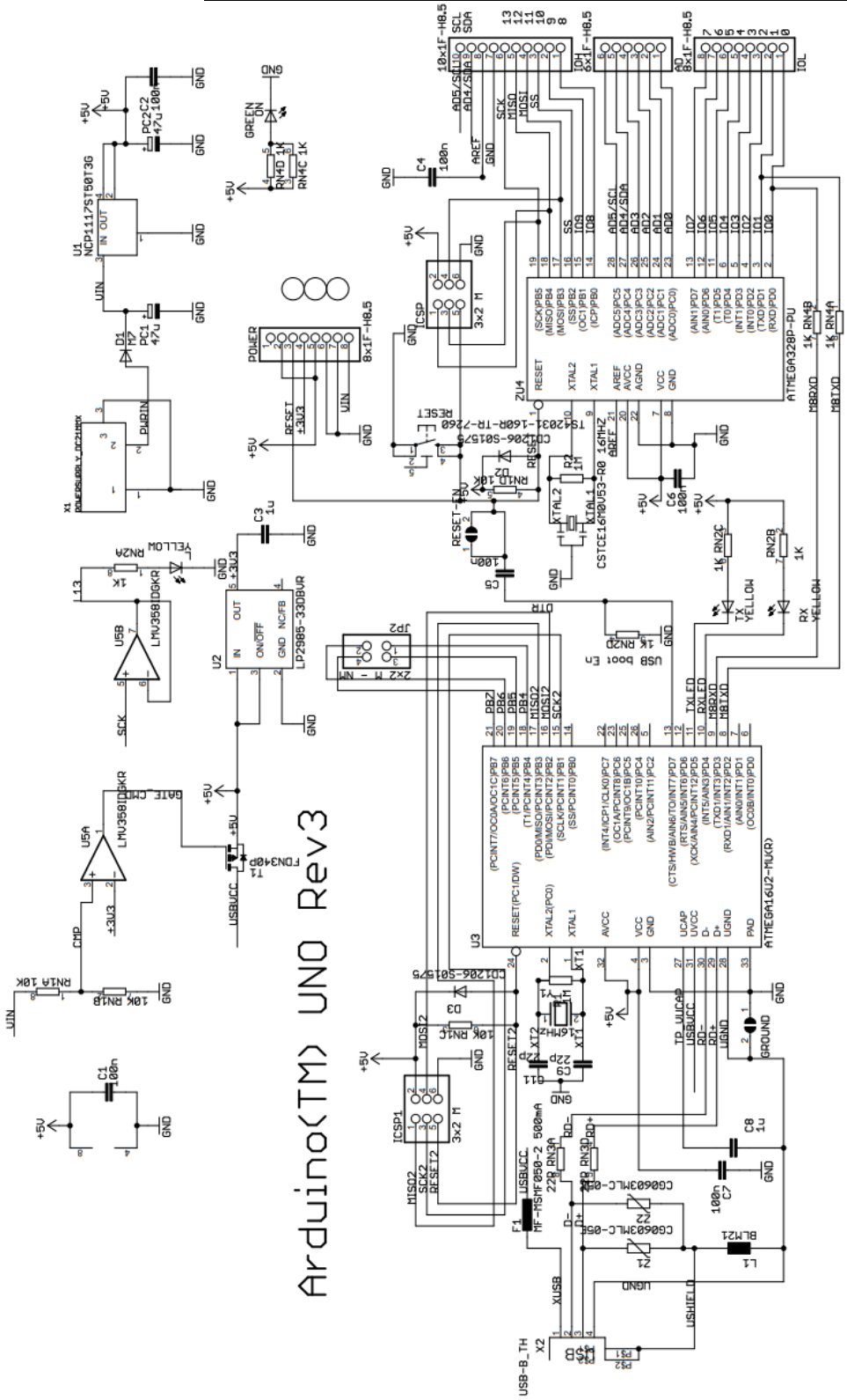
Bande	Disponibilité	Nombre de canaux	Vitesse maxi théorique
868 MHz	Europe	1	20 kbit/s
915 MHz	Amériques et Australie	10	40 kbit/s
2.4 GHz	Disponible partout	16	250 kbit/s

	XBee DegiMesh 2.4
Fréquence	**2.4 GHz**
Distance Max	**Classique : 90m Pro : 1.6km**
Vitesse	**250kbit/s**
Communication Protocol	**Proprietary**
Maximum transmit power	**Classique : 1mW 0dBm Pro : 63 mW 18 dBm**
Supply voltage	**2.8 to 3.4VDC**
Consommation mode veille	**<50µA**
Certified Regions	**US,CA,EU,AU,BR,JP**
Serial data interfaces	**UART**



Arduino uno R3

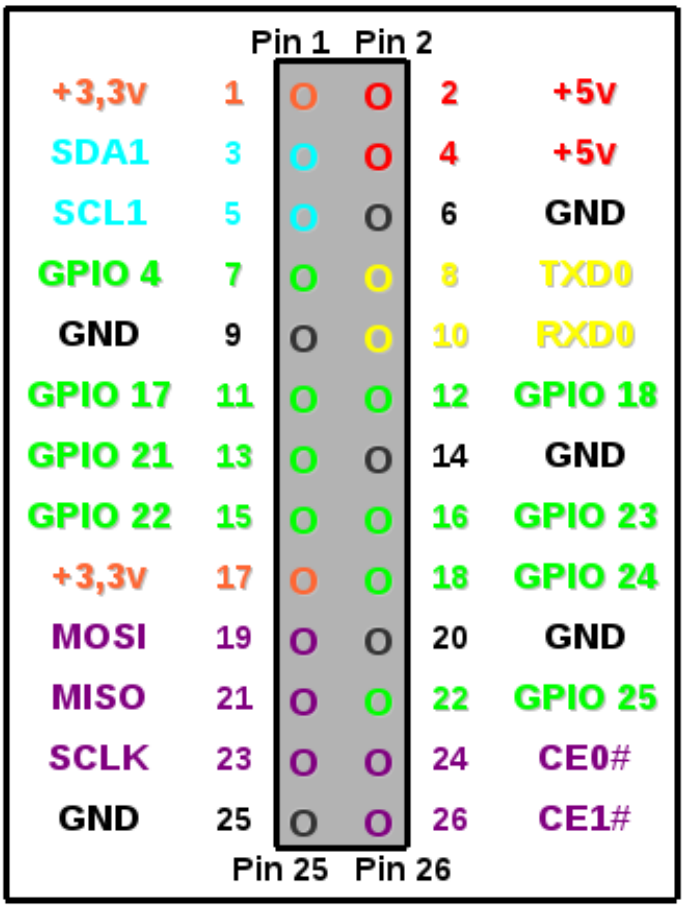
Micro-controleur	Mémoire flash	Broche E/S	Dimensions en pouce	Dimensions en mm
ATmega328P	32	14	2,7'' x 2,1''	68,6mm x 53.3mm



Modèle 3 B

RaspberryPi

Prix	35$
Cpu	1.2 quadricoeur ARM Cortex-A53
Gpu	VideoCore VI
Mémoire RAM	1Go
Port usb	4
S video	HDMI
S audio	Stereo Jack 3.5mm
Lecture / ecriture	MicroSD
Reseaux	10/100 Ethernet
Bas niveau	17xGPIO UART,I²C,SPI
Conso max	720mA
Alimentation	5V via micro usb
Dimension	85.60mm x 53.98 mm x 17 mm
poid	45g
OS	all



b) Auxiliaires

Référence	LEF12KPW
Puissance	72W
Consommation	79W
Flux nominal	12.32Lm
Flux réel	12.07Lm
C° couleur	5500°K
Efficacité lumineuse	152Lm/W
Dimension	615 x 320 x 140 mm

Angle 150°		
Distance	Diamètre	Surface illuminée
1 m	746,40 cm	43,75 m²
2 m	1492,80 cm	175,02 m²
3 m	2239,20 cm	393,80 m²
4 m	2985,60 cm	700,09 m²
5 m	3732,10 cm	1093,94 m²
6 m	4478,40 cm	1575,20 m²
7 m	5224,80 cm	2144,02 m²

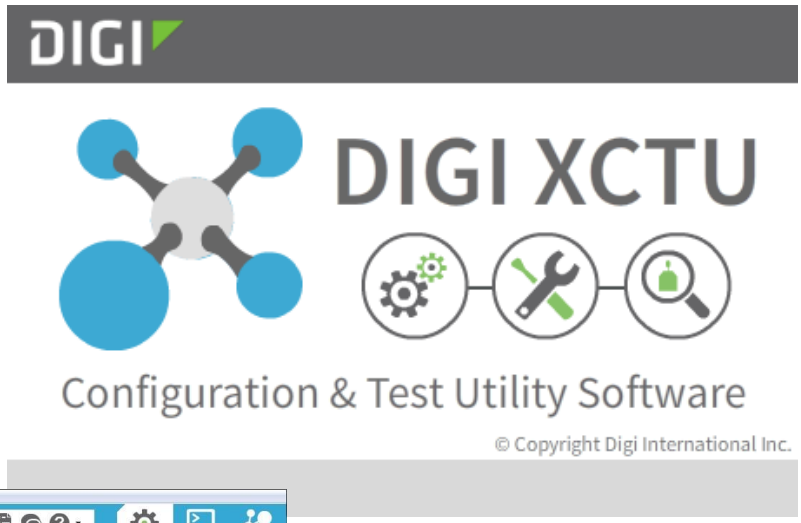
LEF12K	
Eclairement min	Eclairement max
269,68 lx	8849,91 lx
67,42 lx	2950,16 lx
27,51 lx	1311,12 lx
16,85 lx	737,51 lx
10,79 lx	471,99 lx
7,66 lx	335,28 lx
5,63 lx	246,32 lx

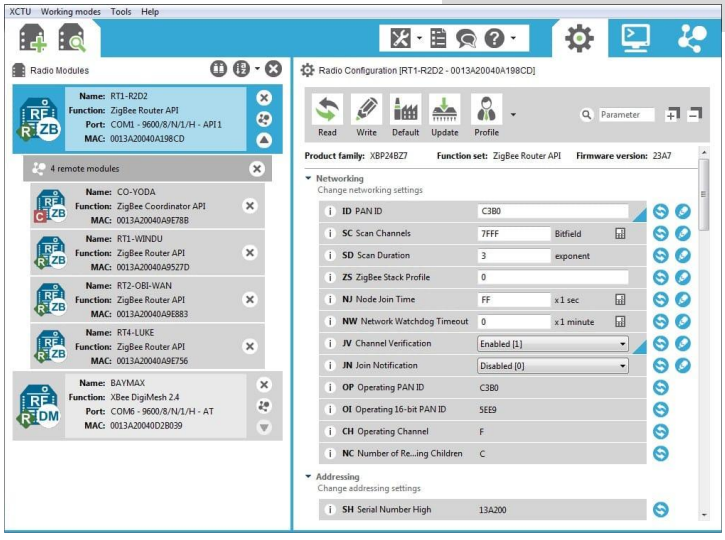
MORE THAN LIGHT

c) Logiciel

XCTU

XCTU est un logiciel conçu par DIGI afin de configurer des modules XBee. Pour cela, il suffit de sélectionner le bouton « Discover radio modules », puis le port correspondant au module XBee et enfin paramétrer correctement le port (dans notre cas utiliser la fonction DigiMech pour le réseau maillé). Pour terminer et passer à la configuration, il faut appuyer sur « Add selected devices ».

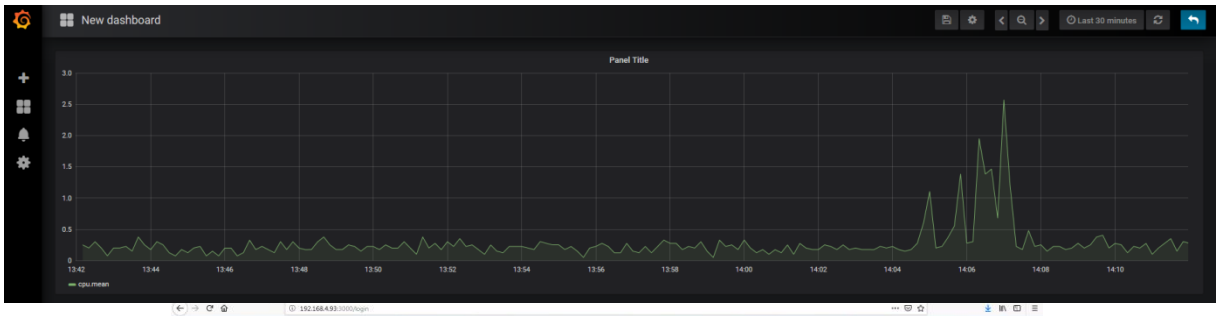


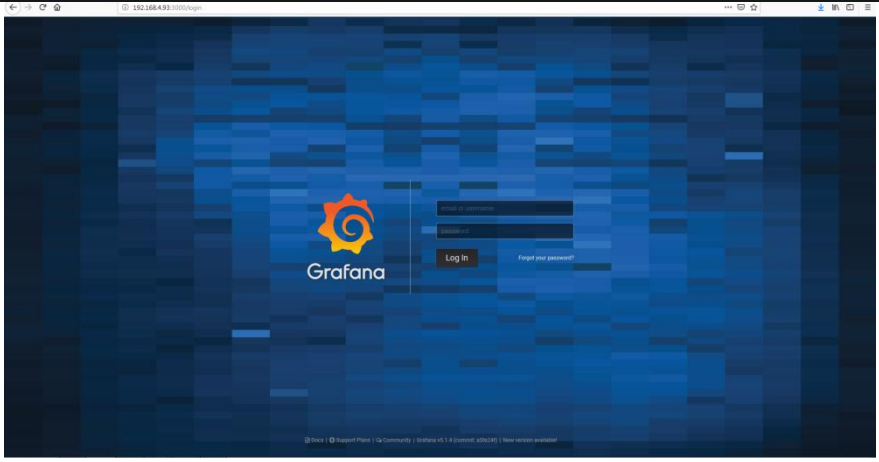


Influxdb est un système de gestion de base de donné gérable depuis la console. Il est disponible sur windows/linux/Macos/raspbianOS ce qui lui permet une grande flexibilité de support. Ce logiciel s'accorde avec Graf ana et Telegraph que nous allons utiliser.

Grafana

Grafana est un logiciel de supervision qui nous permet d'avoir des informations de la base de donné en temps réel a base de graphique, de tableaux, ou encore de texte. Ce système de supervision fonctionne particulièrement bien avec Influxdb. Aussi appelé "interface web" car accessible avec l'IP réseau + port de son hôte par un navigateur web comme firefox ou Google chrome

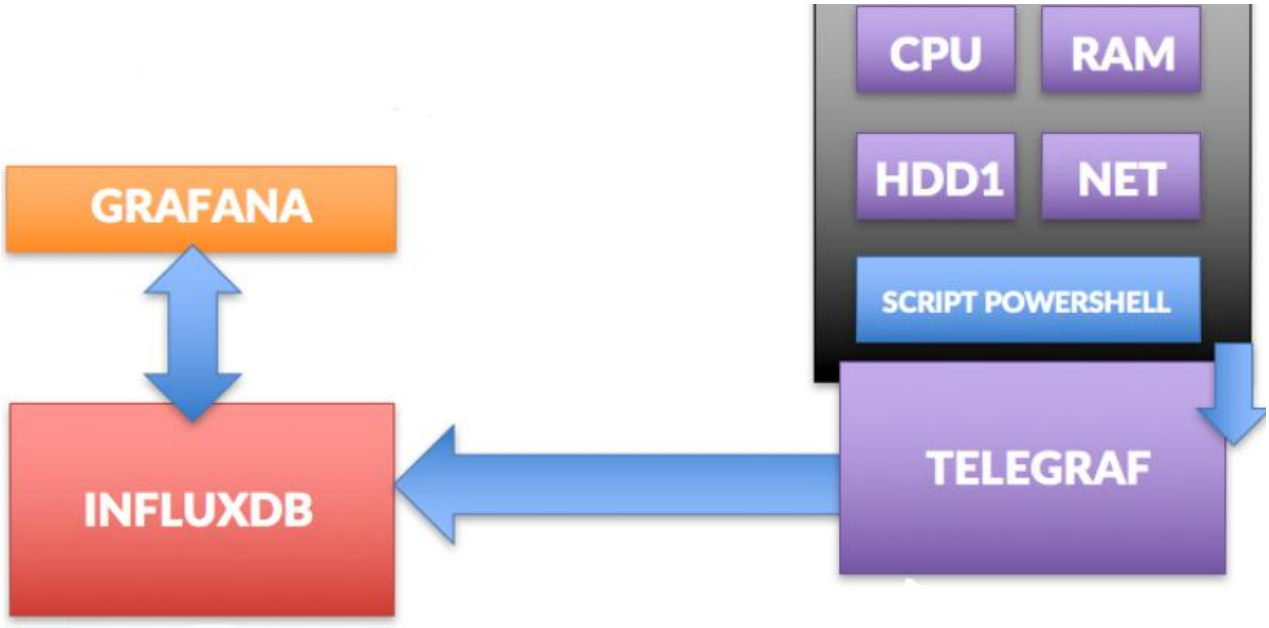


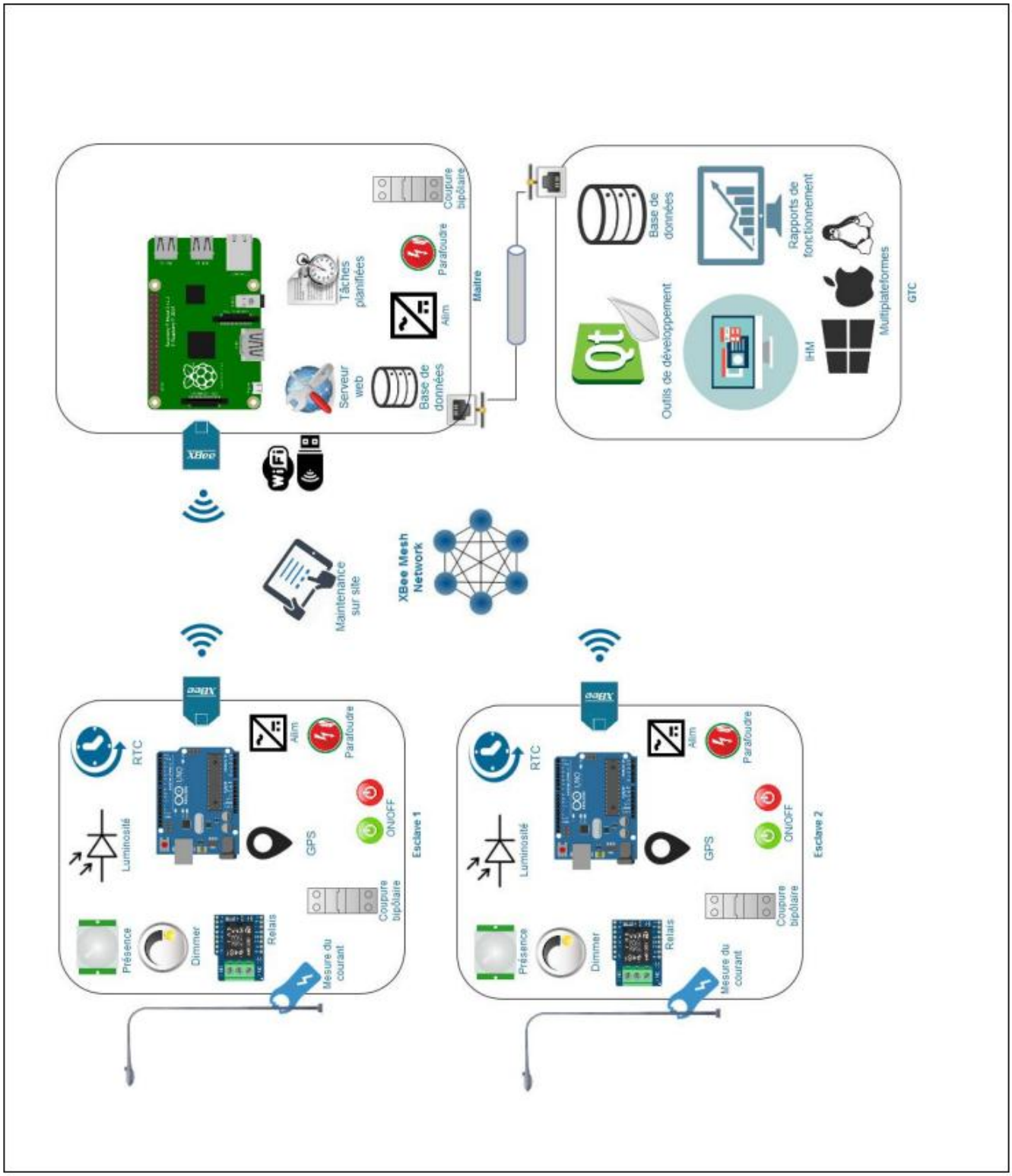


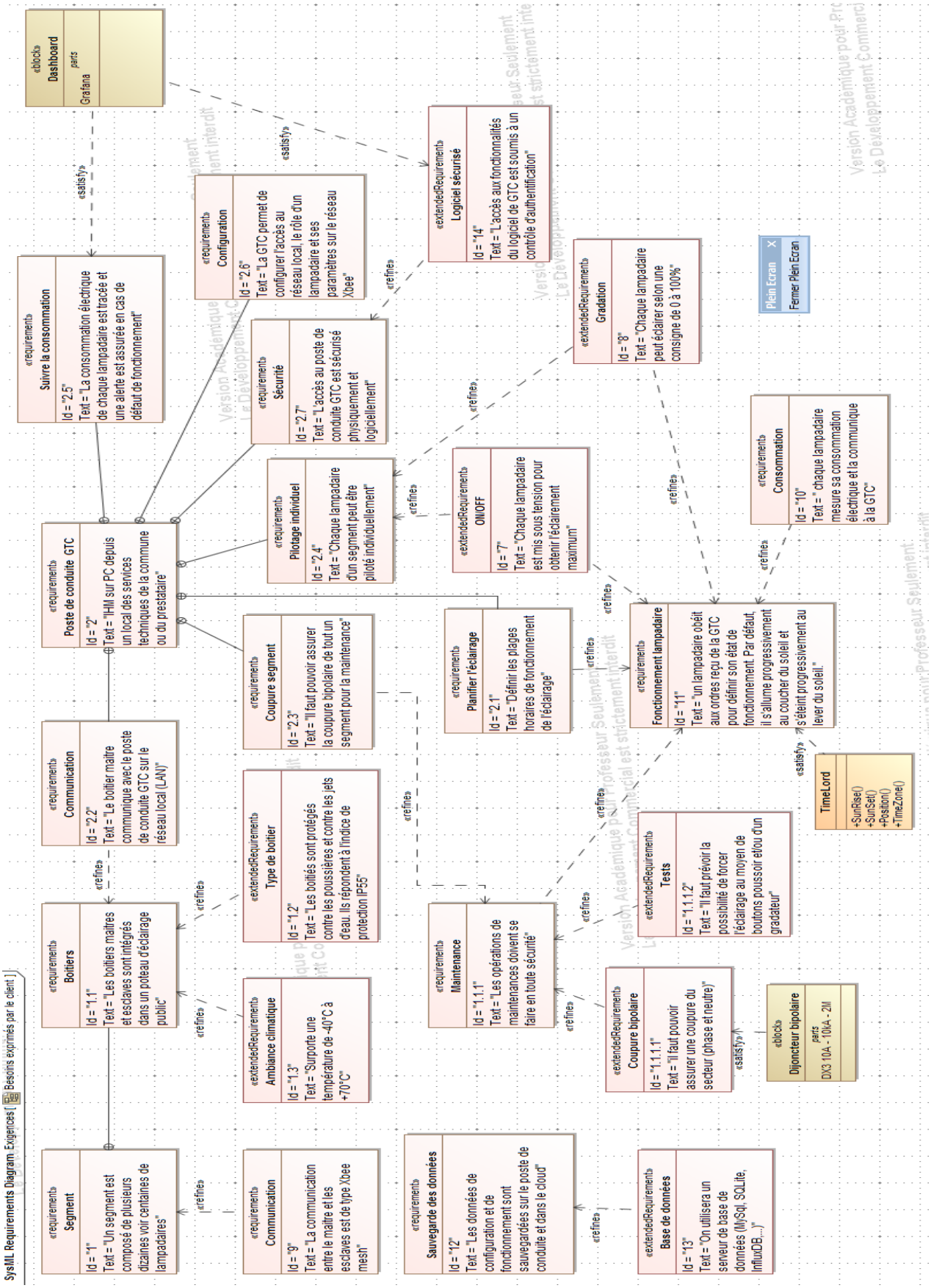
Telegraf

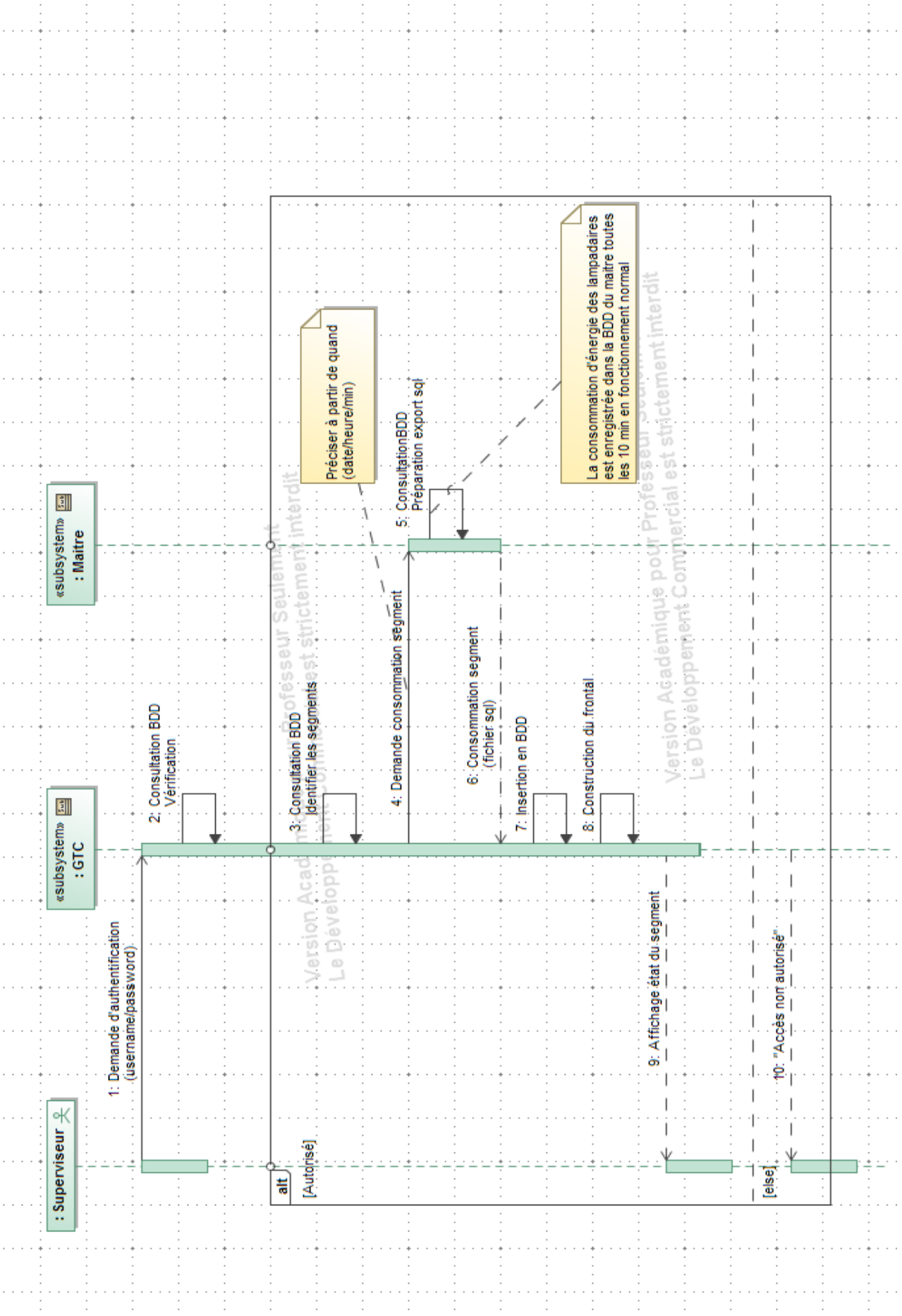
Telegraf est l'entré de la chaine des 3 logiciels+

En visionnant le CPU utilisé, la RAM ou bien d'autres.




Coupure bipolaire
Parafoudre
Tâches planifiées
Alim
Maître
Serveur web
Base de données
Base de données
Rapports de fonctionnement
Multiplateformes
Outils de développement
IHM
GTC
XBee
WiFi
Maintenance sur site
XBee Mesh Network
RTC
Alim
Parafoudre
Luminosité
GPS
ON/OFF
Coupure bipolaire
Présence
Dimmer
Relais
Mesure du courant
Esclave 1
Esclave 2


SysML Requirements Diagram Exigences [Besoins exprimés par le client]
«block» Dashboard
parts
Grafana
«satisfy»
«requirement» Suivre la consommation
Id = "2.5"
Text = "La consommation électrique de chaque lampadaire est tracée et une alerte est assurée en cas de défaut de fonctionnement"
«requirement» Configuration
Id = "2.6"
Text = "La GTC permet de configurer l'accès au réseau local, le rôle d'un lampadaire et ses paramètres sur le réseau Xbee"
«extendedRequirement» Logiciel sécurisé
Id = "14"
Text = "L'accès aux fonctionnalités du logiciel de GTC est soumis à un contrôle d'authentification"
«requirement» Sécurité
Id = "2.7"
Text = "L'accès au poste de conduite GTC est sécurisé physiquement et logiciellement"
«requirement» Poste de conduite GTC
Id = "2"
Text = "IHM sur PC depuis un local des services techniques de la commune ou du prestataire"
«requirement» Pilotage individuel
Id = "2.4"
Text = "Chaque lampadaire d'un segment peut être piloté individuellement"
«extendedRequirement» ON/OFF
Id = "7"
Text = "Chaque lampadaire est mis sous tension pour obtenir l'éclairement maximum"
«extendedRequirement» Gradation
Id = "8"
Text = "Chaque lampadaire peut éclairer selon une consigne de 0 à 100%"
«requirement» Consommation
Id = "10"
Text = " chaque lampadaire mesure sa consommation électrique et la communique à la GTC"
Plein Ecran
Fermer Plein Ecran
«requirement» Communication
Id = "2.2"
Text = "Le boitier maitre communique avec le poste de conduite GTC sur le réseau local (LAN)"
«requirement» Coupure segment
Id = "2.3"
Text = "Il faut pouvoir assurer la coupure bipolaire de tout un segment pour la maintenance"
«requirement» Planifier l'éclairage
Id = "2.1"
Text = "Définir les plages horaires de fonctionnement de l'éclairage"
«requirement» Fonctionnement lampadaire
Id = "11"
Text = "un lampadaire obéit aux ordres reçu de la GTC pour définir son état de fonctionnement. Par défaut, il s'allume progressivement au coucher du soleil et s'éteint progressivement au lever du soleil."
«requirement» Boitiers
Id = "1.1"
Text = "Les boitiers maitres et esclaves sont intégrés dans un poteau d'éclairage public"
«extendedRequirement» Type de boitier
Id = "1.2"
Text = "Les boitiés sont protégés contre les poussières et contre les jets d'eau. Ils répondent à l'indice de protection IP55"
«requirement» Maintenance
Id = "1.1.1"
Text = "Les opérations de maintenances doivent se faire en toute sécurité"
«extendedRequirement» Tests
Id = "1.1.1.2"
Text = "Il faut prévoir la possibilité de forcer l'éclairage au moyen de boutons poussoir et/ou d'un gradateur"
TimeLord
+SunRise()
+SunSet()
+Position()
+TimeZone()
«extendedRequirement» Ambiance climatique
Id = "1.3"
Text = "Surporte une température de -40°C à +70°C"
«extendedRequirement» Coupure bipolaire
Id = "1.1.1.1"
Text = "Il faut pouvoir assurer une coupure du secteur (phase et neutre)"
«block» Disjoncteur bipolaire
parts
DX3 10A - 10kA - 2M
«requirement» Segment
Id = "1"
Text = "Un segment est composé de plusieurs dizaines voir centaines de lampadaires"
«requirement» Communication
Id = "9"
Text = "La communication entre le maitre et les esclaves est de type Xbee mesh"
«requirement» Sauvegarde des données
Id = "12"
Text = "Les données de configuration et de fonctionnement sont sauvegardées sur le poste de conduite et dans le cloud"
«extendedRequirement» Base de données
Id = "13"
Text = "On utilisera un serveur de base de données (MySql, SQLite, InfluxDB,...)"
«refine»
«satisfy»


«subsystem»
: Maitre
«subsystem»
: GTC
: Superviseur
1: Demande d'authentification
(username/password)
2: Consultation BDD
Vérification
3: Consultation BDD
Identifier les segments
4: Demande consommation segment
5: ConsultationBDD
Préparation export sql
6: Consommation segment
(fichier sql)
7: Insertion en BDD
8: Construction du frontal
9: Affichage état du segment
10: "Accès non autorisé"
Préciser à partir de quand
(date/heure/min)
La consommation d'énergie des lampadaires
est enregistrée dans la BDD du maitre toutes
les 10 min en fonctionnement normal
alt
[Autorisé]
[else]

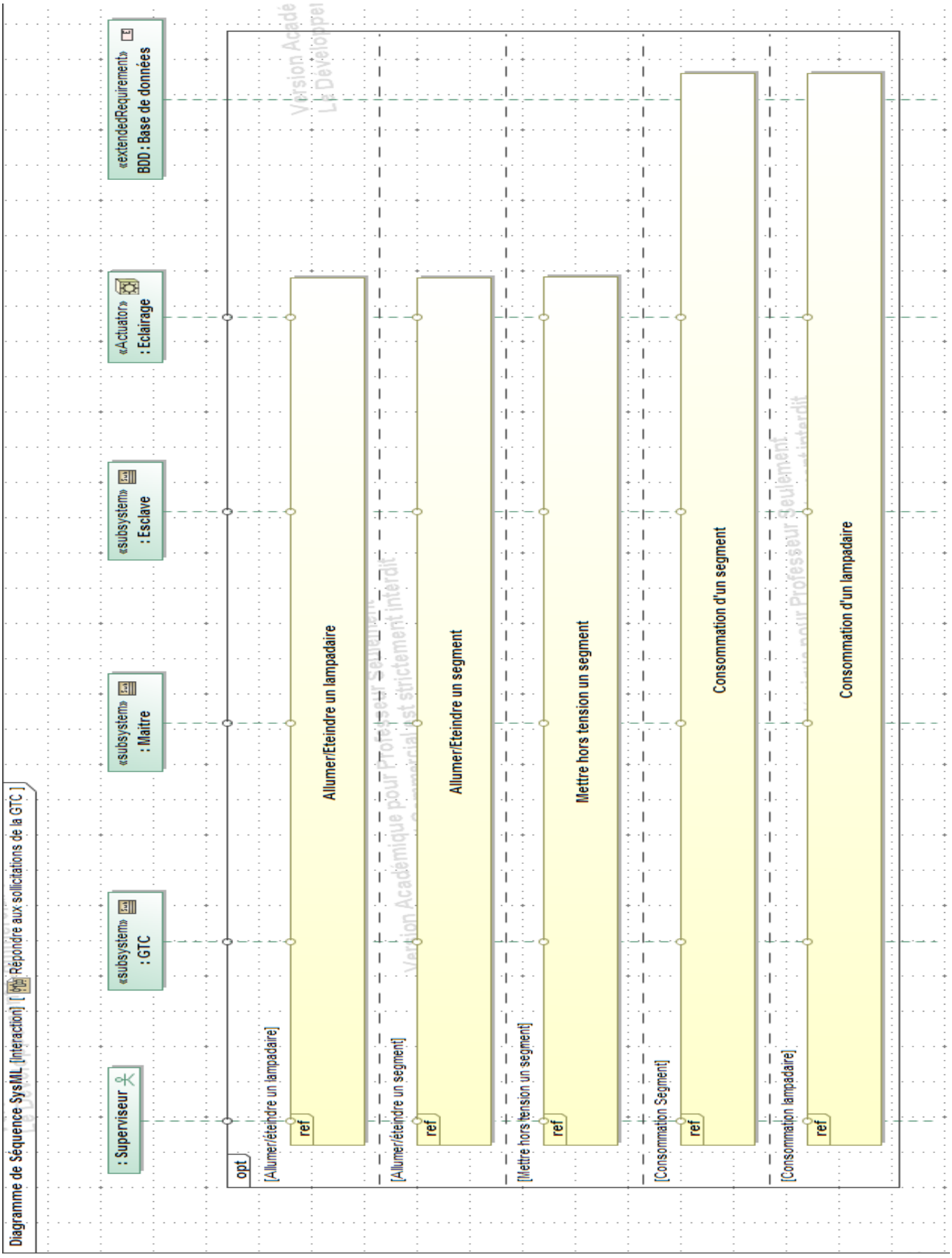


Diagramme de Séquence SysML [Interaction] [Répondre aux sollicitations de la GTC]
: Superviseur
«subsystem» : GTC
«subsystem» : Maitre
«subsystem» : Esclave
«Actuator» : Eclairage
«extendedRequirement» BDD : Base de données
opt
[Allumer/éteindre un lampadaire]
ref
Allumer/Eteindre un lampadaire
[Allumer/éteindre un segment]
ref
Allumer/Eteindre un segment
[Mettre hors tension un segment]
ref
Mettre hors tension un segment
[Consommation Segment]
ref
Consommation d'un segment
[Consommation lampadaire]
ref
Consommation d'un lampadaire

Certaines taches prévues ont étaient effectuées plus tard.

Projet	Mer 09/01/19	Ven 21/06/19		
Specifications générales	Mer 09/01/19	Mer 09/01/19		
Configuration des modules XBEE	Jeu 10/01/19	Jeu 10/01/19		IR2;IR1
Test de communication entre les modules XBEE avec XCTU	Jeu 10/01/19	Jeu 10/01/19		IR1;IR2
Définition des plages horaires de fonctionnement de l'éclairage	Ven 11/01/19	Ven 11/01/19		IR1;IR2
Installation des différents logiciels	**Jeu 10/01/19**	**Mer 16/01/19**		**EC1;IR1;IR2**
Vérification des schémas	Mer 09/01/19	Jeu 10/01/19		EC1
Réflexion sur la solution d'alimentations des cartes	Mer 16/01/19	Mer 16/01/19		EC1
Abordage de la communication ZigBee	Mer 16/01/19	Mer 16/01/19		EC1;IR1;IR2
Test de l'horloge temps réel	Mer 16/01/19	Mer 16/01/19		EC1
Conception de l'IHM du poste de conduite GTC	Ven 25/01/19	Mer 30/01/19		IR2
Test de mesure du courant	Mer 23/01/19	Mer 23/01/19		EC1
Validation des différentes structures	Mer 23/01/19	Mer 23/01/19		EC1
Revue 1	Mar 05/02/19	Jeu 07/02/19		

Création de la BDD sur InfluxDB	Mer 27/02/19	Ven 08/03/19		IR1
Création des différentes requêtes	**Mer 09/01/19**	**Mer 09/01/19**		**EC1;IR1;IR2**
Réception des plages horaires par les maîtres	Mer 13/02/19	Jeu 14/02/19		IR1;IR2
Saisie du schéma de synthèse KiCad	Mer 13/02/19	Ven 15/02/19		EC1
Installation des boîtiers maîtres/esclaves sur les poteaux d'éclairages	Ven 15/02/19	Ven 15/02/19		EC1;IR2
Conception de l'application sur Raspberry	Mer 17/04/19	Ven 17/05/19		IR2
Revue 2	Jeu 25/04/19	Jeu 02/05/19		

Conception de l'application sur Raspberry	Mer 17/04/19	Ven 17/05/19		IR2
Codage du logiciel GTC	Ven 03/05/19	Ven 24/05/19		IR2
Fabrication de la carte pour le lampadaire	Mer 08/05/19	Ven 10/05/19		EC1
Contrôle du bon fonctionnement des esclaves et remontée des défauts de fonctionnement à la GTC	Mer 09/01/19	Mer 09/01/19		IR1[75%];IR2[25%]
Mesures de consommation et envoie des informations au maître	Mer 29/05/19	Jeu 30/05/19	36	EC1;IR1;IR2
Pilotage de l'ensemble des esclaves rattachés	Mer 09/01/19	Mer 09/01/19		IR1
Essais	Jeu 30/05/19	Jeu 30/05/19		EC1;IR1;IR2
Remise du dossier	Mer 05/06/19	Mer 05/06/19	45	
Soutenance finale	Lun 17/06/19	Ven 21/06/19	41	

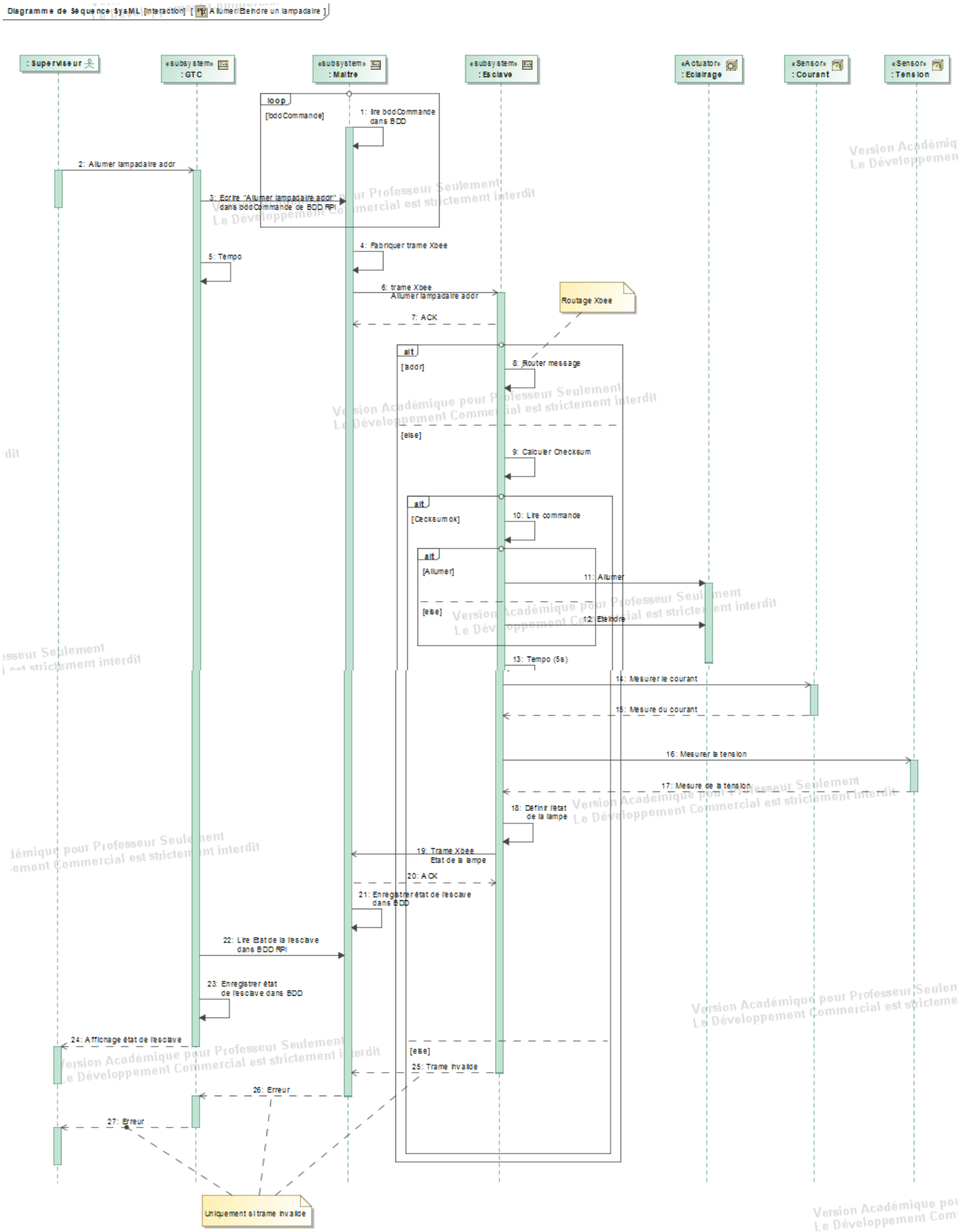
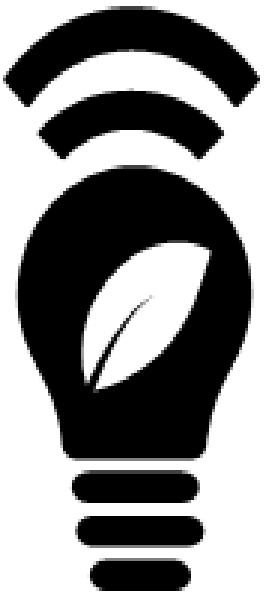


Diagramme de Séquence SysML [Interaction] [Allumer/Eteindre un lampadaire]
: Superviseur
«subsystem» : GTC
«subsystem» : Maitre
«subsystem» : Esclave
«Actuator» : Eclairage
«Sensor» : Courant
«Sensor» : Tension
loop
[toddCommande]
1: lire toddCommande dans BDD
2: Allumer lampadaire addr
3: Ecrire "Allumer lampadaire addr" dans toddCommande de BDD RPi
4: Fabriquer trame Xbee
5: Tempo
6: trame Xbee Allumer lampadaire addr
Routage Xbee
7: ACK
alt
[addr]
8: Router message
[else]
9: Calculer Checksum
alt
[Cecksum ok]
10: Lire commande
alt
[Allumer]
11: Allumer
[else]
12: Eteindre
13: Tempo (5s)
14: Mesurer le courant
15: Mesure du courant
16: Mesurer la tension
17: Mesure de la tension
18: Définir l'état de la lampe
19: Trame Xbee Etat de la lampe
20: ACK
21: Enregistrer état de l'escave dans BDD
22: Lire Etat de la l'esclave dans BDD RPi
23: Enregistrer état de l'esclave dans BDD
24: Affichage état de l'esclave
[else]
25: Trame invalide
26: Erreur
27: Erreur
Uniquement si trame invalide

Projet API-GTC

Revue 3 - Partie Individuelle



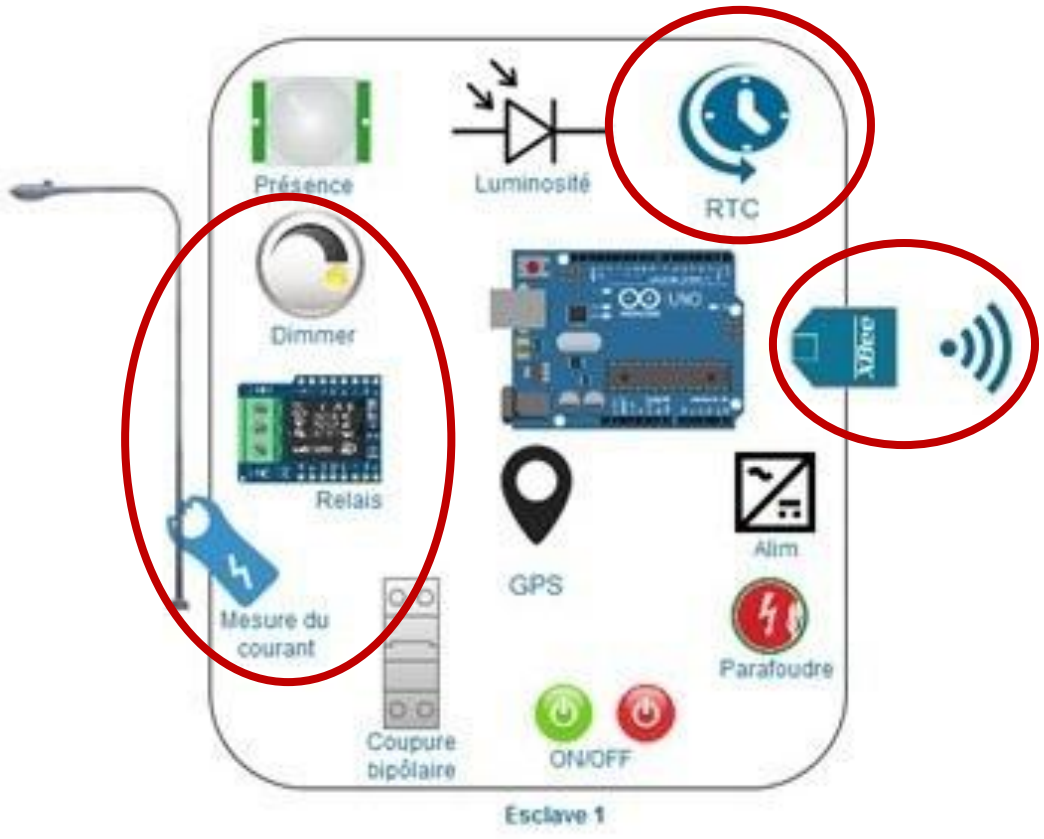
SALMON Robin EC1

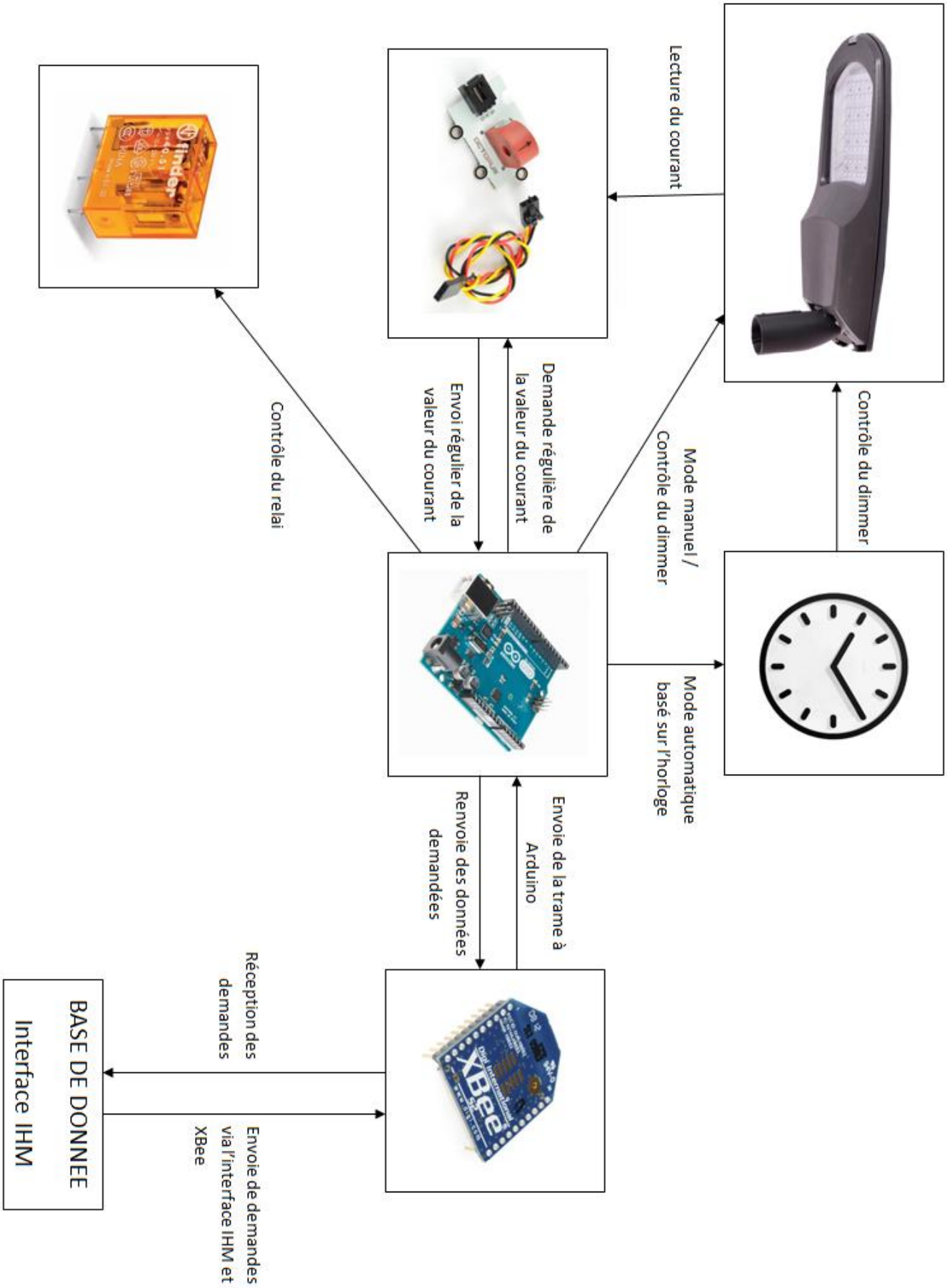
Gestion Technique Centralisée d'un Eclairage Public avec communication XBee

Introduction

Pour ma part, dans notre projet, je suis le seul élève d'EC. Mon travail consiste dans un premier temps à tester et valider tous les éléments proposés sur le prototype de l'année précédente (câblage et test avec des portions de programme Arduino). Ensuite, la deuxième partie de mon travail consistera en la conception d'un shield (une carte annexe) Arduino qui intégrera tous les éléments précédemment validés et retenus lors des tests. Enfin, je devrais faire communiquer ma carte et mon programme avec une application fourni par les élèves d'IR, qui permet de comprendre et utiliser les différentes fonctionnalités du lampadaire.

Travail effectué à se jour :




Lecture du courant
Contrôle du dimmer
Contrôle du relai
Envoi régulier de la valeur du courant
Demande régulière de la valeur du courant
Mode manuel / Contrôle du dimmer
Mode automatique basé sur l'horloge
Renvoie des données demandées
Envoie de la trame à Arduino
Réception des demandes
Envoie de demandes via l'interface IHM et XBee
BASE DE DONNEE
Interface IHM

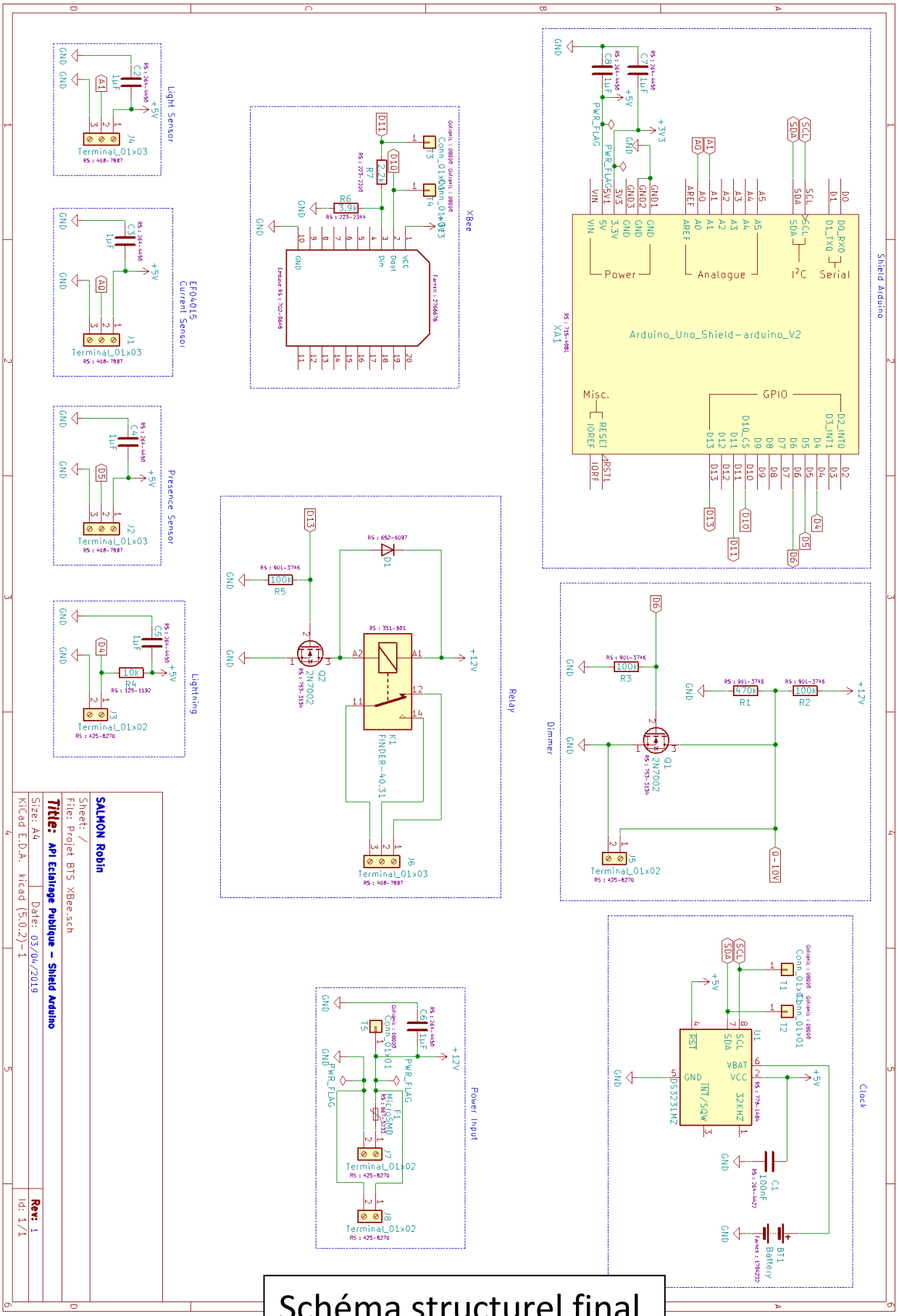


Schéma structurel final

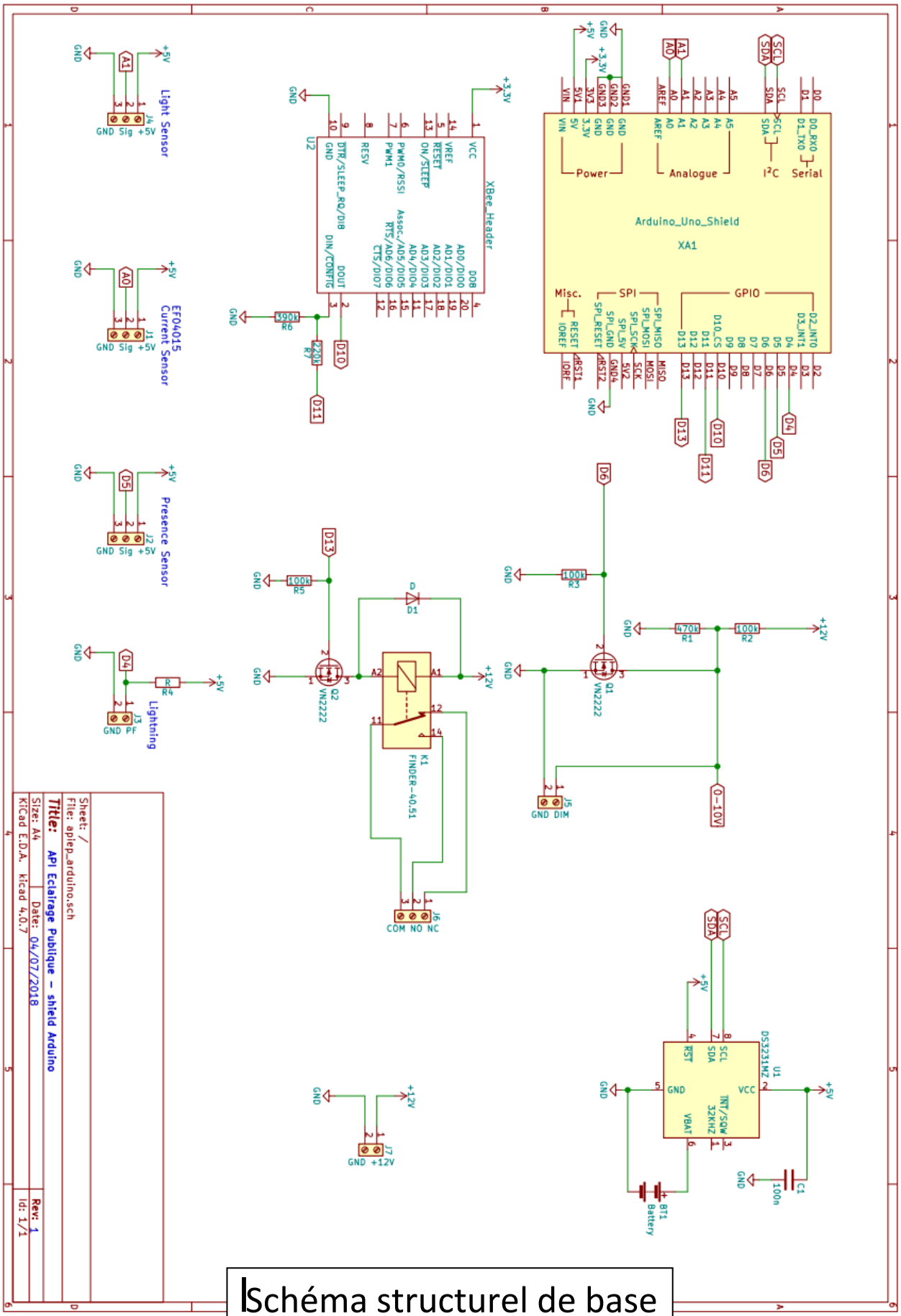


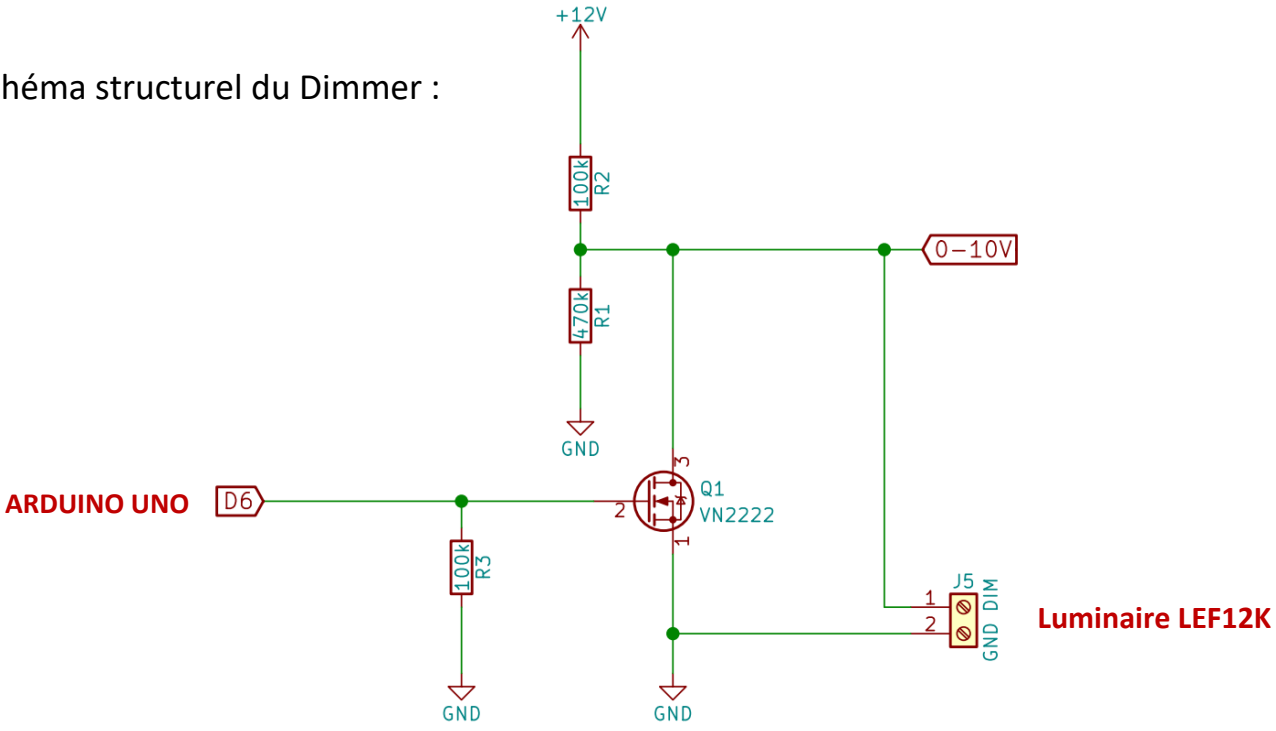
Schéma structurel de base

Travail effectué

A) Les premières recherches :

Afin d'alimenter le « Dimmer » qui va servir à faire varier la luminosité du luminaire (grâce à une tension allant de 0V à 10V), le schéma structurel fourni indiquait comme on le voit ci-dessous une alimentation de 12V. De se fait, il a fallu s'assurer que la carte Arduino peut bel et bien supporter une alimentation de 12V sans risque d'abimer un composant de la carte. En faisant quelques recherches sur la carte Arduino UNO R3, j'ai facilement pu vérifier la tension d'alimentation limite. En effet, une alimentation située entre 7V et 12V est recommandée mais la carte peut supporter des tensions allant de 6V à 20V, ce qui nous permet donc d'injecter sans risque une tension de 12V dans la carte Arduino.

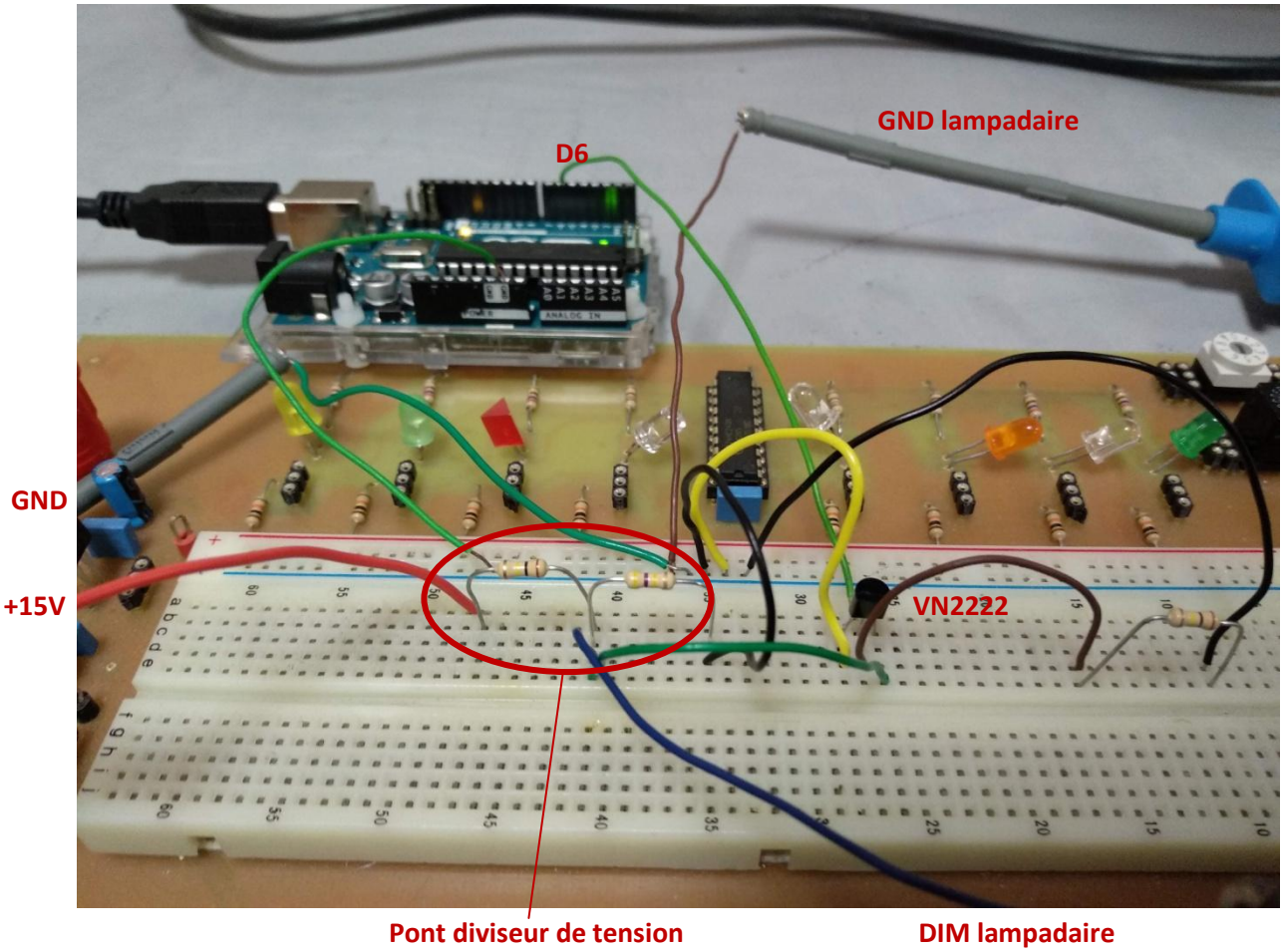
Schéma structurel du Dimmer :



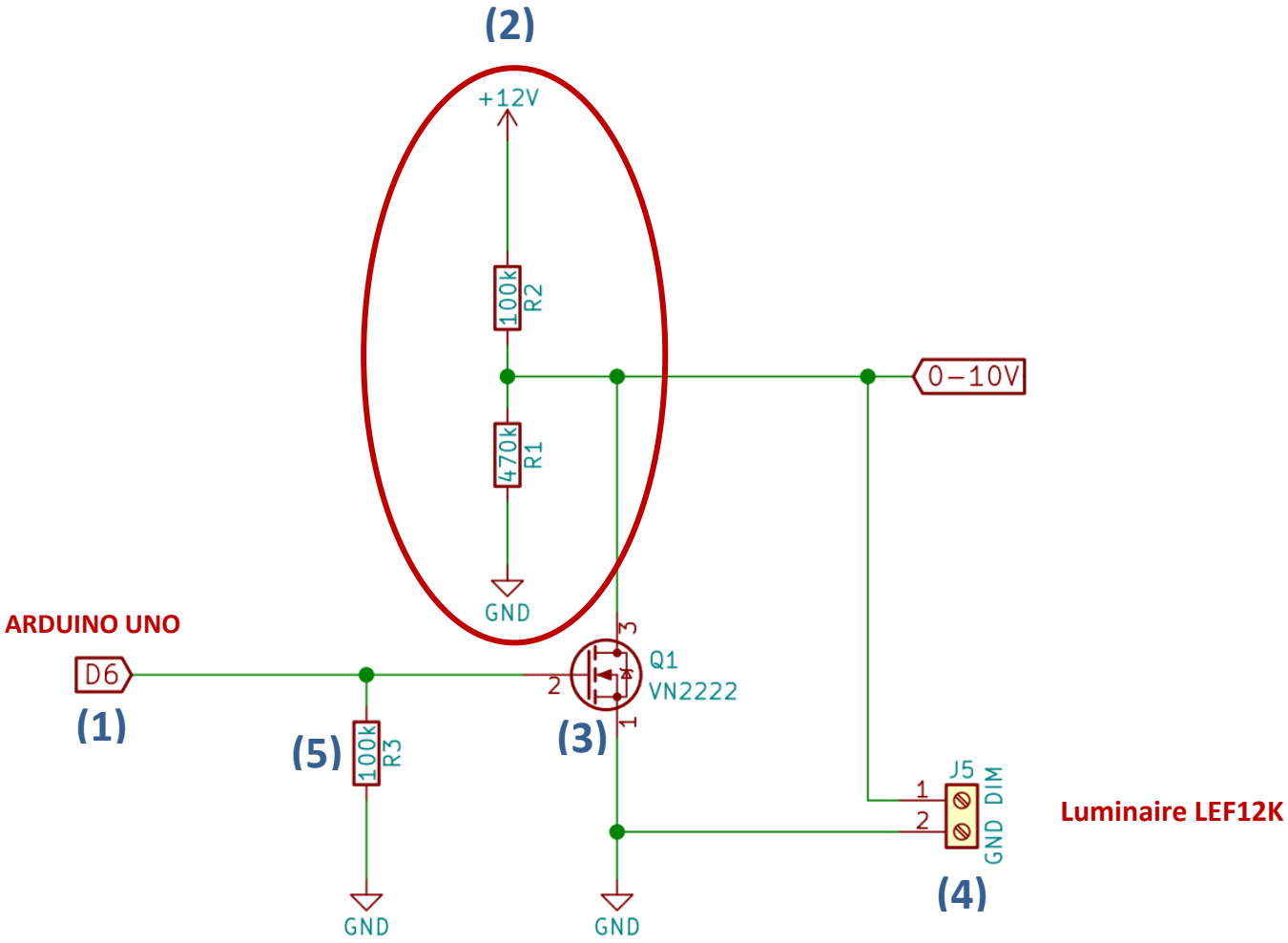
B.1) Câblage et test du Dimmer :

J'ai donc pu dans un deuxième temps faire le câblage du Dimmer accompagné d'une portion de programme, afin de vérifier et valider le câblage donné sur le schéma structurel. Le programme nous permettait de lire une trame de la forme suivante : **« n;DIM;X »**

- **« n »** représentant le numéro d'esclave (en l'occurrence le numéro « 1 » était utilisé dans le cas présent).
- **« DIM »** est la commande utilisée pour contrôler le Dimmer.
- **« X »** est une variable allant de « 0 » % à « 100 » % représentant la luminosité souhaité.
- **« ; »** est le séparateur qui permet au programme de trouver les différentes données de la trame.



B.2) Analyse fonctionnement du Dimmer :



1) La carte Arduino va injecter dans le transistor un signal **D6** PWM (**Pulse Width Modulation ou Modulation de Largeur d'Impulsion**) qui oscille plus ou moins rapidement entre 0 et 5V. Sachant que c'est une sortie numérique et que seulement deux valeurs sont possible : 0V ou 5V ; on utilise pour contrôler le transistor, la valeur moyenne du signal que l'on va faire varier en augmentant ou diminuant le rapport cyclique du signal.

2) On utilise un pont diviseur de tension qui va abaisser la tension d'alimentation qui est à 12V afin de contrôler le luminaire qui lui, nécessite entre 0 et 10V au maximum.

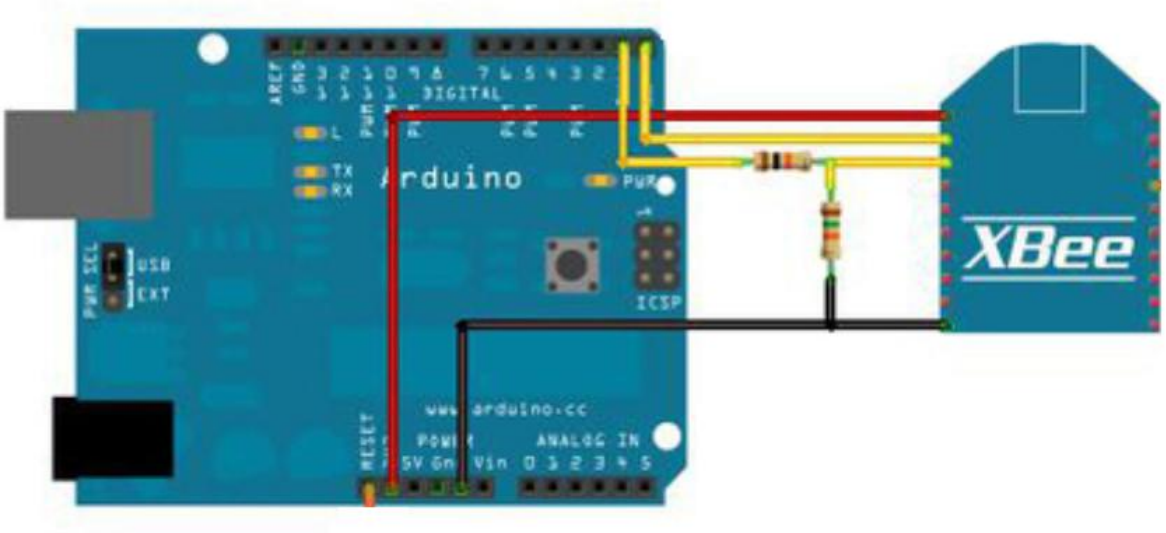
3) Le transistor VN2222 utilisé dans le montage de base (que l'on remplacera par son équivalent, 2N7002) est un transistor MOS de Canal N. Lorsqu'il reçoit une tension de 5V, celui-ci est saturé et donc ramène à la masse la tension du pont diviseur. On aura alors une tension de 0V dans le Luminaire. A l'inverse, lorsque D6 envoi un signal de 0V, le transistor est bloqué et laisse donc passer les 10V qui vont venir alimenter le Luminaire.

4) Le luminaire reçoit une tension de 0 à 10V qui dépend de l'état passant, saturé ou bloqué du transistor.

5) La résistance R3 est une résistance de tirage bas, qui permet en cas de panne ou de problème de la broche **D6** de quand même ramener le potentiel du transistor à 0. De plus, elle permettra de détecter une panne car le transistor étant bloqué à 0V il laisse passer les 10V qui vont venir alimenter le luminaire.

C) Mise en place de XBee sur Arduino :

Dans ce troisième travail, j'ai commencé l'installation d'un module XBee configuré en « esclave » ou « routeur ». Pour commencer, après avoir effectué des recherches sur internet pour m'aider au câblage du module sur la carte Arduino.

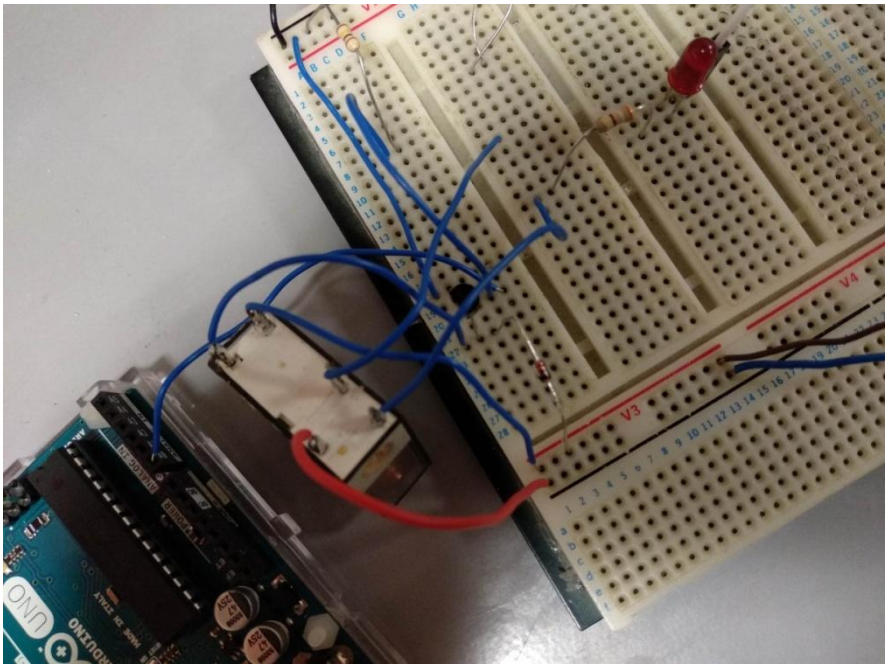


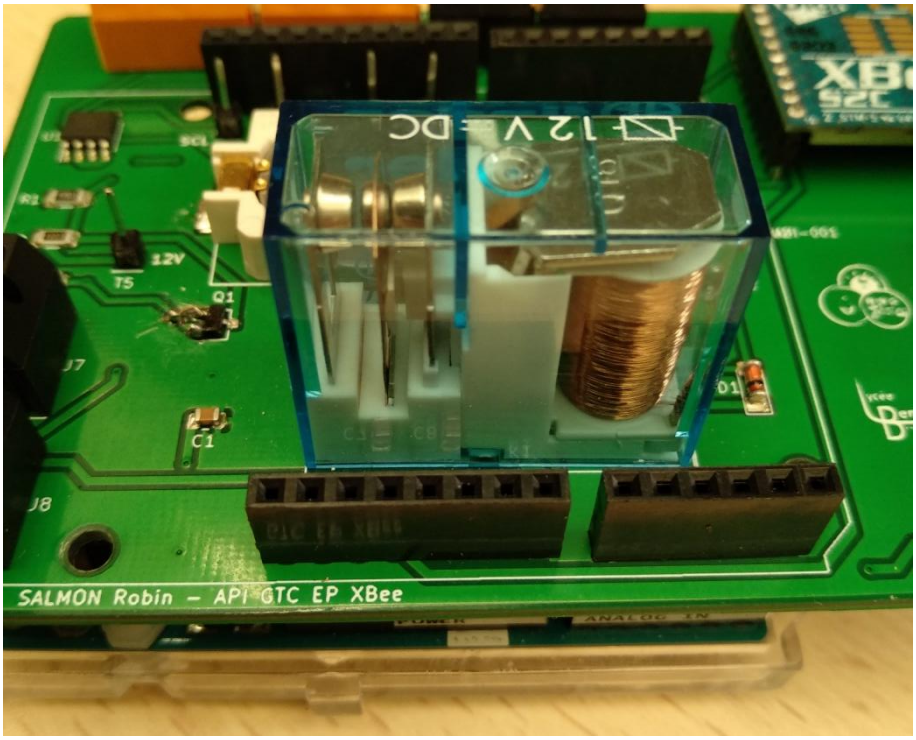
Le câblage et le programme terminé, j'ai pu vérifier grâce aux élèves d'IR que ma carte pouvait bien communiquer avec le module configuré en maitre.

Enfin, en mettant le programme du dimmer en commun avec celui permettant de lire les données reçues par XBee, nous avions la possibilité de contrôler à distance l'éclairage du luminaire.

D) Câblage et test du relais :

La carte doit posséder un relais qui permettra par exemple de brancher des guirlandes aux lampadaires, il a donc fallu tester pour valider le schéma structurel. Le câblage suivant permet grâce à un programme Arduino, d'ouvrir puis fermer le relai à une certaine fréquence pour allumer ou éteindre une led.





E) Programme et test de l'horloge :

Dans mon projet, l'horloge servira à connaitre l'heure et la date afin de programmer un mode automatique. Grace à l'horloge, le programme pourra adapter la luminosité en fonction de la période de l'année et de l'heure du coucher de soleil.

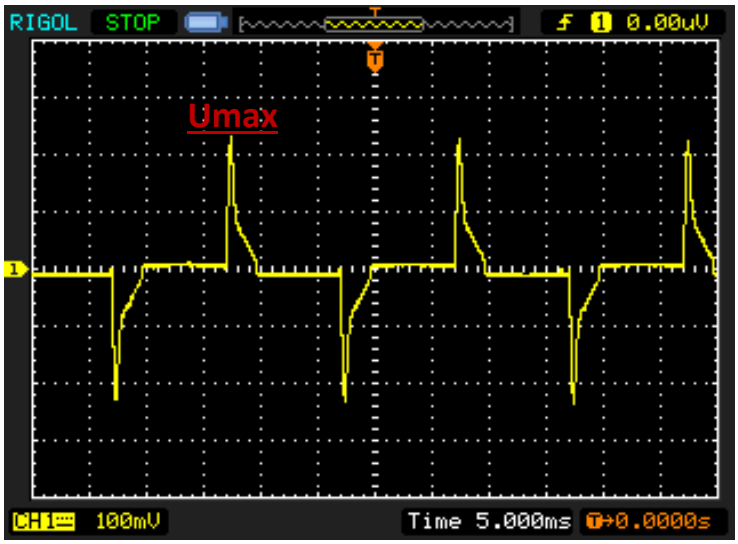
Afin de tester le bon fonctionnement de l'horloge, j'ai utilisé comme base, le programme test fourni avec la librairie Arduino qui permettait simplement d'afficher la date, l'heure et d'autres données comme la température du module. Les données fournies par le programme de base n'étant pas évidentes à comprendre, je l'ai donc modifié et rendu le résultat bien plus facile à lire.

Ci-dessous un extrait du programme permettant l'affichage de l'heure et son résultat dans le moniteur série.

```
if (Clock.getHour(h12, PM) < 10) {
  Serial.print("0");
}
Serial.print(Clock.getHour(h12, PM), DEC);
Serial.print('h');

if (Clock.getMinute() < 10) {
  Serial.print("0");
}
Serial.print(Clock.getMinute(), DEC);
Serial.print('m');
if (Clock.getSecond() < 10) {
  Serial.print("0");
}
Serial.print(Clock.getSecond(), DEC);
Serial.print('s');

Serial.print('\n');

delay(1000);
```

```
06/03/2019 15h01m52s
06/03/2019 15h01m53s
06/03/2019 15h01m54s
06/03/2019 15h01m55s
06/03/2019 15h01m56s
06/03/2019 15h01m57s
06/03/2019 15h01m58s
06/03/2019 15h01m59s
06/03/2019 15h02m00s
06/03/2019 15h02m01s
06/03/2019 15h02m02s
06/03/2019 15h02m03s
06/03/2019 15h02m04s
06/03/2019 15h02m05s
06/03/2019 15h02m06s
06/03/2019 15h02m07s
```

F) Capteur de courant :



Le capteur de courant envoie un signal de la forme suivante :

Afin de connaitre la valeur du courant, il faut dans un premier temps faire un échantillonnage.

```
int sensorValue;             //value read from the sensor
int sensorMax = 0;
uint32_t start_time = millis();
while ((millis() - start_time) < 100) //sample for 100ms
{
  sensorValue = analogRead(CURRENT_PIN);
  if (sensorValue > sensorMax)
  {
    sensorMax = sensorValue;
  }
```

Comme on le voit, cette partie du programme effectue un échantillonnage sur 100ms (5 périodes au total) et va conserver la plus grande valeur (sensorMax) relevée parmi tous les autres points.

```
float amplitude_current = (float)sensor_max / 1024 * 5 / 800 * 2000000;
float effective_value = amplitude_current / 1.414; //minimum_current=1/1024*5/800*2000000/1.414=8.6(mA)
```

Lorsque nous avons relevés sensorMax, qui est compris entre 0 et 1023, il faut convertir cette valeur pour dans un premier temps, obtenir la tension représentative du courant max (amplitude_current = sensorMax / 1024 * 5), puis, convertir cette tension en un courant grâce à la sensibilité du capteur (amplitude_current / 800 * 2000000), donnée dans la documentation.

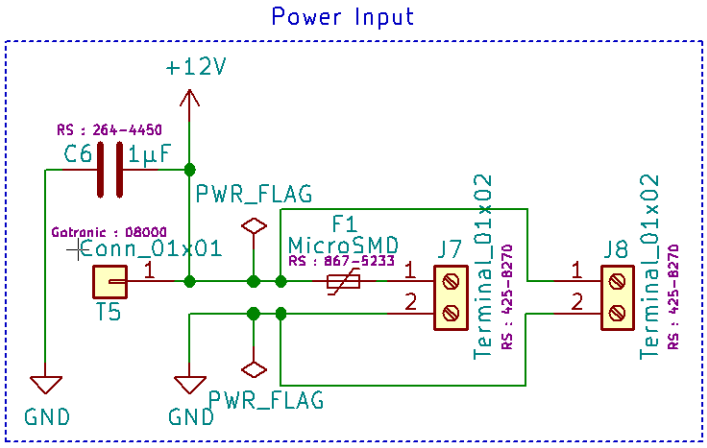
Enfin on convertie cette valeur en courant efficace avec une formule simple de la valeur efficace (effective_value = amplitude_current / √2).

G) Schéma KiCad et routage :

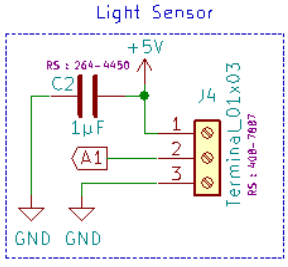
Lorsque chaque futur composants de la carte ont été testés, j'ai pu commencer à créer mon schéma structurel, à identifier tous les composants du schéma et à leurs donner une référence de fournisseur, un modèle 3D et une empreinte.

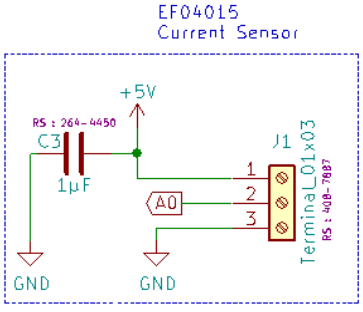
Il y a donc eu un travail de recherche pour trouver les empreintes et model 3D de certains composants comme le module XBee ou le Shield Arduino.

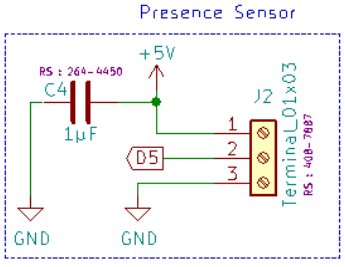
De plus, l'ajout de deux nouveaux borniers permettant d'alimenter la carte Arduino et le Shield Arduino avec la tension d'alimentation de 12V.

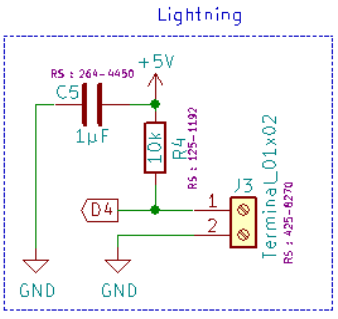


Mais aussi l'ajout d'un condensateur de 1µF aux bornes de chaque bornier, permettant de stocker de l'énergie et fournir plus rapidement un courant à la demande au composant, le temps que la carte Arduino, qui a un temps de réaction plus long, fournisse du courant.



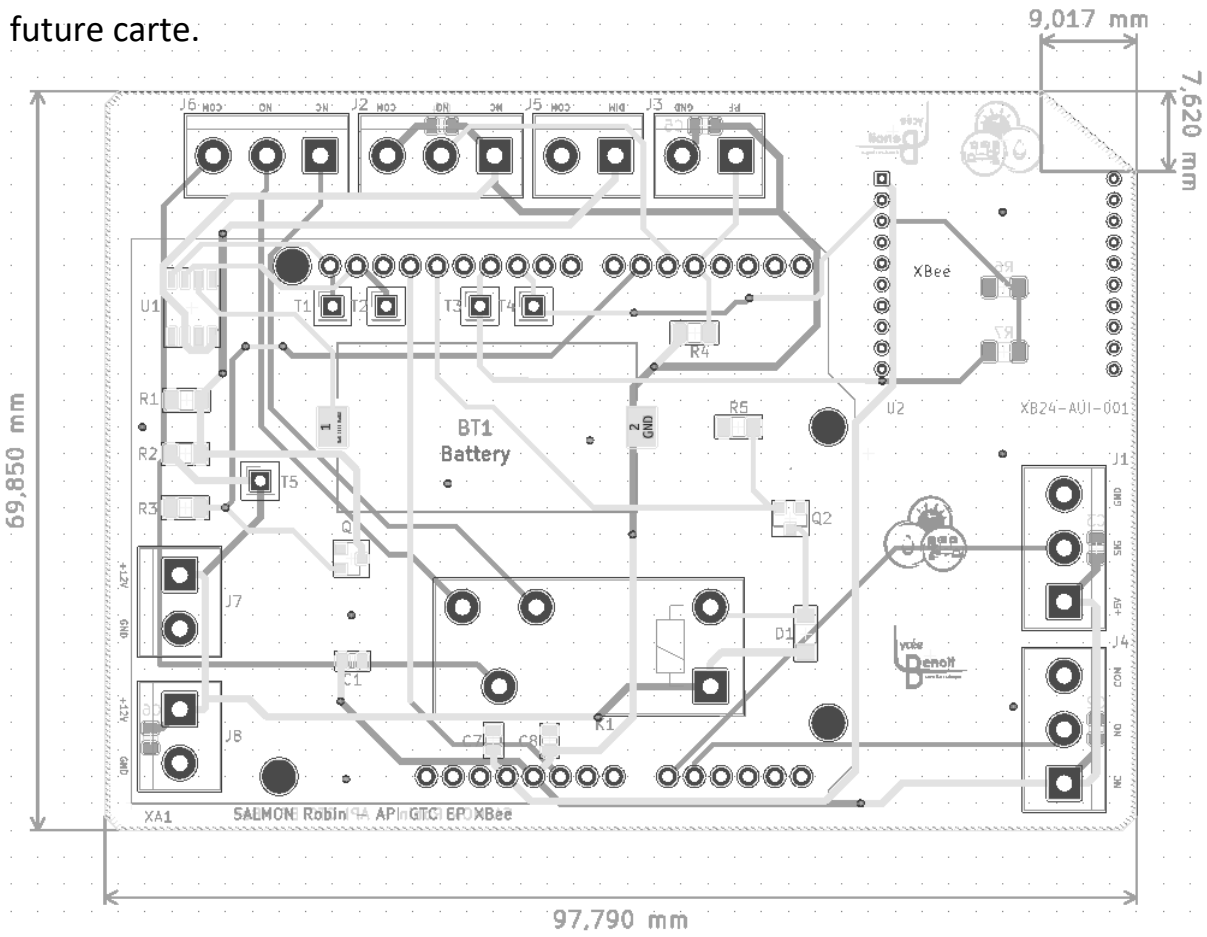


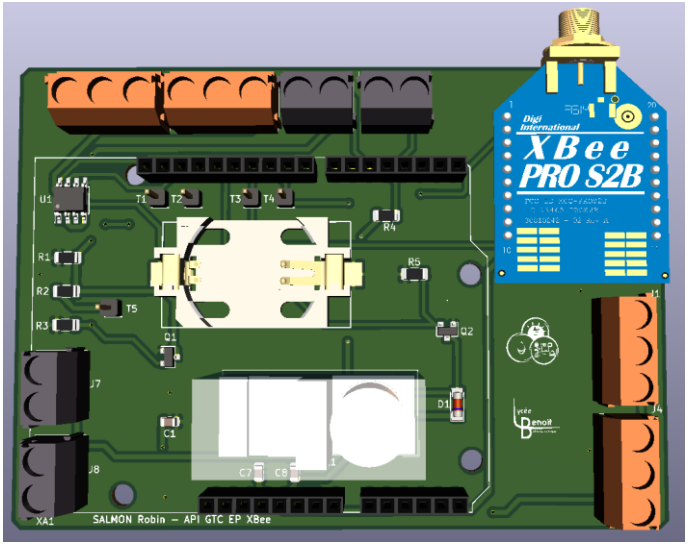


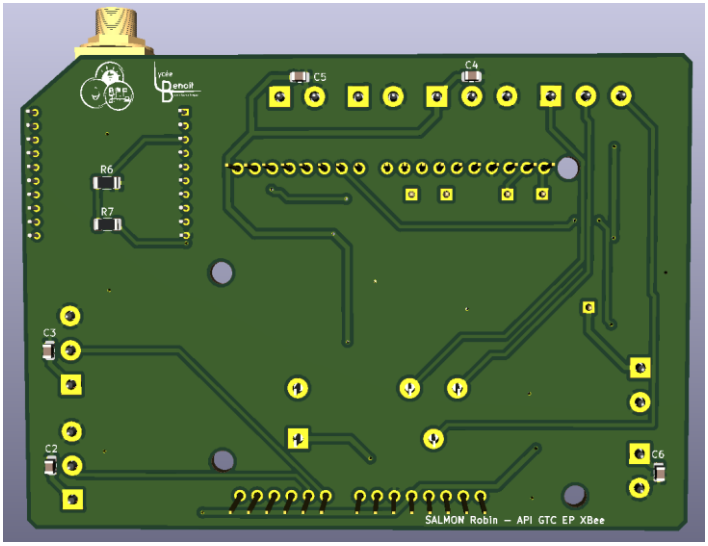


Lorsque tous les composants ont bien étés identifiés et ont leurs empreintes respectives, la partie routage de la carte peut commencer. J'ai donc disposé comme il me semblait le plus pratique, chaque câble et chaque composants, en prenant en compte la taille des composants, la place dont ils disposeront dans la version finale et le fait que le routage doit être le plus ergonomique possible.

On a donc si dessous le schéma de routage KiCad et une représentation 3D de la future carte.



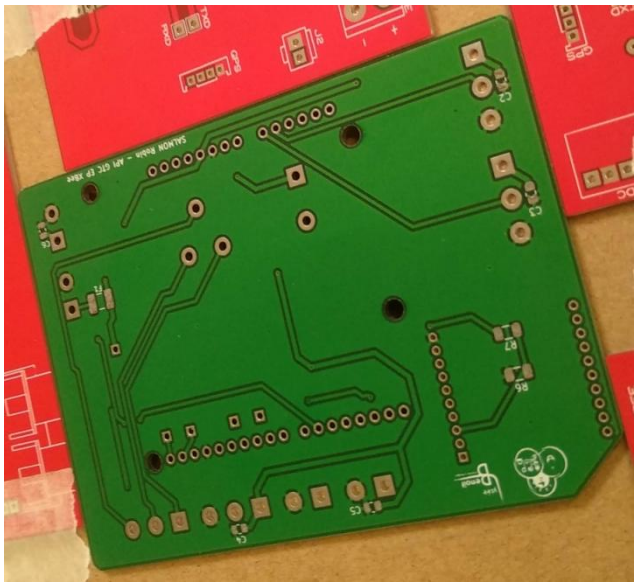




H) Fabrication de la carte :

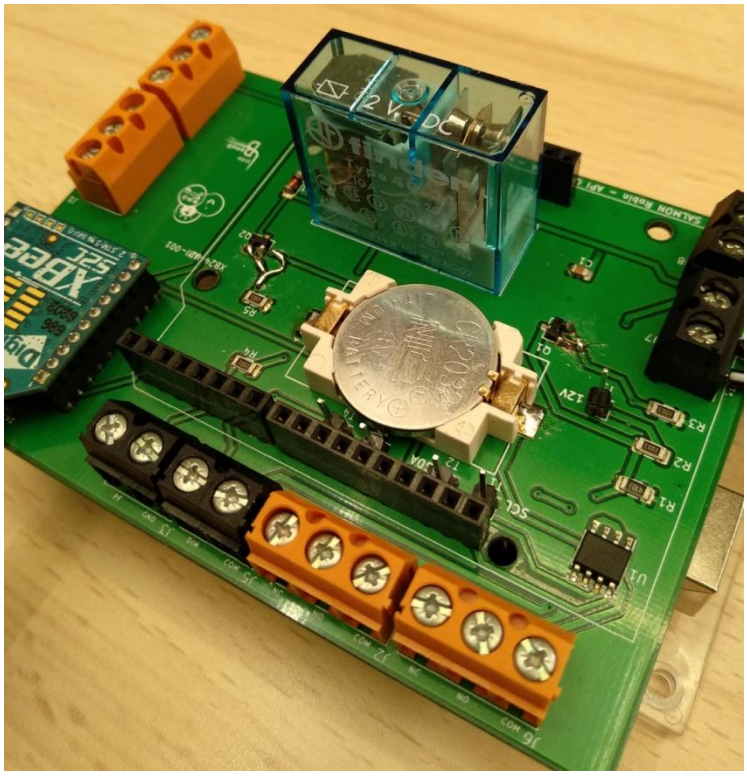
Une fois le routage terminé, il a fallu créer les fichiers Gerber directement à partir du logiciel KiCad ; fichiers à fournir au fabricant JLC PCB, permettant l'impression des PCB (Printed Circuit Board / Circuits Imprimés).

Il faut ensuite appliquer sur tous les emplacements de CMS, une pate abrasive permettant de souder plus facilement les composants CMS de très petite taille.



Lorsque tous les CMS sont sur leur emplacement, il suffit de mettre la carte dans un four à fusion spécial pour souder les CMS.

J'ai donc du par la suite souder à la main le reste des composants pour compléter la carte.



La carte terminée, j'ai pu procéder aux premiers tests afin de vérifier dans un premier temps, que tous les composants sont correctement soudés à leurs emplacements et dans un deuxième temps, les tests de fonctionnement de la carte en action avec un programme Arduino.

I) Test de communication :

Après avoir terminé la carte, et les tests de base, j'ai pu, avec les IR, tester le bon fonctionnement de la communication entre les modules XBee, puis la communication entre l'application de contrôle et le luminaire. On peut donc à présent contrôler le luminaire à distance, avec la carte fabriquée.

A présent, afin de communiquer, les élèves d'IR m'envoient une trame de la forme suivante : **« n;Order;data;Chk\n »**

- **« n »** représentant le numéro d'esclave (en l'occurrence le numéro « 1 » était utilisé dans le cas présent).
- **« Order »** dépend de la commande souhaité (AMP, DIM, REL, RTC, ...).
- **« data »** est une variable qui dépend de l'ordre souhaité (Par exemple pour l'ordre « AMP » on utilise automatiquement la valeur 0).
- **« Chk »** ou **« Check sum »** est une variable qui est calculée par les programmes permettant de vérifier qu'il n'y a pas eu de problème de communication entre les deux cartes.
- **« ; »** est le séparateur qui permet au programme de trouver les différentes données de la trame.
- **« \n »** indique la fin de la trame.

Et ma carte répond avec une trame de la forme suivante :

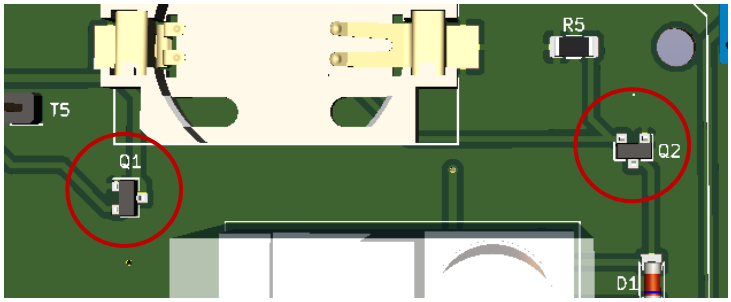
« n;Order;data;timestamp;Chk » ou **« timestamp »** correspond au moment de l'envoie de la réponse.

```
Protocol :
----------
 - orders   : <slaveAddress><sep><order><sep><data><sep><checksum>\n
 - response : <slaveAddress><sep><order><sep><data><sep><timestamp><checksum>\n
```

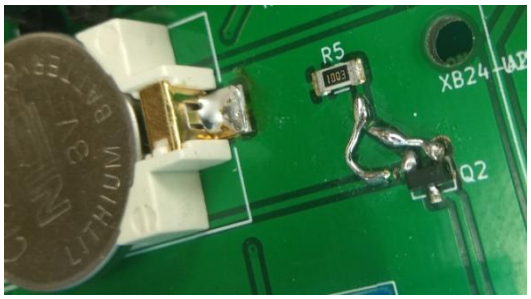
J) Problèmes rencontrés

Durant le projet, j'ai rencontré plusieurs problèmes. Le premier concernait le capteur de courant qui ne fonctionnait pas. En réalité, le problème venait du programme de test utilisé qui faisait une simple mesure ponctuelle, au lieu d'un échantillonnage comme expliqué précédemment. J'ai donc pu vérifier son bon fonctionnement plus tard dans le projet grâce à un oscilloscope qui affichait l'intégralité du signal (c'est à se moment que l'on c'est rendu compte que l'erreur venait en réalité du programme).

Lors des premiers tests de la carte finie, le dimmer et le relai ne fonctionnaient pas, après avoir bien regardé la carte en détail, je me suis rendu compte que l'erreur venait d'un problème d'empreinte sur le schéma KiCad concernant les transistors utilisés. Deux des broches étaient inversés, les transistors ne pouvaient donc pas fonctionner.



Malheureusement, la carte étant déjà fabriquée lorsque j'ai remarqué l'erreur, il était trop tard pour modifier les fichiers KiCad et il a donc fallu corriger l'erreur directement sur la carte.



K) CONCLUSION

Le projet que j'ai fait cette année a déjà été réalisé en partie l'année précédente, j'avais donc l'avantage d'avoir de nombreux documents pour m'aider et un programme arduino déjà fait comme modèle.

Tout de même, le projet était très intéressant de par la diversité des systèmes qu'il inclue et par sa finalité. J'ai également pu m'intéresser à la partie programmation que j'ai du étudier et que j'ai beaucoup apprécié ; grâce à se projet, j'ai pu apprendre à travailler en autonomie avec une équipe, tout en ayant un professeur toujours présent lorsque j'avais un problème ou que j'avais besoin d'une explication sur le projet.

Référence	Valeur	Code commande	Nombre
BT1	Battery	Farnell : 1704232	1
C1	100nF	RS : 264-4422	1
C2 - C8	1µF	RS : 264-4450	7
D1	D	RS : 652-6097	1
F1	MicroSMD	RS : 867-5233	1
J3, J5, J7, J8	Terminal_01x02	RS : 425-8270	4
J1, J2, J4, J6	Terminal_01x03	RS : 408-7887	4
K1	FINDER-40.31	RS : 351-601	1
Q1, Q2	2N7002	RS : 753-3134	2
R1	470k	RS : 223-2619	1
R2, R3, R5	100k	RS : 901-3746	3
R4	10k	RS : 125-1192	1
R6	3.9k	RS : 223-2344	1
R7	2.2k	RS : 223-2300	1
T1 - T5	Conn_01x01	Gotronic : 08000	5
U1	DS3231MZ	RS : 778-1484	1
U2	XB24-AUI-001	Farnell : 2766676	1
XA1	Arduino_Uno_Shield-arduino_V2	RS : 715-4081	1

Suivi Projet

09/01/2019 :

Prise de connaissance du projet

10/01/2019 :

Prise de connaissance et analyse des schémas fournis

11/01/2019 :

- Finition de Grant et installation d'Arduino 1.8.8

- Test de l'alimentation de l'Arduino UNO en 12V (**OK**)

J'ai pu confirmer que la carte Arduino pouvait être alimentée en 12V (6V-20V) On pourra donc utiliser l'alimentation 12V pour alimenter la tête du lampadaire.

- Réflexion, conflit lors d'une alimentation par USB et Jack en même temps (Forum aide) (**OK**)

23/01/2019 :

-Test de câblage de la partie Dimmer

-Programme de test dimmer

24/01/2019 :

-Mise en place de xbee sur arduino

-Contrôle dimmer via xbee par les IR

-Câblage du relais (**NOK ?**)

25/01/2019 :

-Test de relais avec une led et un programme arduino simple (utilisation d'une sortie analogie) (**OK**)

-Installation de KiCad 5.0.2

Suivi Projet (suite)

30/01/2019 :

- Réflexion autour des documents à fournir pour la revue de projet n°1

-Début de la Diapo

31/01/2019 :

-Travail revue de projet partie commune

01/02/2019 :

-Travail revue de projet partie individuelle

Reprogrammation et test du relai avec une led sur une sortie Digitale

Début réflexion horloge

27/02/2019 :

Création d'un programme pour le capteur d'intensité

28/02/2019 :

-Avancement et recherche pour faire fonctionner l'horloge avec arduino

---

01/03/2019 :

-Avancement et recherche pour faire fonctionner l'horloge avec arduino

---

06/03/2019 :

-Programme horloge fonctionnel

-Début de la conception du schéma KiCad et de la recherche des composants nécessaire (Symbole, Model 3D)

Suivi Projet (suite)

07/03/2019 :

Avancement schéma KiCad et mise en œuvre des symboles

08/03/2019 :

Avancement schéma KiCad et creation de la Netliste

13/03/2019 :

Début routage KiCad

14/03/2019 :

Routage KiCad

15/03/2019 :

Routage KiCad

20/03/2019 :

Routage/plan de masse kicad

27/03/2019 :

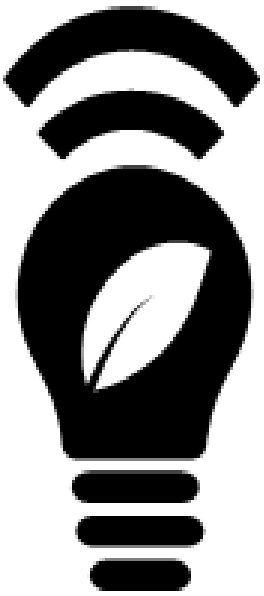
Amélioration schémas KiCad, création liste du materiel

28/03/2019 :

Amélioration schémas KiCad, test de l'ampèremètre pour une éventuelle modification du montage (-220mV donc rien à changer).

Projet API-GTC

Revue 3 - Partie Individuelle



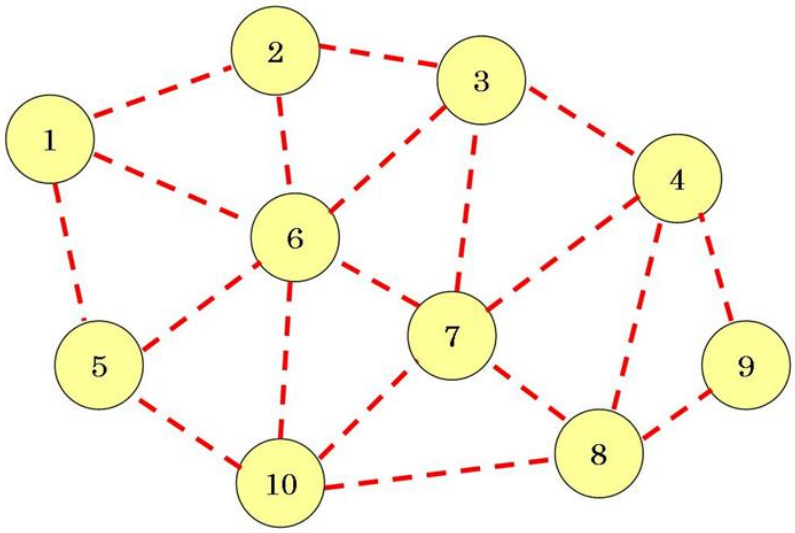
Thomas Weisseldinger IR2

Gestion Technique Centralisée d'un Eclairage Public avec communication XBee

Introduction

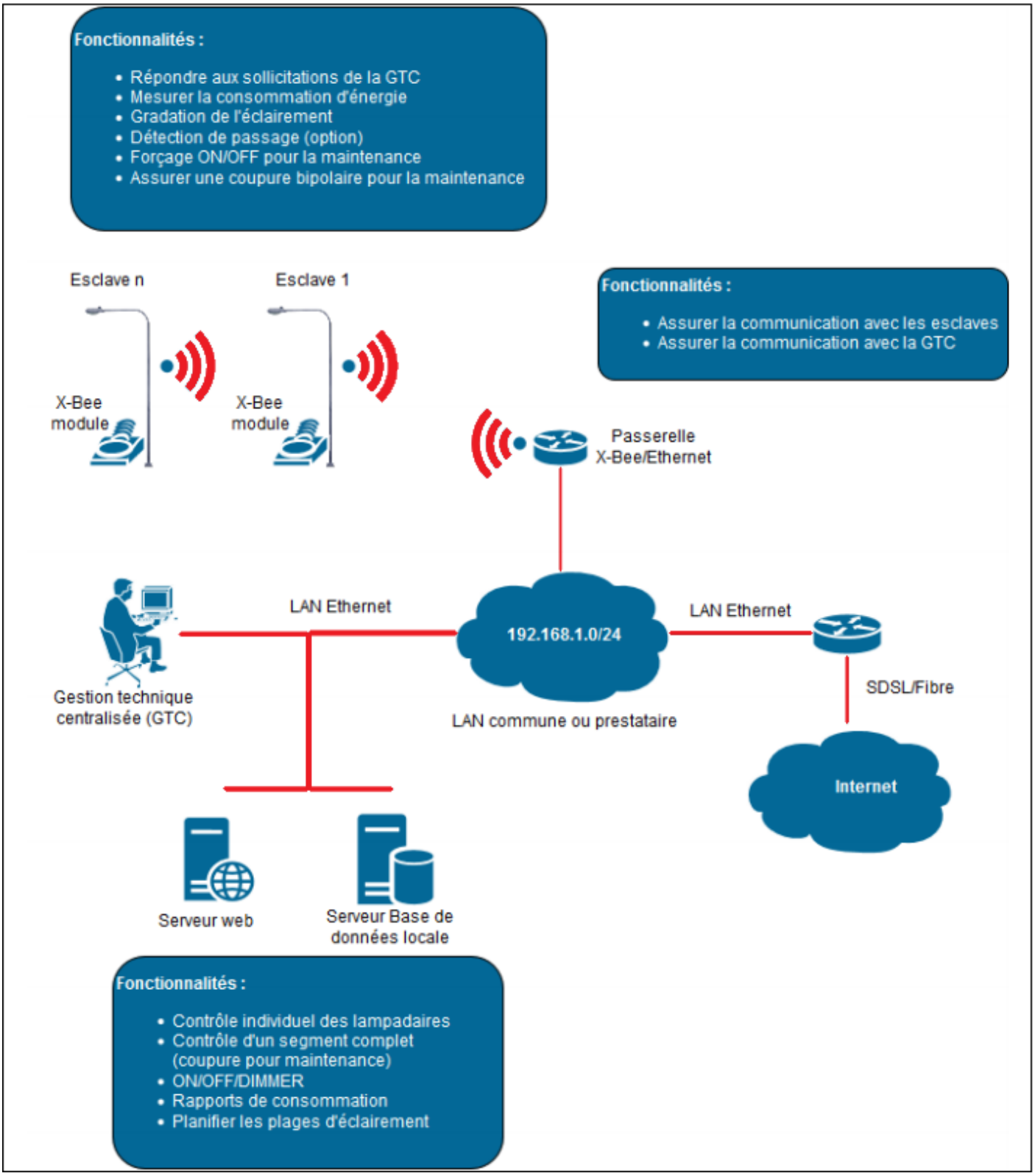
Le projet ci-dessus a été divisé en 3 parties équitables afin de pouvoir effectuer les tâches dans le temps imparti.

La solution nécessite une technologie sans fils qui dispose d'un réseau maillé, ce qui va nous permettre de limiter les interventions qui sont susceptibles de bloquer les communications sur ce réseau, car les informations n'auront qu'à empreinter un autre chemin enfin d'arriver à leurs destinataires.



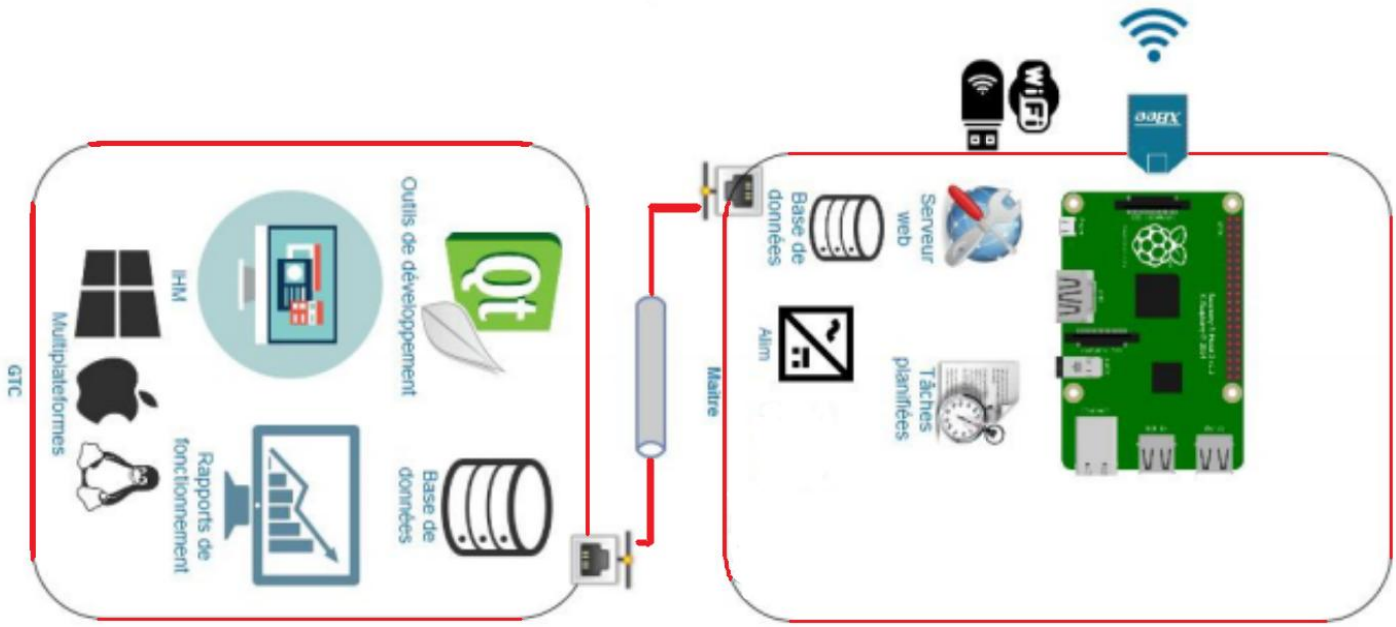
Chaque module a donc une responsabilité afin de recevoir et transmettre des informations quelle qu'elles soient à travers le réseau.

Afin de gérer ce réseau un membre du personnel qualifié devra pouvoir avoir accès au contrôle de ce réseau de lampadaire ainsi qu'à une base de données disponible.


Fonctionnalités :
Répondre aux sollicitations de la GTC
Mesurer la consommation d'énergie
Gradation de l'éclairement
Détection de passage (option)
Forçage ON/OFF pour la maintenance
Assurer une coupure bipolaire pour la maintenance
Esclave n
Esclave 1
X-Bee module
X-Bee module
Fonctionnalités :
Assurer la communication avec les esclaves
Assurer la communication avec la GTC
Passerelle X-Bee/Ethernet
LAN Ethernet
192.168.1.0/24
LAN Ethernet
SDSL/Fibre
Gestion technique centralisée (GTC)
LAN commune ou prestataire
Internet
Serveur web
Serveur Base de données locale
Fonctionnalités :
Contrôle individuel des lampadaires
Contrôle d'un segment complet (coupure pour maintenance)
ON/OFF/DIMMER
Rapports de consommation
Planifier les plages d'éclairement

Partie Personnelle

Étudiant 2 **IR**	*Liste des tâches assurées par l'étudiant* **Le lampadaire maitre** *(Raspberry Pi)* • Configuration du réseau Ethernet/IPV4 • Répondre aux cas d'utilisation "Communiquer avec la GTC", "Sauvegarder l'état des esclaves", "Communiquer avec les esclaves", et "Répondre aux sollicitation de la GTC" avec tous les cas d'utilisation inclus lorsque la sollicitation s'adresse au lampadaire maitre (fonctionnalité esclave).	**Installation** : Environnement de développement Qt Framework Qt/C++ **Mise en œuvre :** **Configuration** : Serveur de base de données Module Xbee maitre **Réalisation** : BDD et tables Logiciel maitre en Qt/C++ Bibliothèque C++ d'accès à la BDD sur RPI Bibliothèque C++ de contrôle d'un lampadaire (Xbee) Dashboard : Etat du segment ou d'un lampadaire sur 24h. **Documentation** : Documentation logicielle Guide d'installation



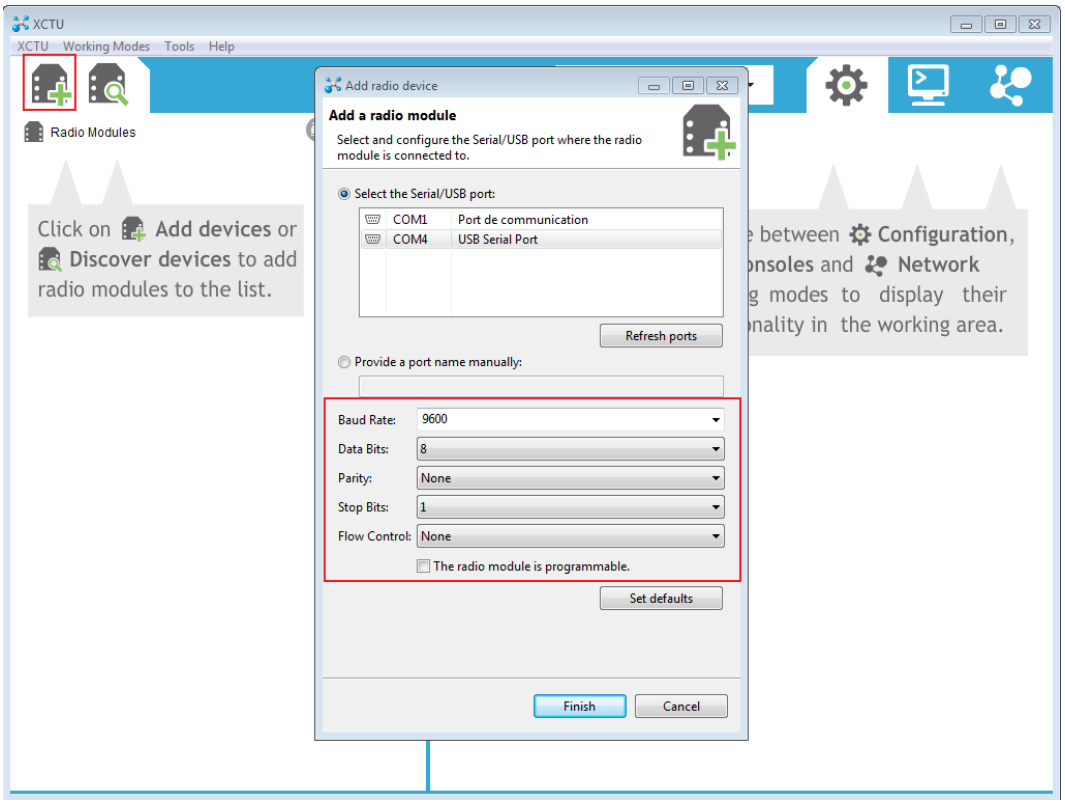
Ma partie va consister a :

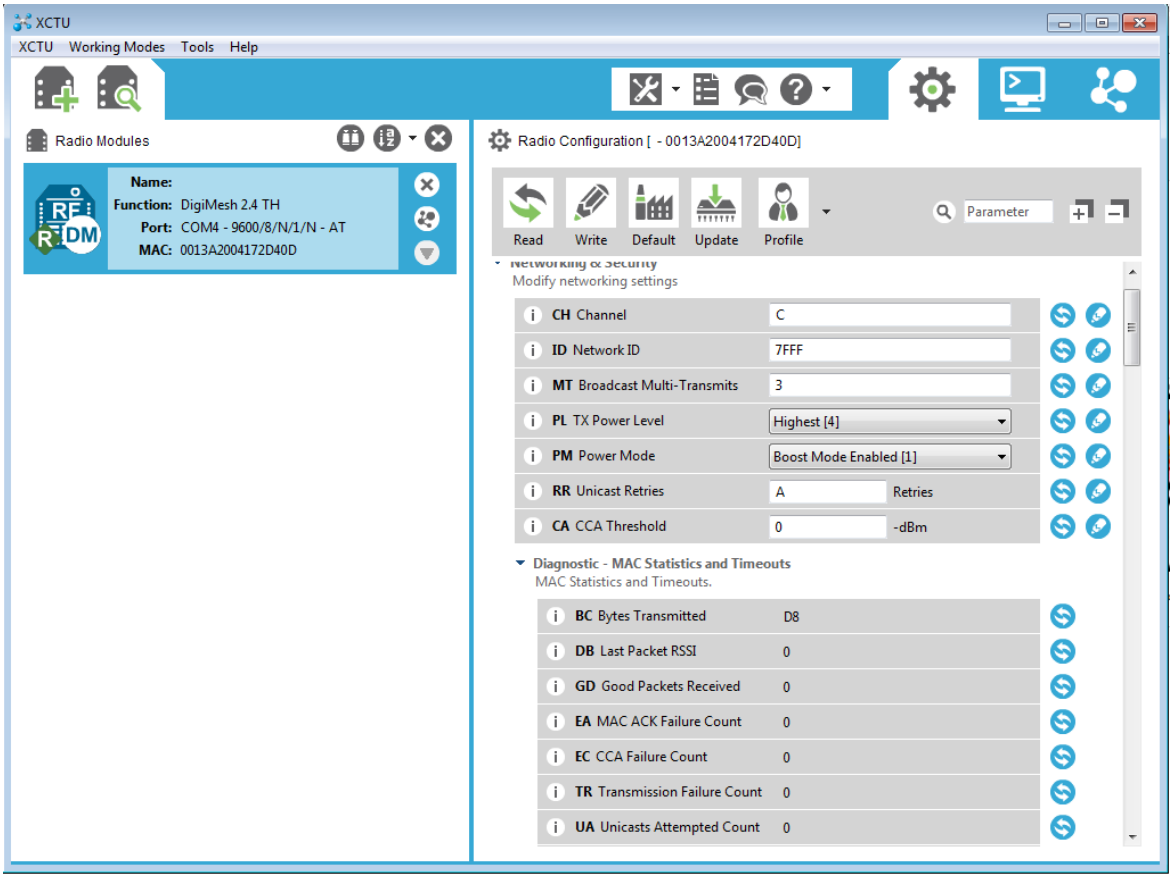
- ➔ **Faire communiquer les modules et créer un réseau stable afin de transmettre convenablement les trames.**
- ➔ **Configurer le réseau Ethernet/IPV4 entre la base de données / GTC / Raspberry disposant du module «Maître».**
- ➔ **Répondre aux sollicitations de la GTC.**
- ➔ **Ainsi que sauvegarder l'état des esclaves**

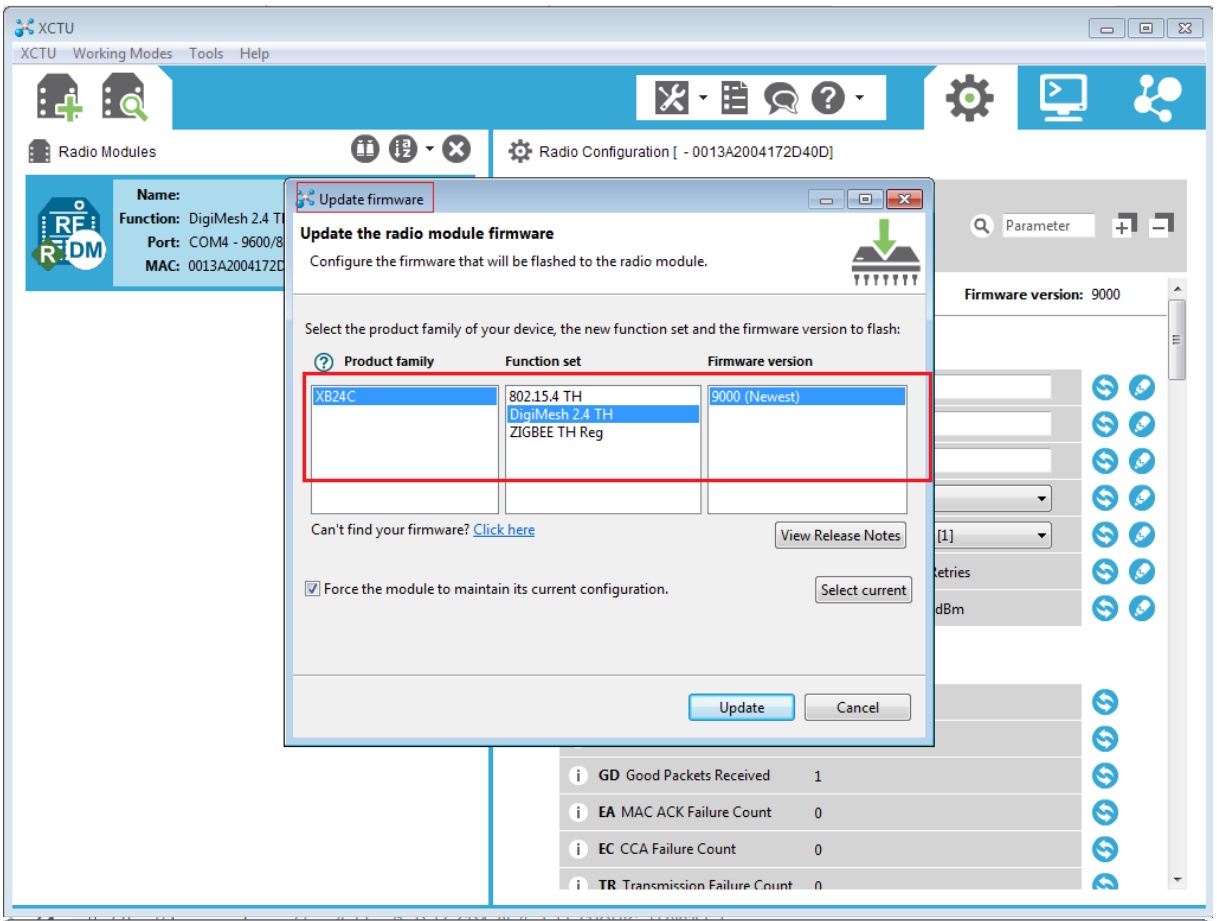
Afin de pouvoir vérifier le bon fonctionnement du matériel utilisé, nous avons effectué cette tâche grâce au logiciel de configuration XCTU qui permet de modifier les paramètres de chaque module individuellement.

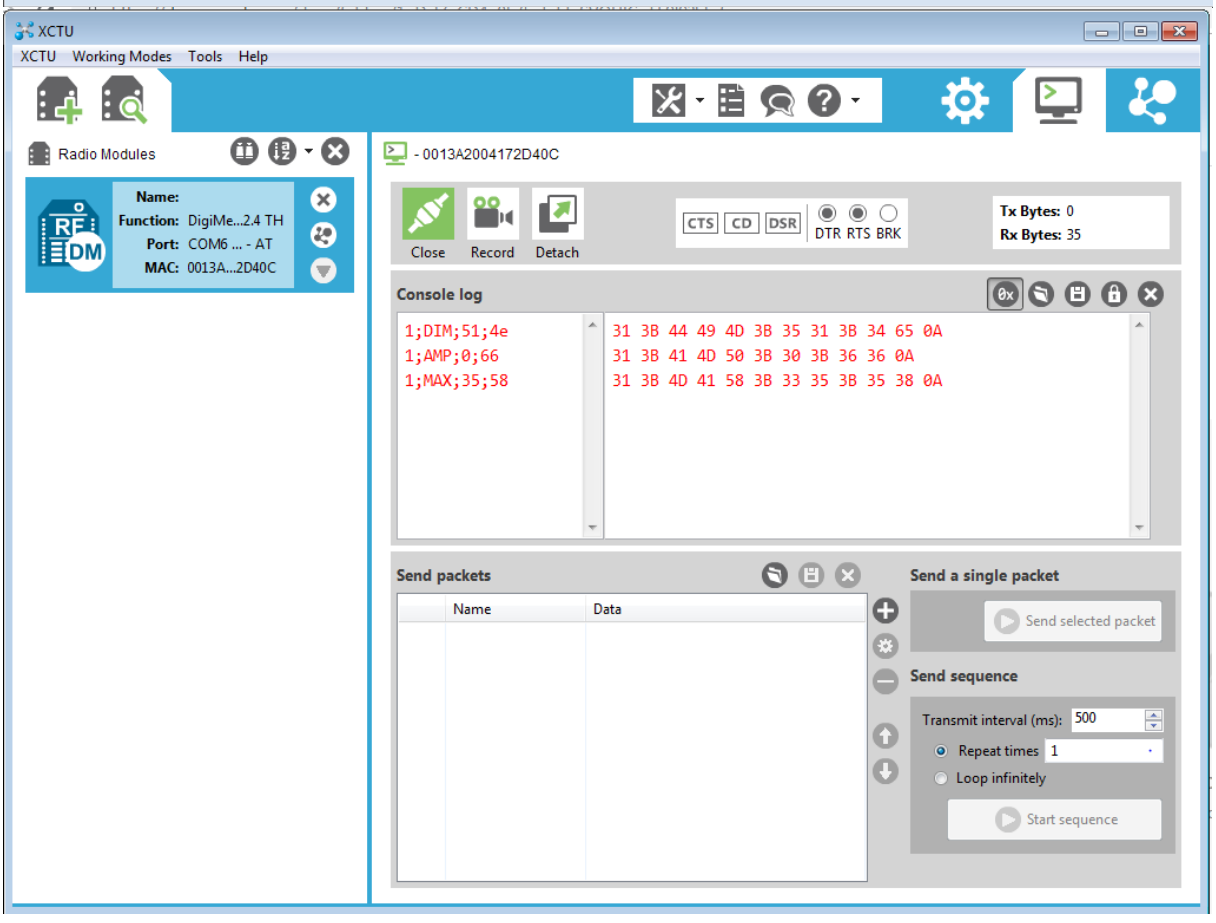
Protocole ZigBee

DIGI XCTU
Configuration & Test
Utility Software


XCTU
XCTU Working Modes Tools Help
Radio Modules
Click on Add devices or Discover devices to add radio modules to the list.
Add radio device
Add a radio module
Select and configure the Serial/USB port where the radio module is connected to.
Select the Serial/USB port:
COM1 Port de communication
COM4 USB Serial Port
Refresh ports
Provide a port name manually:
Baud Rate: 9600
Data Bits: 8
Parity: None
Stop Bits: 1
Flow Control: None
The radio module is programmable.
Set defaults
Finish
Cancel


XCTU
XCTU Working Modes Tools Help
Radio Modules
Name:
Function: DigiMesh 2.4 TH
Port: COM4 - 9600/8/N/1/N - AT
MAC: 0013A2004172D40D
Radio Configuration [- 0013A2004172D40D]
Read
Write
Default
Update
Profile
Parameter
Modify networking settings
CH Channel C
ID Network ID 7FFF
MT Broadcast Multi-Transmits 3
PL TX Power Level Highest [4]
PM Power Mode Boost Mode Enabled [1]
RR Unicast Retries A Retries
CA CCA Threshold 0 -dBm
Diagnostic - MAC Statistics and Timeouts
MAC Statistics and Timeouts.
BC Bytes Transmitted D8
DB Last Packet RSSI 0
GD Good Packets Received 0
EA MAC ACK Failure Count 0
EC CCA Failure Count 0
TR Transmission Failure Count 0
UA Unicasts Attempted Count 0


XCTU
XCTU Working Modes Tools Help
Radio Modules
Radio Configuration [- 0013A2004172D40D]
Name:
Function: DigiMesh 2.4 TH
Port: COM4 - 9600/8
MAC: 0013A2004172D
Update firmware
Update the radio module firmware
Configure the firmware that will be flashed to the radio module.
Select the product family of your device, the new function set and the firmware version to flash:
Product family
Function set
Firmware version
XB24C
802.15.4 TH
DigiMesh 2.4 TH
ZIGBEE TH Reg
9000 (Newest)
Can't find your firmware? Click here
View Release Notes
Force the module to maintain its current configuration.
Select current
Update
Cancel
Parameter
Firmware version: 9000
GD Good Packets Received 1
EA MAC ACK Failure Count 0
EC CCA Failure Count 0

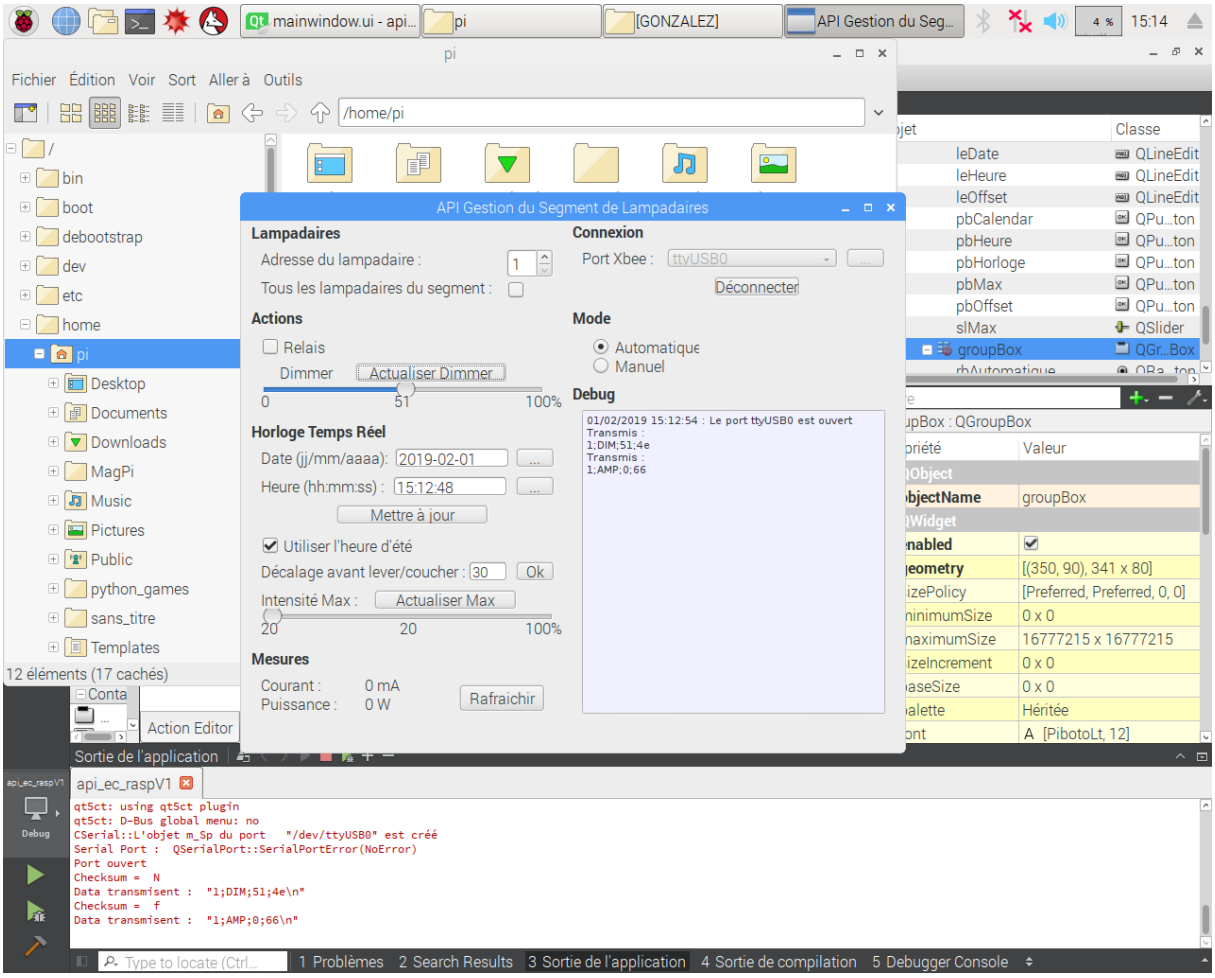

XCTU
XCTU Working Modes Tools Help
Radio Modules
- 0013A2004172D40C
Name:
Function: DigiMe...2.4 TH
Port: COM6 ... - AT
MAC: 0013A...2D40C
Close
Record
Detach
CTS CD DSR
DTR RTS BRK
Tx Bytes: 0
Rx Bytes: 35
Console log
1;DIM;51;4e
1;AMP;0;66
1;MAX;35;58
31 3B 44 49 4D 3B 35 31 3B 34 65 0A
31 3B 41 4D 50 3B 30 3B 36 36 0A
31 3B 4D 41 58 3B 33 35 3B 35 38 0A
Send packets
Name
Data
Send a single packet
Send selected packet
Send sequence
Transmit interval (ms): 500
Repeat times 1
Loop infinitely
Start sequence

51 > dimer en %

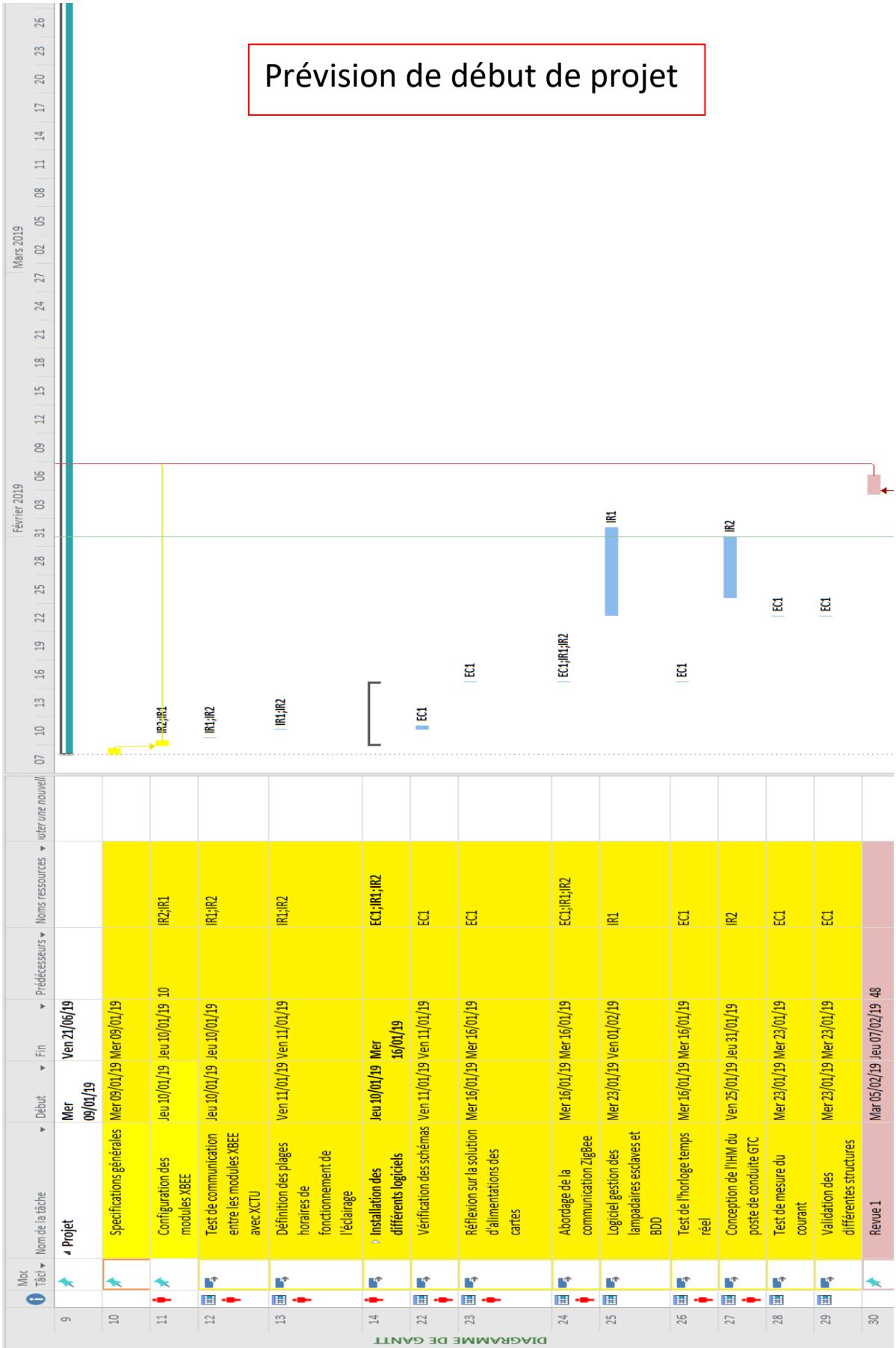
35> intensité lumineuse en %

4^{e} > adresse

Ceci sont des trames définies afin de communiquer correctement avec les lampadaires esclaves.



Prévision de début de projet



Installer InfluxDB :

curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add –

Je commence par télécharger le service InfluxDB en indiquant la position de la clef

apt-get upgrade

sudo apt install apt-transport-https

echo "deb https://repos.influxdata.com/debian jessie stable" | sudo tee /etc/apt/sources.list.d/influxdb.list

sudo apt-get update

sudo apt-get install influxdb

Puis j'installe le service en question en effectuant un update.

sudo nano /etc/influxdb/influxdb.conf

En utilisant le chemin je vais ouvrir le fichier influxdb.conf afin de voir le port ou encore les possibilités de sécuriser le système.

bind-address = ":8086"
auth-enabled = false

Puis en effectuant un redémarrage.

sudo service influxdb restart

Je fini par tester l'installation

```
$ curl -sl -I localhost:8086/ping


HTTP/1.1 204 No Content
Content-Type: application/json
Request-Id: 165b0fa4-ad88-11e6-8a6f-000000000000
X-Influxdb-Version: v1.1.0
Date: Frid, 22 Feb 2019 15:11:08 UTC
```

Ce qui me confirme le bon démarrage du serveur

Influx –precision rfc3339

Afin de se connecter à InfluxDB, ce qui me donne :

Conected to http://localhost:8086 version 1.7.4

InfluxDB shell version: 1.7.4

Enter an InfluxXQL query

Une fois dans InfluxDB j'effectue un :

>show databases

Ce qui va m'afficher les différentes bases et tables actuelles

Afin de prendre InfluxDB en main je commence par utiliser la base « Telegraf ».

>use telegraf

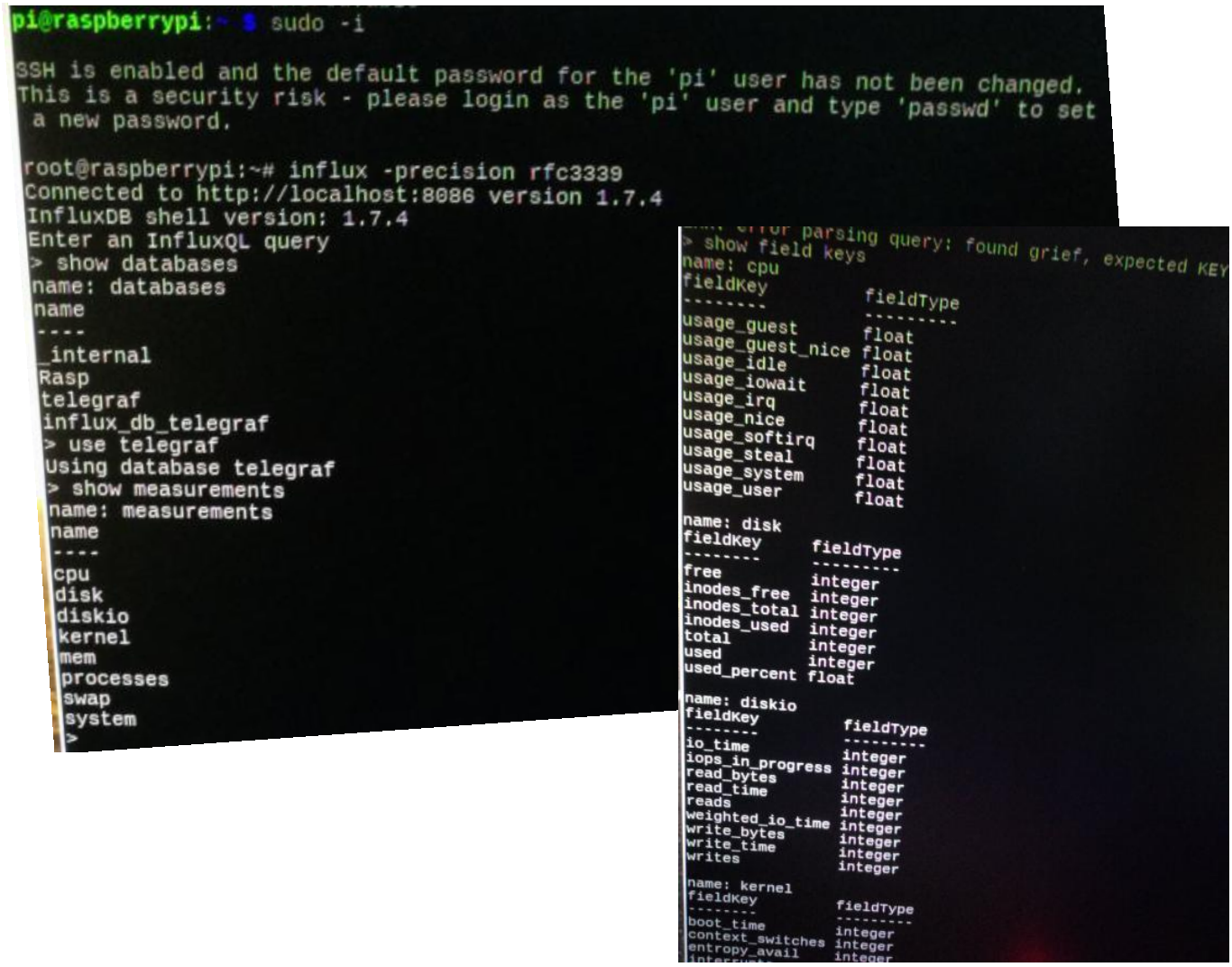
Using database telegraf

Pour continuer je vais chercher les différentes tables de telegraf en entrant :

>show measurements

Ce qui me donne les tables de telegraf, qui correspondent bien avec les tables affichées avec Grafana. Pour finir je vais aller regarder les keys ou types de variable des données.

>show field keys



sudo –i

sudo apt-get install apt-transport-https curl

curl https://bintray.com/user/downloadSubjectPublicKey?username=bintray | sudo apt-key add –

echo "deb https://dl.bintray.com/fg2it/deb stretch main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

sudo apt-get update

sudo apt-get install grafana

sudo service grafana-server restart

(étant déjà en « sudo –i » les sudo ne sont pas obligatoires)
Avec cette commande je commence par installer les paquets puis j'installe le service Grafana tout en gardant le tout à jour.
Enfin j'utilise :

sudo nano /etc/grafana/grafana.ini

Afin d'obtenir le fichier grafana.ini a travers le chemin /etc/grafana/grafana.ini le ou je pourrais avoir accès a certaine information tel que le port

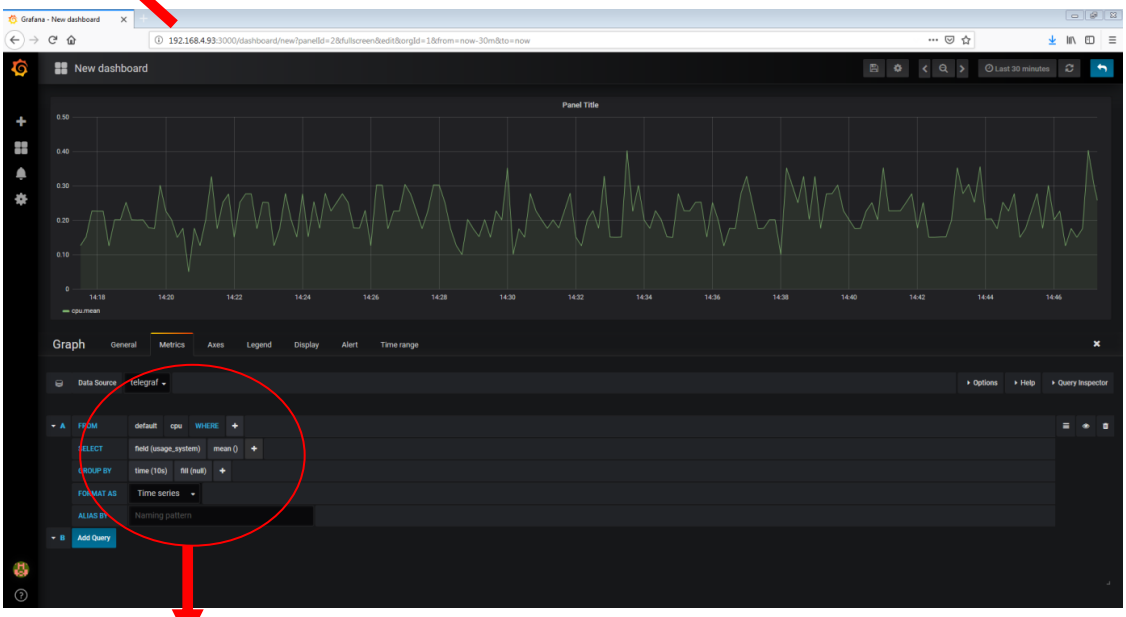
http_port = 3000

Name : admin

Mdp : admin

ip de l'hôte

Port attribué



Configuration du flux d'information de Telegraf vers l'affichage de Grafana

- Sélectionner Telegraf
- Sélectionner la ou les donnée(s)à afficher
- Sélectionner le temps de rafraichissement

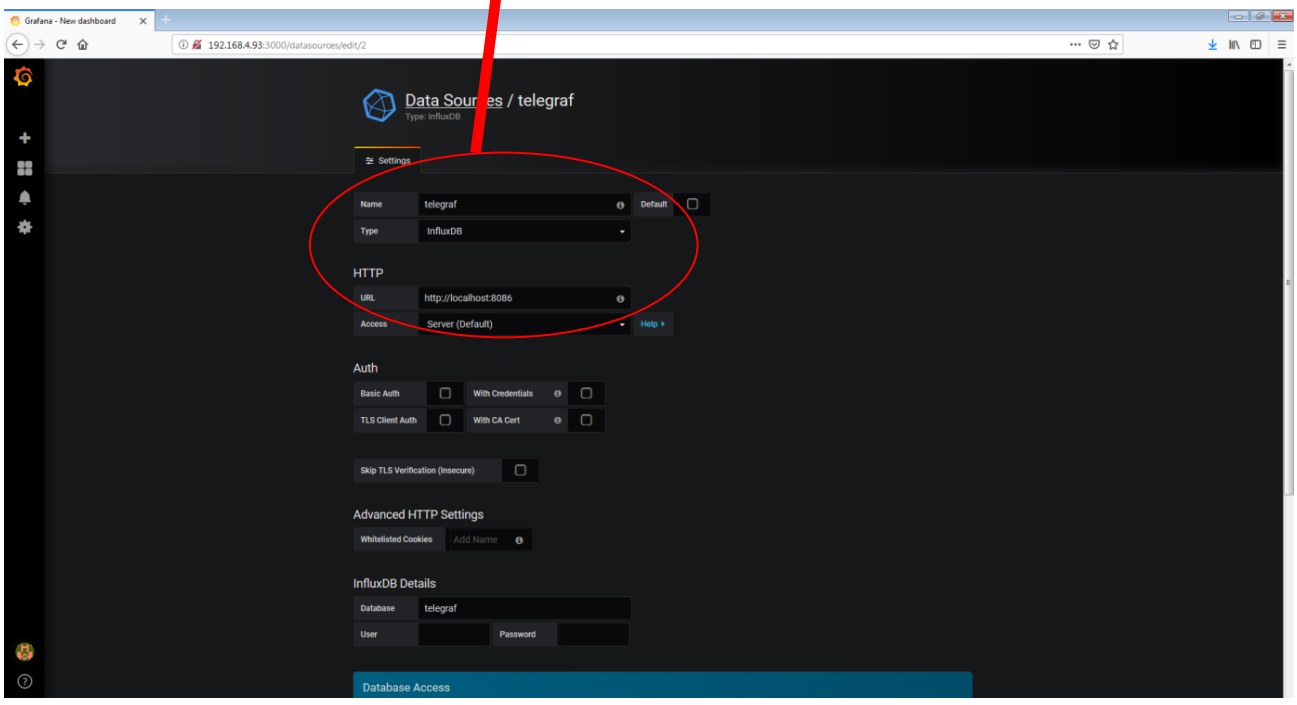
Indication donné à Grafana afin qu'il puisse trouver le chemin de la base de données grâce au type de BDD, au nom de celle-ci ainsi qu'à son adresse et son port.

Name : Telegraf

Type : InfluxDB

URL : http://localhost:8086

En l'occurrence ma base donnée est directement sur ma machine.



Grâce à cela nous pouvons voir qu'InfluxDB fonctionne correctement.

Je commence par installer le service en question :

Sudo -i

Sudo apt-get update

Sudo apt-get install mosquitto-clients

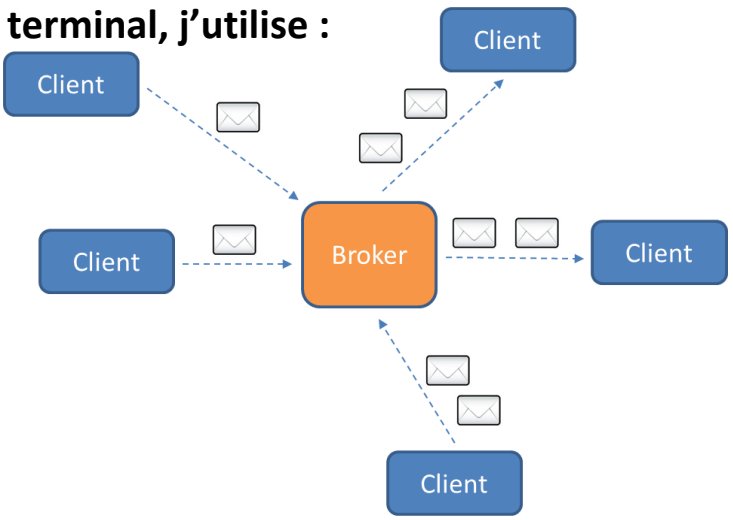
Une fois l'installation fini

Je créer un « topic » avec le nom choisi :

mosquitto_pub –h localhost –t XbeeUP –m ...

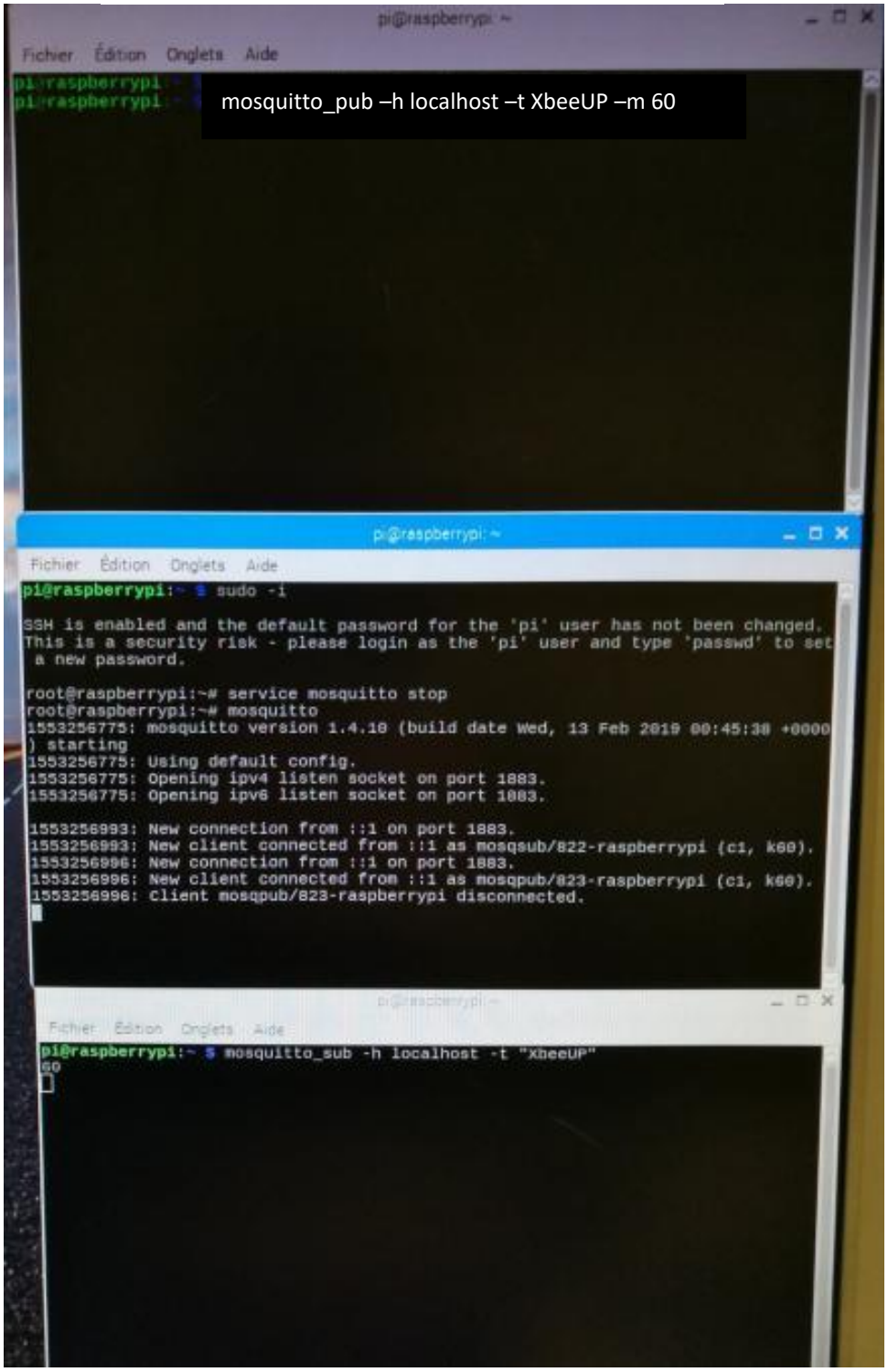
Le broker étant prêt (les "..." étant une valeur voulue envoyée du serveur –h (localhost) du topic –t (XbeeUP). Enfin le –m signifie l'envoie du message) je peux alors vérifier la bonne réception du message en local en ouvrant un autre terminal, j'utilise :

mosquitto_sub –h localhost –t ''XbeeUP''



Pour terminer, le destinataire aura juste à changer le localhost par l'adresse réseau du serveur afin de recevoir ces informations.


mosquitto
MQTT.ORG


pi@raspberrypi: ~
Fichier Édition Onglets Aide
mosquitto_pub –h localhost –t XbeeUP –m 60
pi@raspberrypi: ~
Fichier Édition Onglets Aide
pi@raspberrypi:~ $ sudo -i
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.
root@raspberrypi:~# service mosquitto stop
root@raspberrypi:~# mosquitto
1553256775: mosquitto version 1.4.10 (build date Wed, 13 Feb 2019 00:45:38 +0000
) starting
1553256775: Using default config.
1553256775: Opening ipv4 listen socket on port 1883.
1553256775: Opening ipv6 listen socket on port 1883.
1553256993: New connection from ::1 on port 1883.
1553256993: New client connected from ::1 as mosqsub/822-raspberrypi (c1, k60).
1553256996: New connection from ::1 on port 1883.
1553256996: New client connected from ::1 as mosqpub/823-raspberrypi (c1, k60).
1553256996: Client mosqpub/823-raspberrypi disconnected.
Fichier Édition Onglets Aide
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t "XbeeUP"
60

C++

L'application développée sous QtCreator a pour but d'effectuer la formation de la trame et son envoie a travers le réseau xBee. Elle sera alors déployée sur une raspberry connectée au module Maître afin de pouvoir communiquer aux autres routeurs les ordres à exécuter. Ces ordres auront été décidés avant par le logiciel de la GTC. Afin d'approfondir cette application une partie du code sera alors expliquée si dessous.

```
int MainWindow::envoyerCommande()
{

    commande.append(mAdresse);
    commande.append(SEPARATEUR);
    commande.append(/*mOrdre*/"DIM");
    commande.append(SEPARATEUR);
    commande.append(mData);
    commande.append(SEPARATEUR);
    QChar cs = calculChecksum(commande);
    QString csStr;
    csStr.setNum(cs.unicode(),16);
    commande.append(csStr);
    commande.append("\n");

    if(mXbee->ecrire(commande.toStdString().c_str()))
    {
        qDebug() << "Data transmisent : " << commande;

        influxdb_cpp::server_info si_new("192.168.4.34", 8086,
"bddgtc", "", "");

        string mStringAdresse = mAdresse.toStdString();
        int mIntAdresse = mAdresse.toInt();
        string mStringData = mData.toStdString();
        float mFloatData = mData.toFloat();
mStringAdresse +",flag=1 value="+ mStringData +" "+ mStringTime +"",
si_new);
```

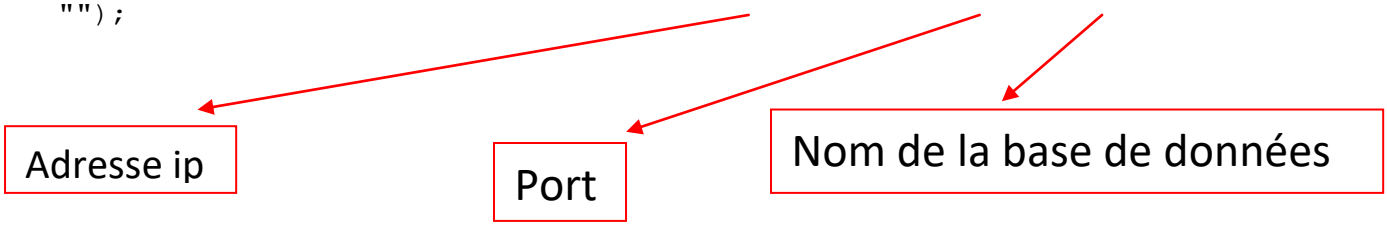
```
        int ret = influxdb_cpp::builder()
            .meas("dim")
            .tag("flag", "1")
            .tag("add",mStringAdresse)
            .field("value", mFloatData)
            .post_http(si_new, &resp);
        cout << ret << endl << resp << endl;
        cout << "insert fait" << endl;

        string query = "DELETE FROM dim WHERE flag='0' AND add = '"+
mStringAdresse +"' AND time="+ mTime +"";
       cout << "query: " << query << endl;
        influxdb_cpp::query(resp,query ,si_new);
        cout << resp << endl;


        commande.clear();
        return 1;
    }
    commande.clear();

    return 0;
}
```

Ceci est le code afin de pouvoir former la trame avec les informations préalablement récupérés dans la base de données de la GTC. Suite a l'envoie de la trame une modification est alors effectuée dans cette même base de données afin d'afficher cette trame en tant que « faite ».

Le 'Flag' étant le ''trigger'' que nous utilisons.

Le chemin de la base est alors requis avec cette ligne de code ci :

```
 influxdb_cpp::server_info si_new("192.168.4.34", 8086, "bddgtc", "",
"");
```



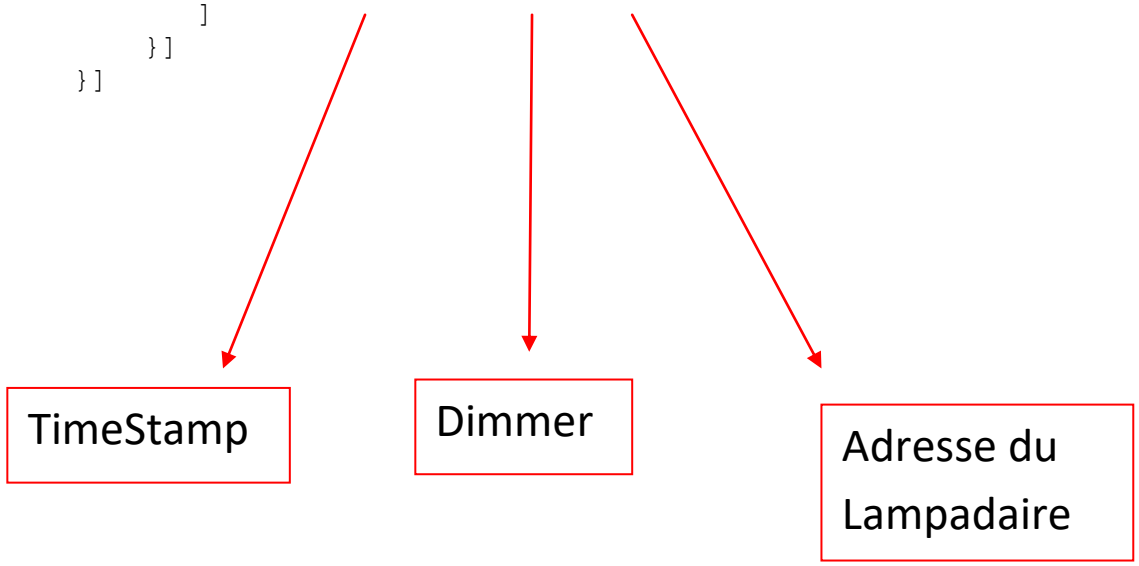
Les 2 dernières places sont le 'usr' ainsi que le 'mdp' de la bdd si il en détient un

Puis le code ci-dessous :

```
void MainWindow::onTimerOut(){  // ExÃ©cutÃ© par le timer

        influxdb_cpp::server_info si_new("192.168.4.34", 8086,
"bddgtc&epoch=ns", "", "");
        influxdb_cpp::query(resp, "SELECT value, add FROM dim WHERE
flag='0'", si_new);


       char * str = (char *) malloc(resp.length() + 1); // +1 pour
le '\0'
       strcpy(str, resp.c_str());

        QJsonDocument json =
QJsonDocument::fromJson(QByteArray(resp.c_str()));
        QJsonObject jo = json.object();
        QJsonArray results = jo["results"].toArray();
        QJsonValue jv = results[0];
        QJsonObject jo2 = jv.toObject();
        QJsonArray series = jo2["series"].toArray();
        QJsonObject jo3 = series[0].toObject();
        QJsonArray values = (jo3["values"].toArray())[0].toArray();

json_map["time"].toString().toStdString() << endl;

          cout << "resp : " << resp << endl;

        if(values[1].toInt() != NULL)
        {
            cout << values[1].toInt() << endl;
            mData = QString::number(values[1].toInt()).toLatin1();
            mAdresse = values[2].toString().toLatin1();
            mTime = resp.substr(98,19)/*.toDouble()).toLatin1()*/;

            cout << "time : " << mTime << endl;
            cout << "mData : " << QString(mData).toStdString() <<
endl;
            cout << "mAdresse : " << QString(mAdresse).toStdString()
<< endl;

            // mXbeeConnected();
            envoyerCommande();
        }
        else
        {
            cout << "pas de donnÃ©e" << endl;
        }
        //cout << "commande : " << QString(commande).toStdString()
<< endl;
}
```

Ce code va me permettre d'appeler la base de données de la GTC afin de recevoir toute les nouvelles informations (adresse, dim) où le ''flag'' est a 0 soit que la commande n'est pas encore effectuée. La trame est reçue sous la forme :

```
{"results":
      [{"statement_id":0,"series":
          [{"name":"dim","columns":
             ["time","value","add"],"values":
             [["1592849595854",50,"2428"]
             ]
          }]
      }]
 }
```



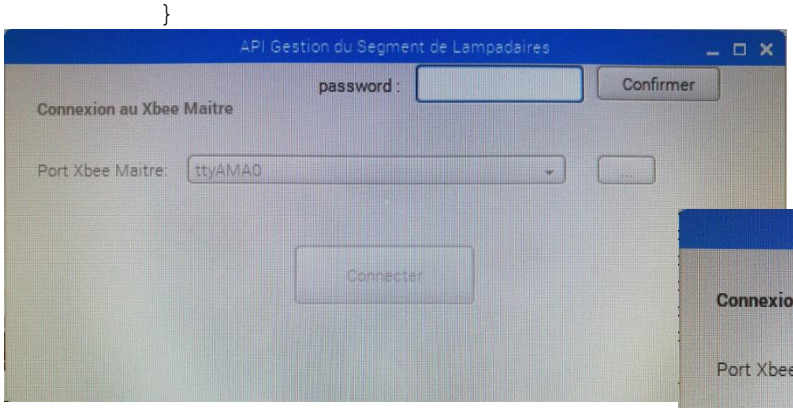
Afin de récupérer ces 3 valeurs contenues dans un format Json. J'avance de tableau en tableau jusqu'à arriver au niveau des valeurs voulues. Le TimeStamp sera alors récupéré en chaîne de caractère tout comme l'adresse mais le 'Dim' sera récupéré en 'int' ou simple valeur. Afin d'augmenter la sécurité de ce programme je me suis permis de rajouter un mot de passe ''local'' pour pourvoir activer la boucle qui permettra d'aller chercher les nouvelles commandes dans la base de données et de les envoyer.

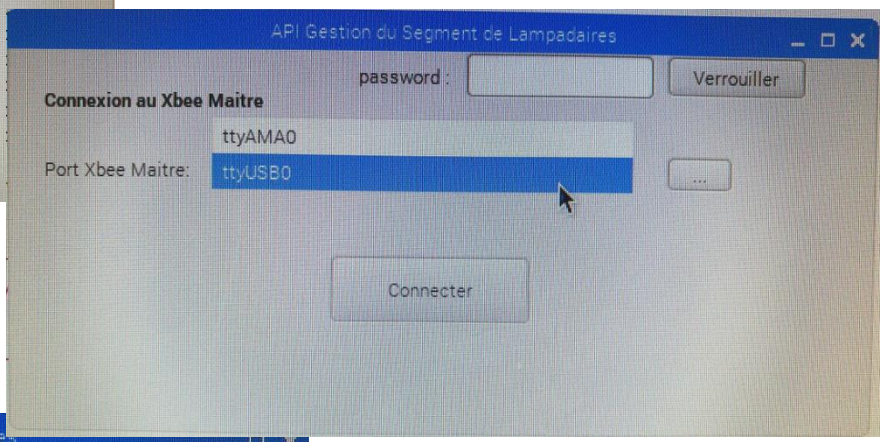
```
void MainWindow::on_pbMdp_clicked()
{
    if (ui->leMdp->text().toStdString() == mdp && ui->pbMdp-
>text().toStdString() == "Confirmer")
    {
        ui->gbConnexionXbee->setEnabled(true);
        ui->leMdp->setText("");
        ui->pbMdp->setText("Verrouiller");
    }
    else
    {
        ui->leMdp->setText("");
        ui->pbMdp->setText("Confirmer");
        ui->gbConnexionXbee->setEnabled(false);
    }
}
```

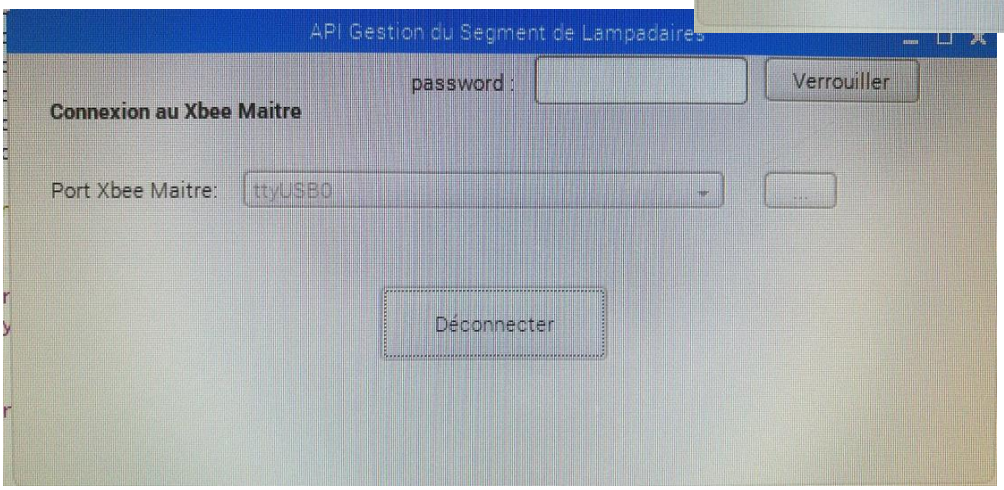
Enfin si le mot de passe est bon la partie connexion au module sera alors dégrisée et la configuration de celui-ci sera alors effectuée.

```
void MainWindow::on_pbConnecterXbee_clicked()
{
    if (ui->pbConnecterXbee->text() == "Connecter")
    {
        mXbee = new CSerial(this, "/dev/"+ui->cbPorts-
>currentText());
        connect(mXbee,
SIGNAL(sigErreur(QSerialPort::SerialPortError)), this,
SLOT(on_erreur(QSerialPort::SerialPortError)));
        connect(mXbee, SIGNAL(sigData(QByteArray)), this,
SLOT(on_recevoirDataDuPeriph(QByteArray)));

        mXbee->initialiser(QSerialPort::Baud9600,
                           QSerialPort::Data8,
                           QSerialPort::NoParity,
                           QSerialPort::OneStop,
                           QSerialPort::NoFlowControl);
        if(mXbee->ouvrirPort())
        {
            ui->pbConnecterXbee->setText("DÃ©connecter"); //dÃ¨s que
le port est ouvert, afficher le texte "DÃ©connecter"
            qDebug() << "Port ouvert";
            ui->cbPorts->setEnabled(false);
            ui->pbRaffraichirListePorts->setEnabled(false);
            timer->start();
        }
        else
```

```
        {
            qDebug() << "Le Port n'est pas ouvert !"; //si le port
n'est pas ouvert, affiche le message d'erreur
            ui->cbPorts->setEnabled(true);
            ui->pbRaffraichirListePorts->setEnabled(true);

        }
    }
    else
    {
        ui->pbConnecterXbee->setText("Connecter");
        ui->cbPorts->setEnabled(true);
        ui->pbRaffraichirListePorts->setEnabled(true);
        timer->stop();

        delete mXbee;
        QString messageFermer =
mDate.currentDate().toString("dd/MM/yyyy") + " " +
mNow.currentTime().toString("hh:mm:ss") + " : ";
        messageFermer += "Le port " + ui->cbPorts->currentText() + "
est fermÃ©";

    } // else
}
```







Mot de passe : Admin

```
std::string mdp = "admin";
```

Exécution du Programme

qt5ct: using qt5ct plugin

qt5ct: D-Bus global menu: no

CSerial::L'objet m_Sp du port "/dev/ttyUSB0" est créé

Serial Port : QSerialPort::SerialPortError(NoError)

Port ouvert

Indication de la bonne connection au module.

resp :
{"results":[{"statement_id":0,"series":[{"name":"dim","columns":["time","value","add"],"values":[[1559722394700960900,50,"5"]]}]}]}

Trame reçue provenant de la base de données de le gtc.

time : 1559722394700960900

mData : 50

mAdresse : 5

Checksum = K

Les données sont alors extraites de la trame et le calcul est alors effectué.

Data transmissent : "5;DIM;50;4b\n"

La trame vers le réseau xBee est alors formée avec les valeurs obtenues.

insert fait

query: DELETE FROM dim WHERE flag='0' AND add = '5' AND time=1559722394700960900

Les 2 commandes vers cette même database sont effectuées ainsi ce même ordre ne sera pas refait car le flag sera passé à 1.

pas de données

resp : {"results":[{"statement_id":0}]}

La boucle étant fini le programme recommence alors à chercher un nouvel ordre à effectuer.

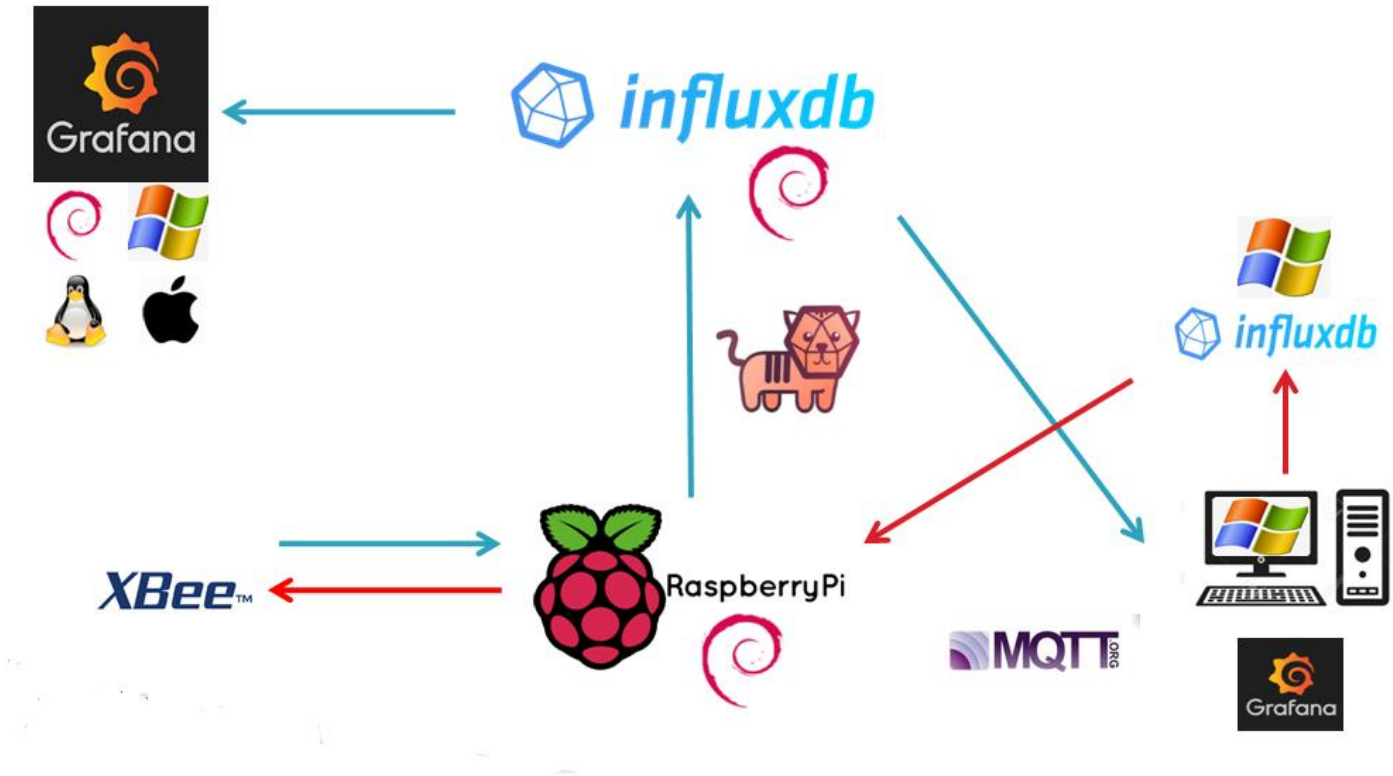
```
> select * from dim
name: dim
time                  add flag value
----                  --- ----- -----
1559722394700960900 5   0     50
> select * from dim
name: dim
time                  add flag value
----                  --- ----- -----
1559722461723298800 5   1     50
>
```

-> nouvelle commande = 0
-> commande effectuée = 1

Dans cette exemple nous pouvons voir que les trames a envoyer seront alors la 3 et la 4

	Dimmer	addr	actual
1 ←	0	1	1
2 ←	54	2	1
3 ←	62	6	0
4 ←	36	7	0

Finalité du Projet



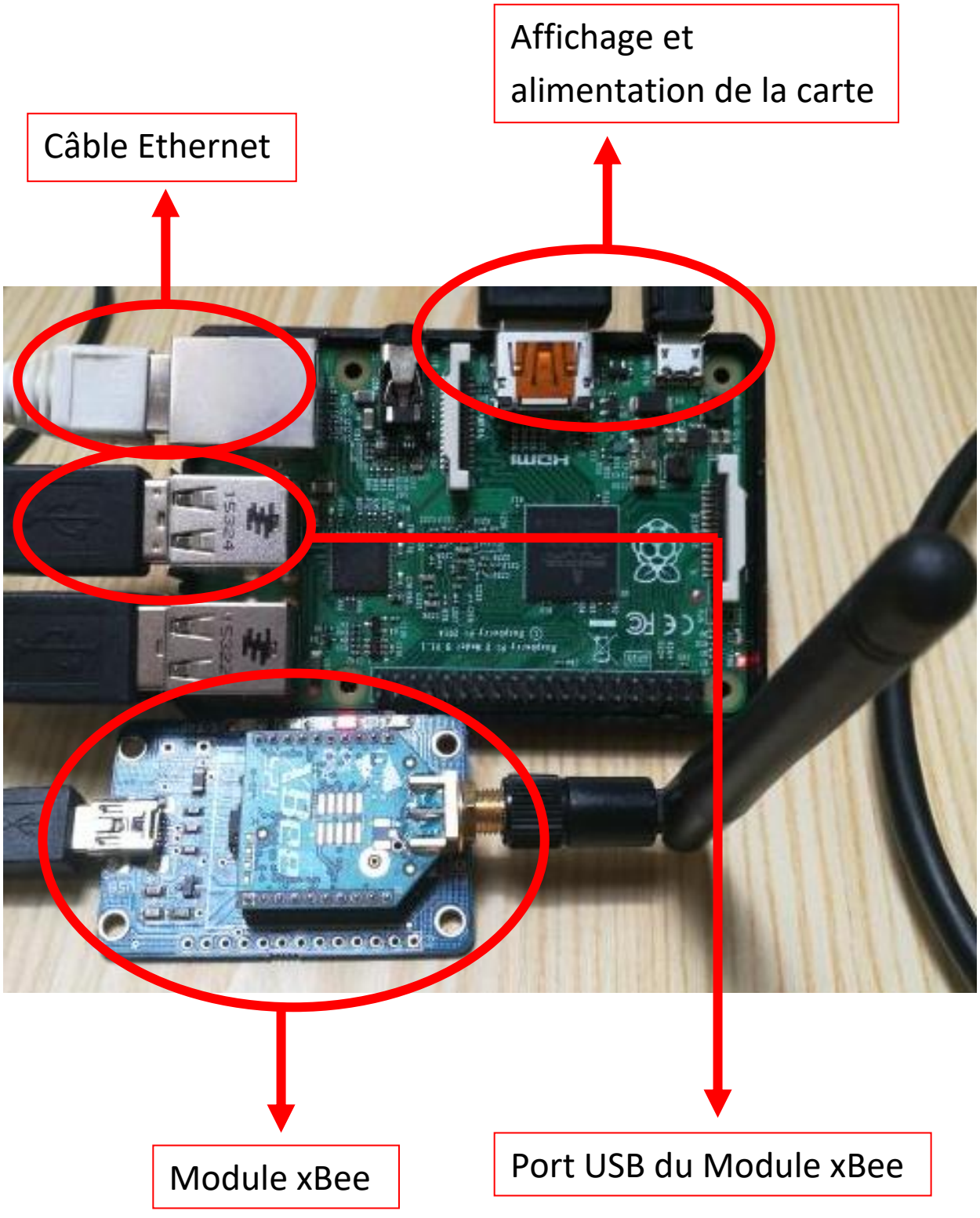
Fleche de Retour (conso…)

Fleche d'envoie de Trame (Ordre…)

Les trames de retour étant en cours de finition.

La partie d'envoie des trames est totalement fonctionnel**.**

Carte Raspberry



Conclusion

Ce projet m'a appris beaucoup en c++ ainsi qu'en base de données. La communication entre tous les éléments a développé chez moi un grand intérêt ce qui m'a poussé à faire des recherches sur d'autres solutions possible, notamment sur le système de 'flag', sur lequel je me suis beaucoup penché (autre solution trouvée : konqueror). Mais ce projet a aussi développé chez moi un esprit d'équipe poussé surtout avec l'élève d'IR1 (Thomas Gonzalez). Pour la suite je tiens à passer mon application en 'service' afin de réduire au maximum le travaille des techniciens.

Projet API-GTC

Revue 3 - Partie Individuelle

Thomas Gonzalez IR1

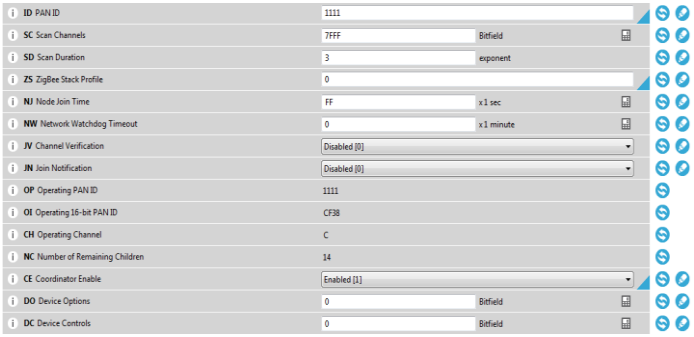
Gestion Technique Centralisée d'un Eclairage Public avec communication XBee

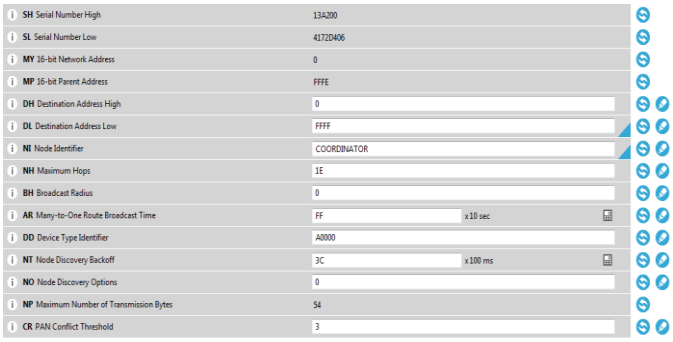
CONFIGURATION DES MODULES SUR XCTU

Sur le logiciel XCTU de Digi, afin de tester la communication entre deux modules XBee, j'ai configuré un routeur et un configurateur.

ID PAN ID	1111		
SC Scan Channels	7FFF	Bitfield	
SD Scan Duration	3	exponent	
ZS ZigBee Stack Profile	0		
NJ Node Join Time	FF	x1 sec	
NW Network Watchdog Timeout	0	x1 minute	
JV Channel Verification	Disabled [0]		
JN Join Notification	Disabled [0]		
OP Operating PAN ID	1111		
OI Operating 16-bit PAN ID	CF38		
CH Operating Channel	C		
NC Number of Remaining Children	14		
CE Coordinator Enable	Disabled [0]		
DO Device Options	0	Bitfield	
DC Device Controls	0	Bitfield	
SH Serial Number High	13A200		
SL Serial Number Low	4172D40C		
MY 16-bit Network Address	1677		
MP 16-bit Parent Address	FFFE		
DH Destination Address High	0		
DL Destination Address Low	0		
NI Node Identifier	ROUTER		
NH Maximum Hops	1E		
BH Broadcast Radius	0		
AR Many-to-One Route Broadcast Time	FF	x10 sec	
DD Device Type Identifier	A0000		
NT Node Discovery Backoff	3C	x100 ms	
NO Node Discovery Options	0		
NP Maximum Number of Transmission Bytes	54		
CR PAN Conflict Threshold	3		

Les paramètres à configurer pour configurer un module en routeur sont l'**ID** (1111 ici), le **JV** (0), **DL** et **DH** (tout les deux à 0) et le **CE** (0).





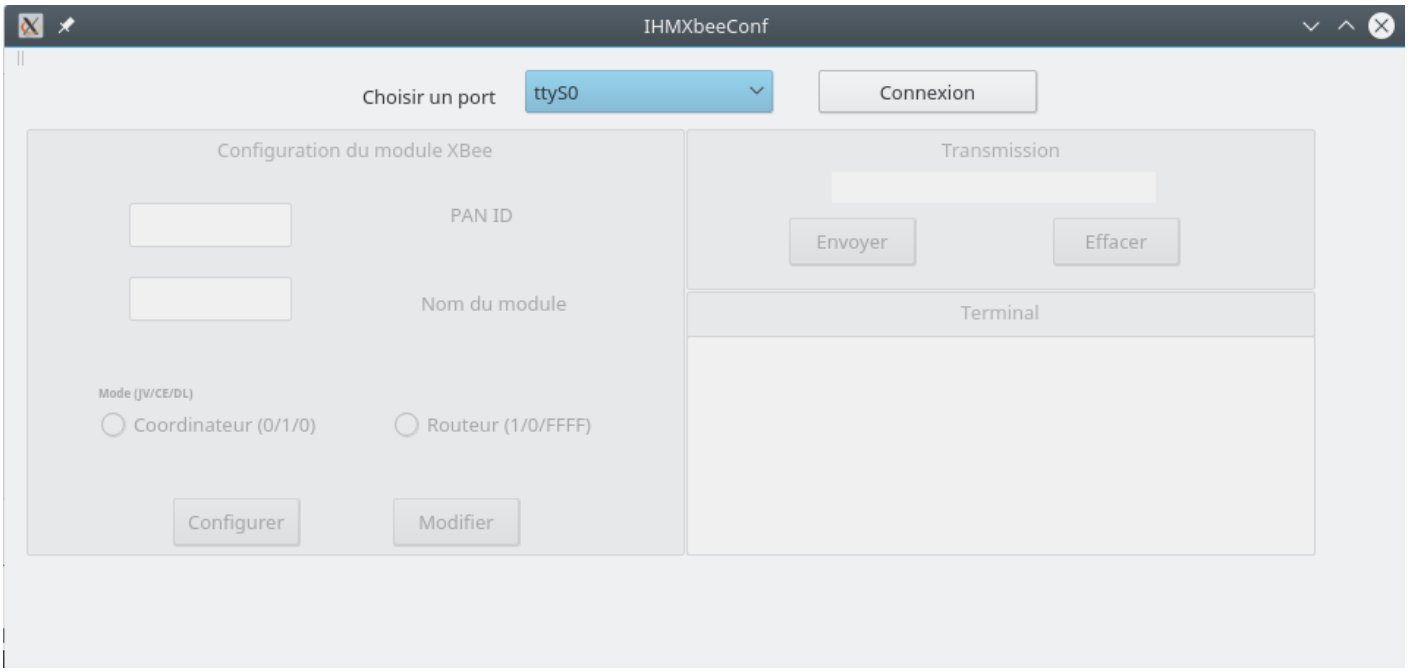
Les paramètres à configurer pour configurer un module en configurateur sont l'**ID** (1111 ici), le **JV** (0), **DL** et **DH** (FFFF pour le **DL**) et le **CE** (1).

PREMIERE VERSION DU LOGICIEL DE CONFIGURATION DES MODULES XBee.

1. Description de l'IHM

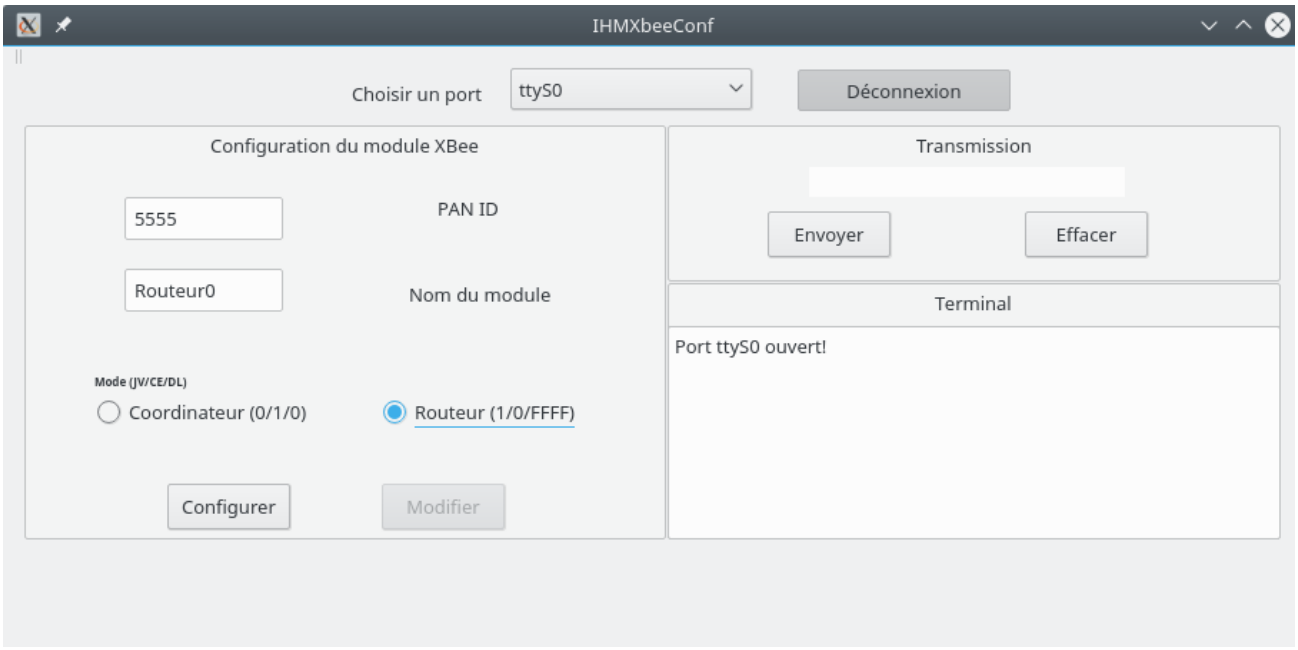
A l'instar du logiciel XCTU, l'objectif était de créer un logiciel permettant de configurer les modules XBee prenant en compte les paramètres suivants : le **PAN ID** (qui permet d'identifier le module), le **JV** (ou Channel Vérification qui permet de vérifier le canal, si il est définit à 1 un routeur vérifie que le coordinateur est sur son canal de fonctionnement lors de la connexion d'un cycle d'alimentation, si il est à 0 le routeur continue de fonctionner sur son canal actuel même si aucun coordinateur n'est détecté lors de la connexion ou de la déconnexion du cycle), **DH** (partie haute de l'adresse) et **DL** (adresse de destination : broadcast), enfin le **CE** (qui permet de définir si le module est un coordinateur ou non).

L'application doit être permettre la connexion d'un module XBee, sa configuration (avec les paramètres cités précédemment) ainsi que l'envoie de message à un autre module afin de tester la connectivité.



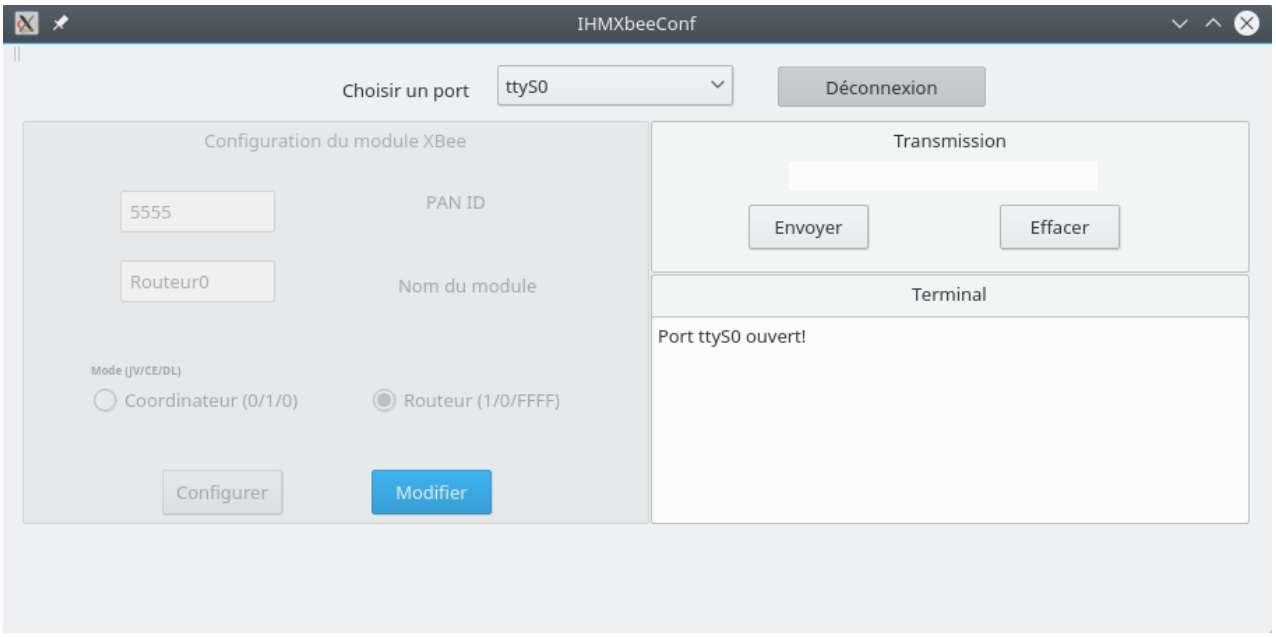
Capture d'écran n°1

Comme sur la capture d'écran n°1, la partie connexion se situe en haut de l'application et propose de sélectionner un port avec l'objet *QComboBox* ainsi que la connexion du module. A gauche, la configuration du module avec le PAN ID, le nom du module ainsi qu'un *RadioButton* permettant de définir si le module est un routeur ou un coordinateur. A droite, se trouve la partie transmission qui permet l'envoie d'un message quelconque à un autre module avec deux *QPushButton* et plus bas un terminal affichant les messages d'actions et d'erreurs.



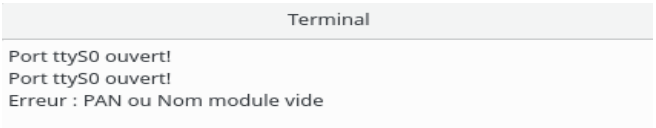
Capture d'écran n°2

Lorsque le module XBee est connecté, le texte du *QPushButton* passe « Connexion » à « Déconnexion », les *QGroupBox* de la configuration, de la transmission et du terminal sont activés (alors que sur la capture d'écran n°1 ils sont désactivés/grisés) et un message s'affiche dans le terminal.



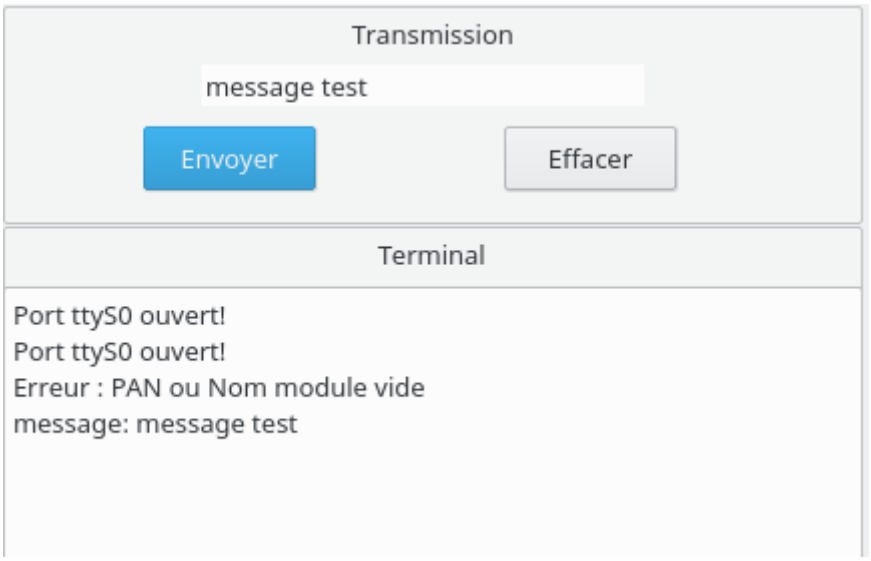
Capture d'écran n°3

Sur la capture d'écran n°3, dans le cas où le champ « PAN ID » et « Nom du module » et le mode (coordinateur ou routeur) sont remplis, le *QGroupBox* se désactive et le bouton « Modifier » initialement désactivé, s'active.



Capture d'écran n°4

Dans le cas contraire, comme sur la capture d'écran n°4, un message d'erreur s'affiche dans le terminal signalant qu'un des *QTextEdit* n'est pas rempli et le *QGroupBox* n'est pas désactivé.



Capture d'écran n°5

Sur la capture d'écran ci-dessus, le message entré dans la partie « Transmission » s'affiche dans le debugger de la partie « Terminal ».

2. Description du code

a. Classe IHMXbeeConf

Le code suivant est celui de l'envoie du message. On définit une variable *msg* que l'on assigne au *QTextEdit* « writeMsg ». Enfin, dans le debugger **Qt** et le terminal de l'application on affiche le message dans le format **« message : [texte] »**

```
void IHMXbeeConf::on_pbEnvoyer_clicked()
{
    QString msg;
    msg = ui->writeMsg->text();
    ui->txtDebug->append("message: " + msg);
    qDebug()<<"message: "<< msg;
}
```

```
void IHMXbeeConf::on_pbConnexion_clicked()
{
    if(ui->pbConnexion->text()=="C&onnexion") // si le texte du bouton est "Connexion"
    {
        QString nomPort = ui->cbPort->currentText();
        port = new Cserial(this, nomPort);
        port->initialiser(QSerialPort::Baud9600, QSerialPort::Data8,
                          QSerialPort::NoParity, QSerialPort::OneStop,
                          QSerialPort::NoFlowControl);
        connect(port, SIGNAL(sigOk()), this, SLOT(slotStatusOK())); // connexion du signal au slot
        connect(port, SIGNAL(sigConfXbee(QByteArray)), this, SLOT(slotConfXbee(QByteArray)));
```

Le code ci-dessus est celui de l'appui sur le bouton connexion. Dans la première boucle **if**, si le texte du *QPushButton* est « Connexion » alors on définit une variable *nomPort* qui instancie la classe Cserial. Ensuite, la variable *initialiser,* définie dans la classe Cserial.h, initialise les fonctions permettant d'accéder aux ports séries comme la vitesse de transmission (9600 baud), le bit de parité, le bit de stop, le contrôle de flux etc… Enfin la fonction connect() permet la connexion du signal sigOK au slot slotStatusOK et du signal sigConfXbee au slot slotConfXbee.

```
if(port->ouvrirPort())
{
    ui->pbConnexion->setText("&Déconnexion");
    ui->txtDebug->append("Port " + nomPort + " ouvert!");
    qDebug()<<"Port"<<nomPort<<" ouvert !"; //affichage du nom du port ouvert dans le debugger
    ui->grpConfig->setEnabled(true);
    ui->grpDebug->setEnabled(true);
    ui->grpTrans->setEnabled(true);
    ui->pbConfigurer->setEnabled(true);
}
else
{
qDebug() << "Echec de l'ouverture du port " << nomPort << " !"; //si le port n'est pas ouvert, une erreur est affichée
ui->grpConfig->setEnabled(false);
ui->grpDebug->setEnabled(false);
ui->grpTrans->setEnabled(false);
ui->pbConfigurer->setEnabled(false);
```

Si le port est ouvert (méthode *ouvrirPort()*) alors le texte du bouton passe « Connexion » à « Déconnexion » et l'ensemble des *QGroupBox* sont activés. Le port ouvert est affiché sur le terminal et le debugger de **QtCreator**. Sinon, les *QGroupBox* restent désactivés et un texte d'erreur s'affiche dans le debugger.

```
    else
    {
        ui->grpConfig->setEnabled(false);
        ui->grpDebug->setEnabled(false);
        ui->grpTrans->setEnabled(false);
        ui->pbConfigurer->setEnabled(false);
        disconnect(port, SIGNAL(sigOk()), this, SLOT(slotStatusOK()));
        disconnect(port, SIGNAL(sigConfXbee(QByteArray)), this, SLOT(slotConfXbee(QByteArray)));
        delete port;
        qDebug() << "Déconnexion effectuée";
        ui->pbConnexion->setText("C&onnexion");
    }
}
```

Si le texte n'est pas « Connexion » (en l'occurrence s'il est « Déconnexion »), alors les signaux sont des déconnectés avec la méthode *disconnect*. Le message d'alerte avertissant de la déconnexion est affiché et le texte du bouton connexion repasse à « Connexion ».

```
int Cserial::ecrire(const char *trame)
{
    return m_serial->write(trame);
}

QString Cserial::getAction() const
{
    return m_action;
}

void Cserial::setAction(const QString &action)
{
    m_action = action;
}
```

Dans la partie ci-contre, trois méthodes *ecrire*, *getAction* et *setAction* sont définies. La première permet l'écriture de la trame.

```
void IHMXbeeConf::slotStatusOK()
{
qDebug() << "Status : OK - Action : " << port->getAction(); //affiche dans le debugger l'action effectuée
ui->txtDebug->append("Status : OK - Action : " + port->getAction());
if(port->getAction()=="reset") // si l'action est un reset
{
    port->ecrire("ATRE\r");
    qDebug() << "commande : ATRE\r";
    port->setAction("save");
}
```

Le code ci-dessus est celui du slot slotstatusOK :

```
if(port->getAction()=="writeId") //si l'action est l'obtention du pan id
{
    const char *commande = ("ATID"+ui->lePAN->text()+"\r").toStdString().c_str();

    qDebug() << "commande : " << commande; //affiche le pan id
    port->ecrire(commande);
    port->setAction("writeCe");
}
else
```

```
if(port->getAction()=="writeCe")
{
    QString Ce ="0";
    if(ui->rbCoordinateur->isChecked()) Ce ="1"; // si le radio button rbCoordinateur est sélectionné, définir le Ce à
    const char *commande = ("ATJN"+Ce+"\r").toStdString().c_str(); // toStdString convertie un objet QString en std::st
    qDebug() << "commande : " << commande; //affiche le paramètre Ce (définit à 1 pour le rôle du
    port->ecrire(commande);
    port->setAction("writeJv");
}
else
```

Un paramètre « Ce » est définie à 0, si le *QRadioButton* du coordinateur est sélectionné (**isChecked**) alors le « Ce » passe à 1. Une variable *commande* est créée et renvoie le paramètre « Ce » convertie de Qstring à l'objet std::string. La commande s'affiche dans le debugger.

```
if(port->getAction()=="writeJv")
{
    QString Jv ="0";
    if(ui->rbRouteur->isChecked()) Jv ="1"; //si le radio button rbRouteur est sélectionné, définir le J
    const char *commande = ("ATJV"+Jv+"\r").toStdString().c_str();
    qDebug() << "commande : " << commande;
    port->ecrire(commande);
    port->setAction("writeDh");
}
else
```

De la même façon que le paramètre « Ce », un paramètre « Jv » est définie à 0, si le *QRadioButton* du routeur est sélectionné alors le

paramètre « Jv » passe à 1. La suite est le même fonctionnement que pour le paramètre « Ce ».

```
if(port->getAction()=="writeDh") //adresse de destination maximale
{
     const char *commande = "ATDH0\r";
     qDebug() << "commande : " << commande;
     port->ecrire(commande);
     port->setAction("writeDl");
}
else
if(port->getAction()=="writeDl") //adresse de destination minimale
{
    QString Dl ="0";
    if(ui->rbCoordinateur->isChecked()) Dl ="FFFF"; //si le radio buton rbCoordinateur est sélectionné, définit l à FFFF
    const char *commande = ("ATDL"+Dl+"\r").toStdString().c_str();
    qDebug() << "commande : " << commande;
    port->ecrire(commande);
    port->setAction("writeNi");
}
    else
    if(port->getAction()=="writeNi") // écrire le nom du module
    {
        const char *commande = ("ATNI"+ui->leNom->text()+"\r").toStdString().c_str();
        qDebug() << "commande : " << commande;
        port->ecrire(commande);
        port->setAction("save"); //sauvegarde la commande entrée
    }
    else
```

Le code ci-dessus est celui des autres paramètres (Dh, Dl et le nom du module). Le fonctionnement est la même chose que pour les précédents.

Ci-contre, la sauvegarde est réalisée avec l'écriture de « ATWR\r » sur la trame et son affichage sur le debugger. Si l'action effectuée est la lecture de l'ID, sur le debugger sera affiché « commande : ATID\r ». Enfin si l'action est la sortie du mode (exit) alors le message affiché dans le debugger sera « sortie du Mode conf : OK ».

```
    if(port->getAction()=="save")
    {
        port->ecrire("ATWR\r");
        qDebug() << "commande : ATWR\r";
        port->setAction("readId");
    }
    else
        if(port->getAction()=="readId")
        {
            port->ecrire("ATID\r");
            qDebug() << "commande : ATID\r"
        }

if(port->getAction()=="exit")
{
    qDebug() << "Sortie du Mode conf : OK";
}

}
```

```
void IHMXbeeConf::slotConfXbee(QByteArray conf)
{
    if(port->getAction()=="readId")
    {
        qDebug() << "ID : " << QString(conf) ;
        ui->lePAN->setText(QString(conf));
        //
        // Attention : utilisation de JN a la place de CE si Xbee S2
        //
        qDebug() << "commande : ATJN\r";
        port->ecrire("ATJN\r");
        port->setAction("readCe");
    }
    else
```

A gauche, le code de slotConfXbee. L'ID est celui rentré dans le *QTextEdit* par l'utilisateur. La valeur est « enregistrée » dans la variable *conf* et est affichée dans le debugger.

A droite, les paramètres « Ce » et « Jv », vus précédemment pour slotStatusOK. La fonction *at()* permet de récupérer un caractère d'une chaîne donnée. Ici, il récupère le caractère de conf. Si le *QRadioButton* du coordinateur est coché alors sur le debugger est affiché « commande : ATJV\r ».

```
if(port->getAction()=="readCe")
{
    qDebug() << "CE : " << QString(conf) ;
    if(conf.at(0)=='1') ui->rbCoordinateur->setChecked(true);
    qDebug() << "commande : ATJV\r";
    port->ecrire("ATJV\r");
    port->setAction("readJv");
}
else
if(port->getAction()=="readJv")
{
    qDebug() << "JV : " << QString(conf) ;
    if(conf.at(0)=='1') ui->rbRouteur->setChecked(true);
    qDebug() << "commande : ATDH\r";
    port->ecrire("ATDH\r");
    port->setAction("readDh");
}
else
```

Le fonctionnement est le même pour les autres paramètres (Jv, Dl, Dh et Ni, le nom du module).

```
void IHMXbeeConf::on_pbConfigurer_clicked()
{
    if(ui->leNom->text()=="" || ui->lePAN->text()=="")
    {
        qDebug() << "Erreur : PAN ou Nom module vide";
        ui->txtDebug->append("Erreur : PAN ou Nom module vide");

    }
    else
    {
        port->ecrire("+++");
        port->setAction("writeId");
        ui->grpConfig->setEnabled(false);
        ui->pbModifier->setEnabled(true);
        ui->pbConfigurer->setEnabled(false);

    }
}
```

Ci-contre, le code du bouton Configurer. Si le nom du module et/ou le PAN ID ne sont pas renseignés, alors un message d'erreur s'affiche à la fois sur le debugger **Qt** et sur le terminal de l'application. Sinon, démarrage de la connexion avec le module (+++) et activation du bouton Modifier. Le code du bouton Modifier est le même que le bouton Configurer.

b. Classe Cserial

```
int Cserial::ouvrirPort()
{
    return m_serial->open(QIODevice::ReadWrite);
}
```

La méthode *ouvrirPort()* ouvre le port pour la lecture et l'écriture (avec QIODevice ::ReadWrite).

m_serial est un pointeur de **QSerialPort.**

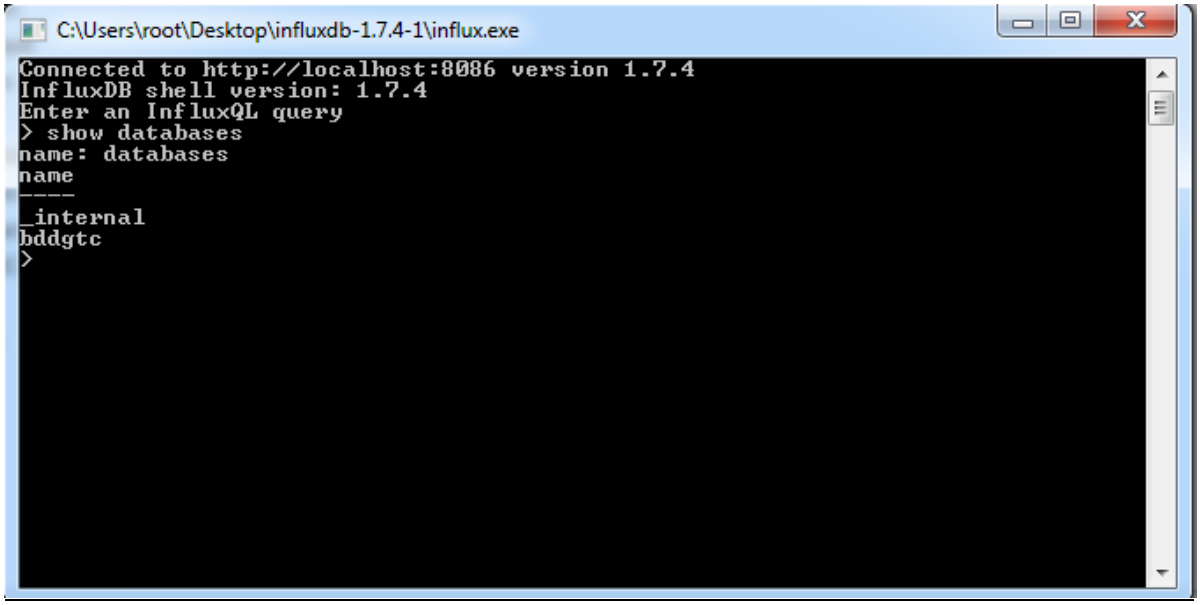
```
Cserial::Cserial(QObject *parent, const QString &nomPort)
{
    m_serial = new QSerialPort(parent);
    m_serial->setPortName(nomPort);
    m_recu = 0;
    connect(m_serial, &QSerialPort::readyRead, this, &Cserial::slotLireDonnees);
    qDebug() << "L'objet CSerial m_serial est créé ";
}
```

Le code ci-dessus est celui du constructeur de la classe CSerial. Le nom du port série est définit sur « nomPort ». Le signal m_serial est connecté au slot slotLireDonnees. L'application n'envoie pas encore la configuration et la communication entre deux modules (comme avec XCTU), la classe Cserial n'est pas terminée. Cependant les actions au niveau « physique » de l'IHM sont fonctionnelles et l'application ressemble, dans les grandes lignes, à ce qu'elle devra être dans sa version finale.

CREATION DE LA BASE DE DONNEE

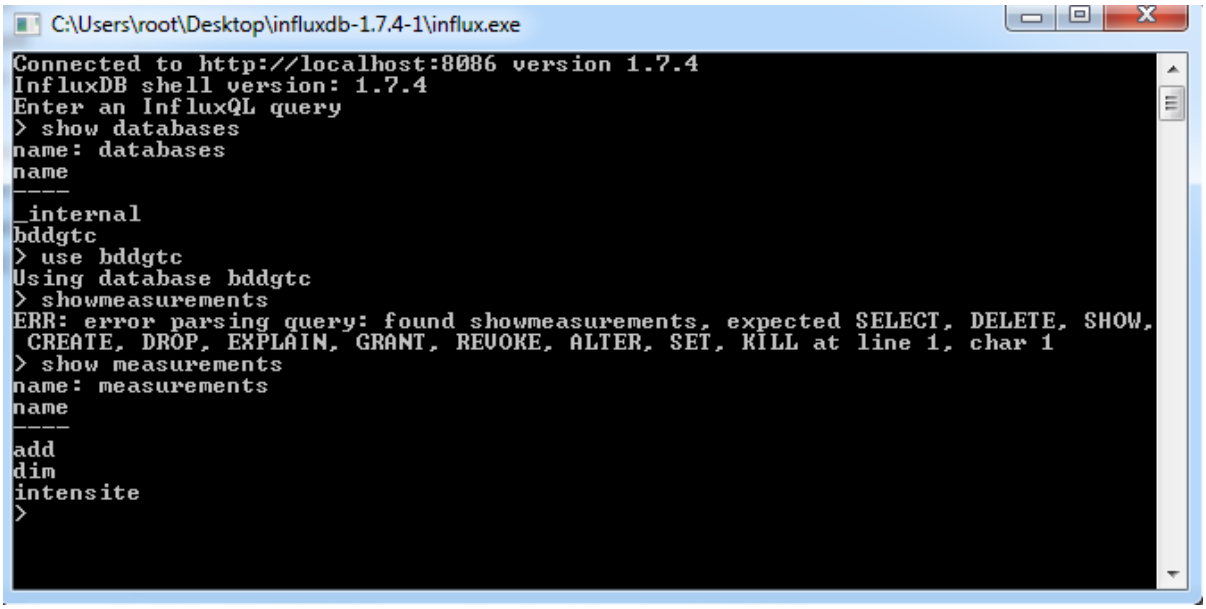
1. InfluxDB

L'installation de la base de données s'est fait avec **InfluxDB**. Ce dernier est un système de gestion de base de données orientées séries chronologiques c'est-à-dire que les données sont récupérées chronologiquement. Le logiciel se présente comme ceci :



Capture d'écran n°6

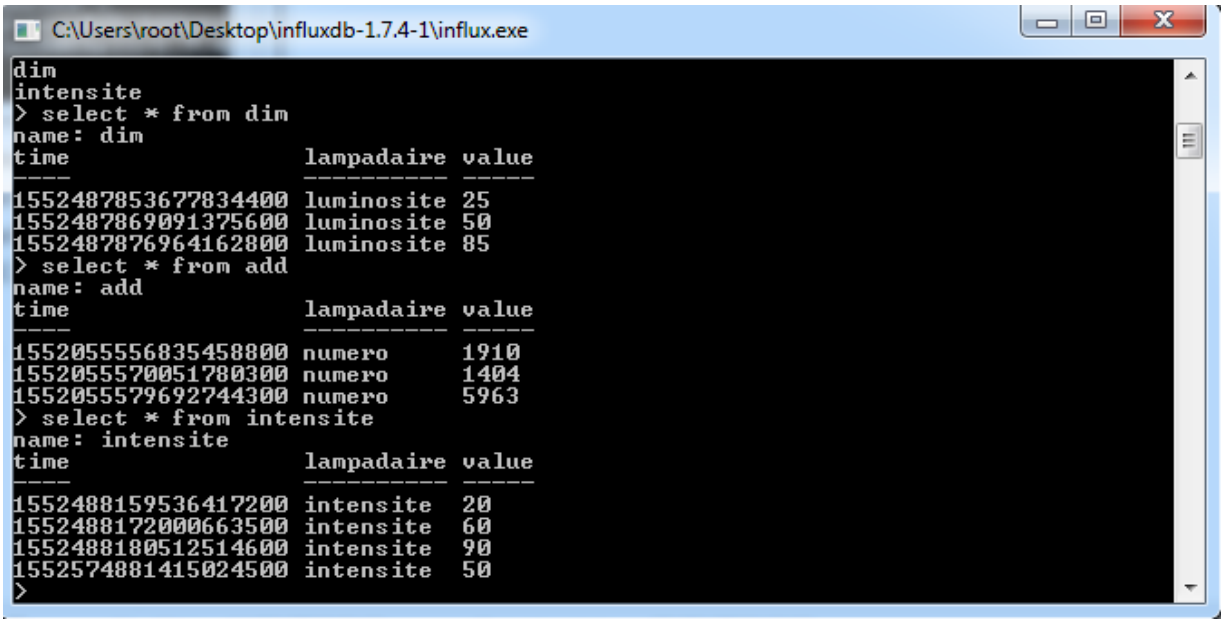
La commande entrée *showdatabases* permet de montrer les différentes bases de données créées. Ici, celle qui nous intéresse est celle que j'ai créée manuellement (avec la commande *create database*) nommée **bddgtc**



Capture d'écran n°7

Cette capture d'écran, datant de quelques semaines, me permet de montrer un changement dans la perception du projet. Désormais, *intensite* n'existe plus car, après réflexion, il n'y a pas vraiment de différence entre l'intensité et la luminosité d'un lampadaire d'un point de vue physique. Enfin, *add* sert à stocker les adresses des différents modules XBee.

La commande *showmeasurements* permet de montrer les différentes tables de la base de données (en effet les tables sont appelées « measurements » sur InfluxDB). Ici, les « tables » créées sont *add* (adresses des lampadaires), *dim* (valeur du dimmer, de la luminosité des lampadaires) et *intensite* (intensité des lampadaires).

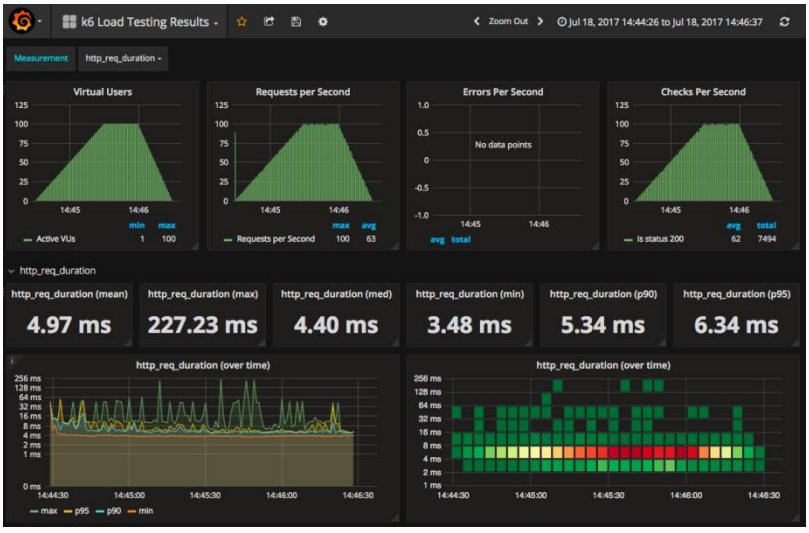


Capture d'écran n°8

La commande *select * from [measurement]* permet d'afficher les valeurs des mesures. Bien évidemment, les valeurs sont factices car aucune liaison n'est faite avec le **MQTT** permettant la récupération des données en temps réel des modules XBee. Pour insérer les valurs des « tables », il a fallu utiliser la commande suivante : *insert [nomTable], [tag set]=XXX value=XXX*.

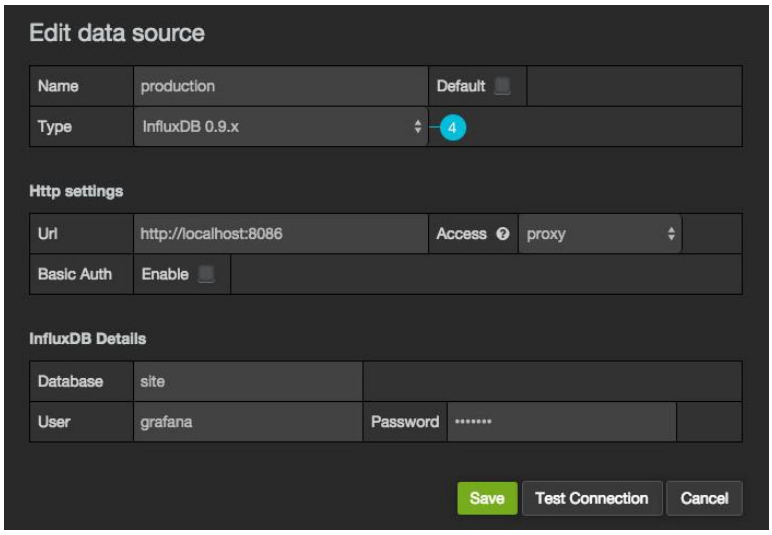
2. Grafana

Grafana est un logiciel de visualisation et mise en forme de données métriques sous forme de graphiques et tableaux de bords entre autres.



Exemple d'utilisation de Grafana

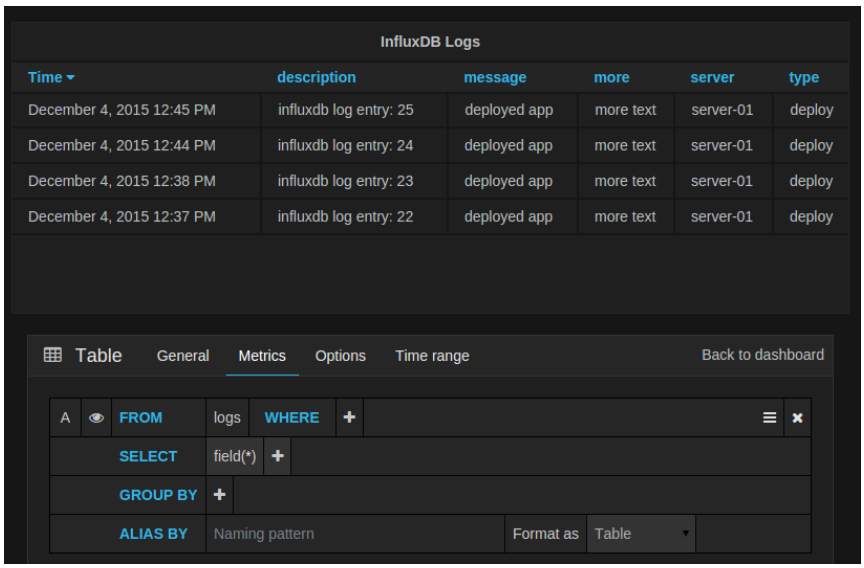
Après l'installation du logiciel, il a fallu, dans le dossier *conf* sur notepad++, changer la valeur de *http_port* par celle du localhost. Initialement définie à 8080, elle est passée à 8086. Il a donc suffit de rentrer l'adresse du localhost dans un navigateur pour arriver sur la page d'accueil de Grafana.



Pour ajouter une nouvelle base de données, il a seulement fallut renseigner le nom de l'utilisateur et le mot de passe définis sur InfluxDB.

Sur l'exemple ci-dessous, nous pouvons voir la gestion des métriques pour la visualisation d'une table de la BDD. N'ayant pas pris de capture d'écran de ma simulation pour illustrer mon propos, il a tout simplement fallu choisir la table, les valeurs à afficher et régler l'affiche des données sur le graphique (modifier le temps de rafraîchissement des données par exemple).

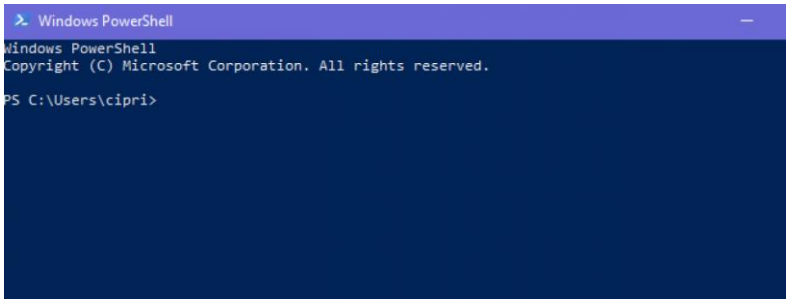
Capture d'écran n°9



Exemple d'édition de BDD sur Grafana

3. Telegraf

Telegraf est un outil permettant la collecte de données et mesures. C'est lui qui « relie » InfluxDB à Grafana. Pour le faire fonctionner, c'est un peu plus compliqué puisqu'il faut éditer (avec notepad ++ par exemple) le fichier de configuration (conf).



Windows PowerShell

Ceci étant fait, il a fallu configurer Telegraf comme un service Windows (à l'aide de Windows PowerShell).

Les commandes à entrer, après avoir insérer le fichier Telegraf dans le dossier Program Files(x86), étaient les suivantes :

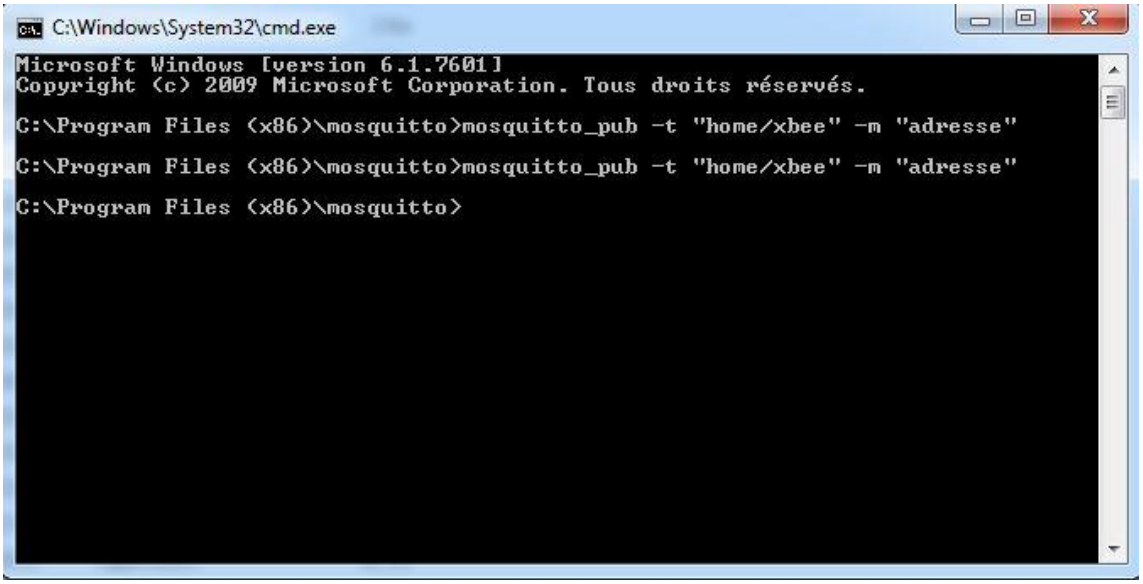
- *C:\"Program Files"\Telegraf\telegraf.exe --service install* : Installation du service dans le gestionnaire de service Windows.
- *C:\"Program Files(x86)"\Telegraf\telegraf.exe --config C:\"Program Files(x86)"\Telegraf\telegraf.conf –test*
- *net start telegraf* : Commencer à collecter les données
- *telegraf.exe --service start* : démarrage du service

4. Mosquitto

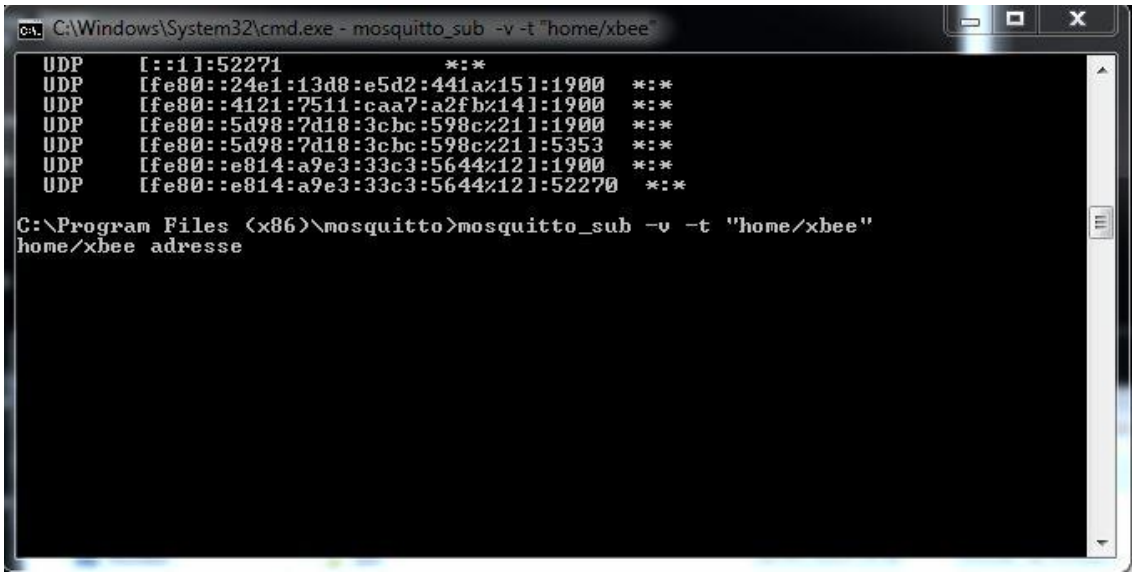
Mosquitto est un broker MQTT (protocole qui envoie et reçois des données). Le broker propose un système de topics sur lesquels les gens peuvent écrire et recevoir des données une fois qu'ils y sont abonnés. Après l'installation du broker et de ses différents services, on ouvre deux terminaux en parallèle.

Premier terminal

Dans le premier on regarde l'adresse du protocole installé (1883) et on s'abonne au topic désiré (avec la commande *mosquitto_sub*), dans le deuxième on publie des données au format JSON (*mosquitto_pub*), et on peut en recevoir dans le cas où un autre utilisateur est abonné à notre topic.



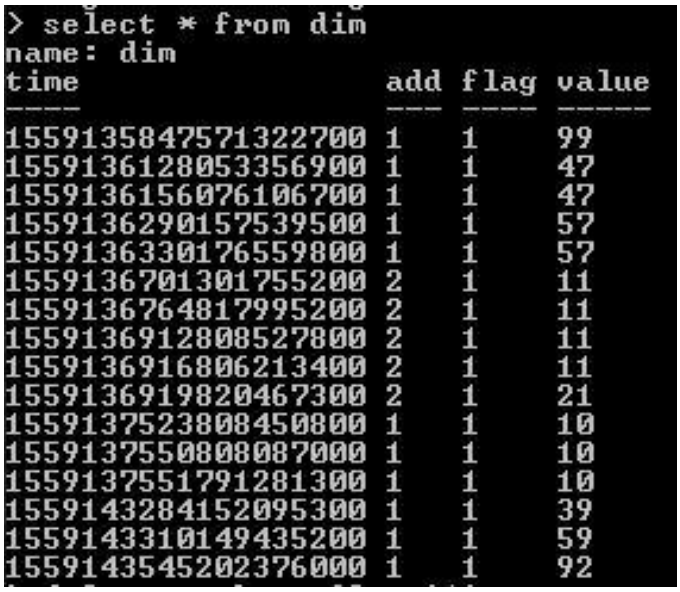
Terminal servant à la publication de message



Terminal recevant le message suite à l'abonnement au topic

5. Les bases de données

Les étapes suivantes ont servies à la création d'une base de données InfluxDB sur Windows et une autre disponible sur Raspberry Pi. L'application GTC se sert de ces deux bases de données afin de récupérer les informations des mesures (BDD Raspberry Pi) et apporter les modifications nécessaires sur les lampadaires comme la luminosité (BDD Windows).



```
> select * from dim
name: dim
time                add flag value
----                --- ---- -----
1559135847571322700 1   1    99
1559136128053356900 1   1    47
1559136156076106700 1   1    47
1559136290157539500 1   1    57
1559136330176559800 1   1    57
1559136701301755200 2   1    11
1559136764817995200 2   1    11
1559136912808527800 2   1    11
1559136916806213400 2   1    11
1559136919820467300 2   1    21
1559137523808450800 1   1    10
1559137550808087000 1   1    10
1559137551791281300 1   1    10
1559143284152095300 1   1    39
1559143310149435200 1   1    59
1559143545202376000 1   1    92
```

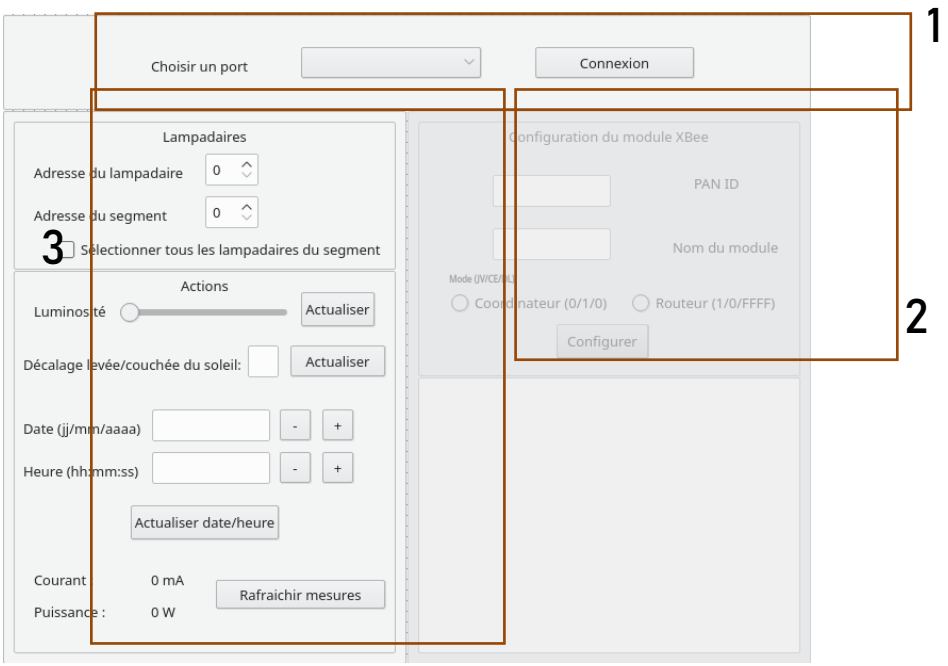
Capture d'écran n°10

La capture d'écran n°10 montre l'ensemble des valeurs effectuées à partir de l'application GTC. Contrairement à la capture d'écran n°8, les valeurs ne sont donc pas factices ni entrées manuellement. Lorsque la requête est envoyée au lampadaire, elle est « vérifiée » instantanément par l'application de l'étudiant IR2. Cette même application passe le flag à 1 et récupère la valeur (*value*) et l'adresse (*add*) pour l'envoyer au lampadaire en formant la trame ci-dessous :

ADD ; DIM ; <valeur entre 0 et 100> ; CHECKSUM

ADD correspond à l'adresse du lampadaire, **DIM** à l'ordre voulu (ici le dimmer donc la luminosité), la valeur entre 0 et 100 et enfin le **checksum** calculé par l'application de l'**IR2**.

CONCEPTION DE L'APPLICATION DE LA GESTION TECHNIQUE CENTRALISEE (GTC)



La partie encadrée en haut (1) sert à la connexion du module XBee. Cette partie a le même fonctionnement que sur l'application de configuration du module. D'ailleurs la partie encadrée 2, est une réduction de cette application. En effet, lors du développement du projet, pour des raisons d'ergonomie, je me suis rendu compte qu'il serait plus pratique de rassembler la configuration des modules avec le réglage des lampadaires. On retrouve donc la sélection des modes (coordinateur ou routeur), l'ID et le nom du module à configurer. Il sera donc possible à l'utilisateur de configurer un module, s'il en éprouve le besoin, mais aussi d'accéder à plusieurs modifications sur le(s) lampadaire(s).

C'est l'utilité de la partie 3, de haut en bas, la sélection des adresses du lampadaire, du segment ou bien encore la possibilité de sélectionner tous les lampadaires d'un segment. Plus bas, la luminosité et l'intensité sont réglables. Encore plus bas, la date et l'heure locale peut être modifiée (apparition d'un calendrier envisagée). Enfin, les données du courant et de puissance d'un lampadaire sont affichées en bas. Il est à noter qu'à l'heure où j'écris ces lignes, la partie configuration n'est pas encore au point ainsi que la réception des mesures (courant et puissance) et la mise à jour de la date et de l'heure.

Pour conclure, avant l'explication du code, la réception de la luminosité par le lampadaire est optimale. Les seuls points à régler désormais sont la réception des mesures de consommation, leur affichage sur Grafana et la configuration des modules.

Description (de quelques parties majeures) du code :

```
void app_gtc::on_pb_actuDim_clicked()
{
    mOrdre = "DIM";
    mData.setNum(ui->sl_dim->value());

    const char* adresse = mAdresse.constData();
    float donnees = QString(mData).toFloat();   //convertion en flottant
    influxdb_cpp::server_info si("192.168.4.34", 8086, "bddgtc", "", "");
    influxdb_cpp::builder()
            .meas("dim")
            .tag("add", adresse)
            .tag("flag","0")
            .field("value", donnees)
            .post_http(si, &resp);
}
```

La partie ci-dessus permet de définir l'action suivant l'appuie sur le bouton *Actualiser* du dimmer. Après avoir converti la valeur *mAdresse* du module en char, je crée une variable *donnees* qui configure la valeur *mData* (valeur conrrespondant à la valeur du slidder du dimmer) en float. Ensuite, je me connecte à la base de données en renseignant l'adresse, le port et le nom de la BDD. Puis je renseigne dans *builder()*, le measurement (ici dim), les tags (add qui contiendra l'adresse du module et le flag qui sera définit à 0), le field qui affichera dans *value* la valeur de la luminosité.

```
CSerial::CSerial(QObject *parent, const QString &nomPort)
{
    m_parent = parent;
    m_serial = new QSerialPort(parent);
    m_serial->setPortName(nomPort);
    connect(m_serial, SIGNAL(readRead()), this, SLOT(onReadyRead())); //connection entre le signal readyRead et le slot onReadyRead
    connect(m_serial, SIGNAL(error(QSerialPort::SerialPortError)), this, SLOT(onErreur(QSerialPort::SerialPortError)));
    qDebug() << "CSerial::L'objet m_Sp du port  " << nomPort << "est créé"; //affichage sur la console de la création du port
}
```

Ici, je connecte le signal *readyRead()* au slot *onReadyRead()*. Aussi, je connecte un signal *error* au slot *onErreur*. Enfin , j'affiche dans le debug un message informant de la création du port.

```
void CSerial::onReadyRead()
{
    QByteArray recu;
    recu = m_serial->read(100);
    mRecu.append(recu); //copie du signal reçu
    if(recu.endsWith('\n'))
    {
        emit sigData(mRecu); //emission du signal reçu
        mRecu.clear(); //efface le contenu du signal reçu
    }
}
```

Le slot *onReadyRead* permet d'émettre le signal *mRecu* si la variable *recu* se termine par **\n**.

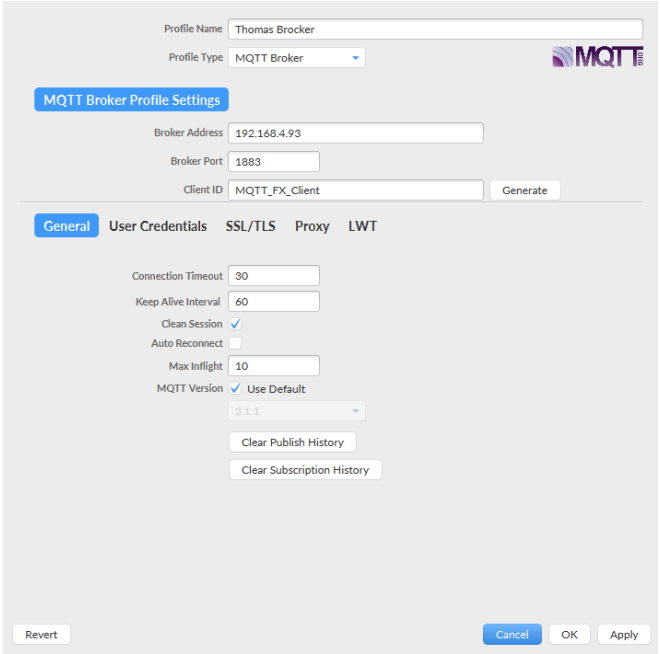
```
void app_gtc::on_sb_lampadaire_valueChanged(int arg1)
{
    if(ui->cb_segment->isChecked())
        mAdresse = "0";
    else mAdresse.setNum(arg1);
}
```

Le code ci-contre concerne le Spin Box segment du lampadaire : la variable *mAdresse* (initialement définie à 1) récupère la valeur *arg1* générée par le Spin Box.

MQTT.fx

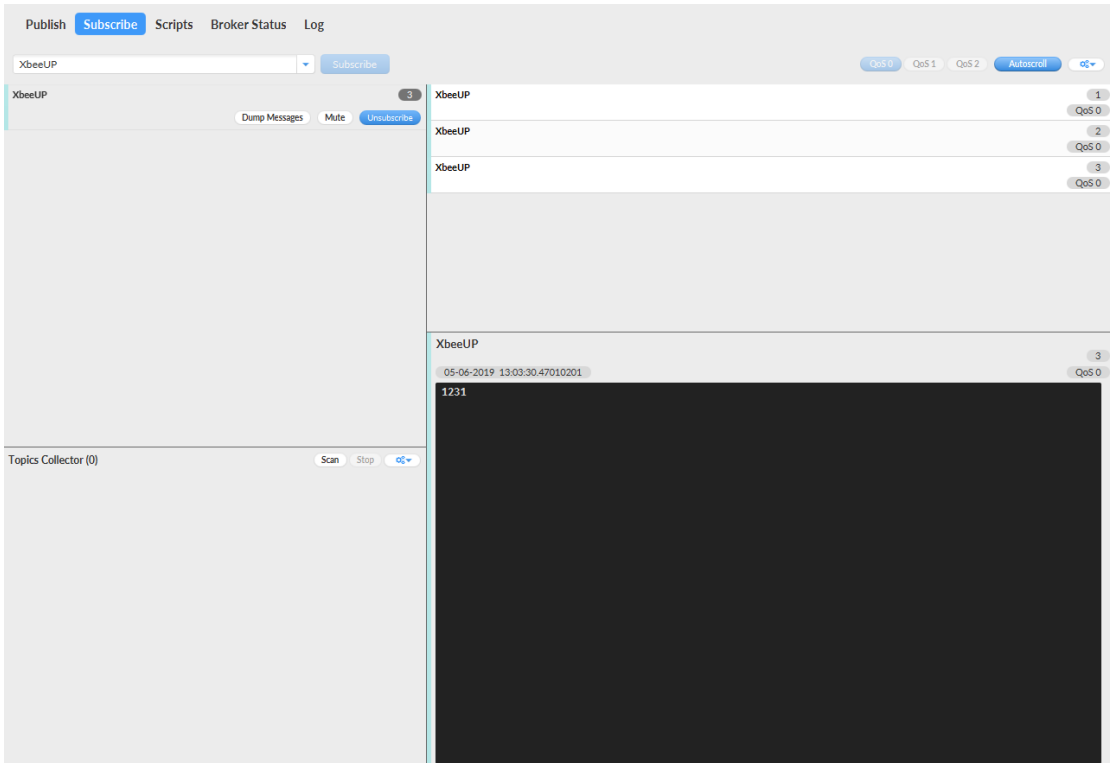
Afin de tester la communication avec le broker, j'ai eu recours à l'utilisation du logiciel **MQTT.fx** qui permet de s'abonner (dans la partie *Subscribe*) à un topic d'un broker (ici le broker ce nomme *Thomas Brocker* et le topic *XbeeUP*).

Pour sélectionner un broker, il faut entrer quelques paramètres dont le port du broker, l'adresse IP ou encore le type.



Paramètres du broker MQTT

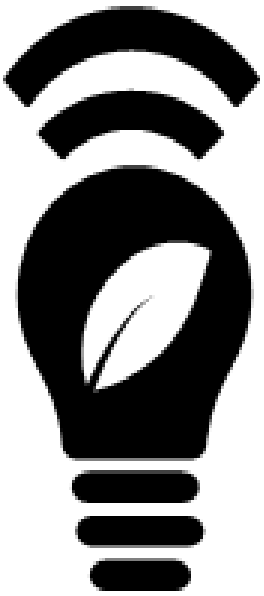
Toujours dans la même partie, on reçoit le message (au format JSON) transmis à partir du broker. Ce dernier envoie 1231 comme on peut le remarquer dans la console.



Test sur MQTT.fx

Projet API-GTC

Revue 3 – Annexes



Gestion Technique Centralisée d'un Eclairage Public avec communication XBee

FICHE RECETTE 1 , QUALIFICATION DU SYSTEME

Nom du système :	**GTC Lampadaires**	Code de la campagne de test :	
Technicien 1 : Technicien 2 : Technicien 3 :	IR1- GONZALEZ Thomas IR2- Weisseldinger Thomas EC1 – SALMON Robin	Date :	

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	**Activation du Programme de le Raspberry**
Objectif du scénario :	**Activation du Programme de le Raspberry afin de contrôler le module maitre.**

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

La base de données (côté Windows/GTC) doit être fonctionnelle. Les requêtes doivent aussi fonctionnées. Ainsi que une connection usb a un module xBee.

EXECUTION DU TEST

Description courte :	Entrer le mot de passe du programme et confirmer. (module étant branché) 'connecté' le module en fonction du port usb auquel il est relié. Une fois le programme activé, presser 'Verrouillé' afin de re sécurisé par le mot de passe. L'application sera alors totalement autonome. L'application GTC devra alors juste alimenté la base de données.
Résultats attendus :	Configuration automatique du module maitre. Envoie et formations des trames sur le réseau xBee par rapport aux données envoyées sur la base de données. Modification et actualisation de la base de données afin de ne pas faire la même commande.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	

FICHE RECETTE 2, QUALIFICATION DU SYSTEME

Nom du système :	**GTC Lampadaires**	Code de la campagne de test :	
Technicien 1 : Technicien 2 : Technicien 3 :	IR1- GONZALEZ Thomas IR2- Weisseldinger Thomas EC1 – SALMON Robin	Date :	

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	**Contrôle de la luminosité d'un lampadaire**
Objectif du scénario :	Changer la luminosité du lampadaire à partir de l'application GTC.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Les deux bases de données (côté Windows et côté Raspberry) doivent être fonctionnelles. Les requêtes doivent aussi fonctionnées. Le branchement du lampadaire et la construction de la trame doivent eux aussi marchés.

EXECUTION DU TEST

Description courte :	Suite à la sélection de la valeur de la luminosité sur l'application, cette même valeur doit s'afficher sur la base de données (côté Windows) et être récupérer par la base de données (côté Raspberry) pour, enfin, être réceptionné par le lampadaire.
Résultats attendus :	Le lampadaire s'éclaire à la luminosité sélectionné sur l'application.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	

FICHE RECETTE 3, QUALIFICATION DU SYSTEME

Nom du système :	**GTC Lampadaires**	Code de la campagne de test :	
Technicien 1 : Technicien 2 : Technicien 3 :	IR1- GONZALEZ Thomas IR2- Weisseldinger Thomas EC1 – SALMON Robin	Date :	

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	**Configuration des modules XBee**
Objectif du scénario :	Configurer les modules XBee sur l'application GTC

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

La lecture du/des port(s) doit se faire ainsi que la connexion du module XBee. Renseigner les paramètres de configuration.

EXECUTION DU TEST

Description courte :	L'utilisateur doit régler l'ensemble des paramètres (CE, JV, ID, DL) afin de configurer les modules
Résultats attendus :	Le module connecté est configuré comme un routeur ou comme coordinateur

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	

Fiche de Montage

Etape 1 - Montage du shield :

-Commencer par disposer tout les composants de type CMS sur la première face en suivant les numéros sérigraphies indiqués sur la carte (souder le support de pile manuellement de préférence par précaution).

-Vérifier la continuité de chacun des composants CMS de la première face puis passer à la face suivante (répéter les tests pour la deuxième face avant de continuer).

-Disposer et souder ensuite manuellement les borniers à leurs emplacements respectifs.

-Ajouter et souder le reste des composants (pins, relai, support)

Etape 2 – Tests du shield :

-Avant de tester la carte, ne pas oublier de vérifier la bonne continuité des masses et des différentes alimentations en s'aidant du schéma structurel fourni.

-Vérifier la bonne disposition de l'intégralité des composants et disposer le shield sur la carte Arduino sans l'alimenter.

-Disposer le module XBee à son emplacement et ne pas oublier de vérifier la pile avant d'alimenter.

-Alimenter la carte arduino avec une tension de 12V et effectuer des tests sur les composants principaux à l'aide d'un voltmètre.

Etape 3 – Mise en œuvre de la carte :

Connecter la carte au luminaire et relier chaque bornier au composant correspondant :

J1 : Capteur de courant

J2 : Détecteur de présence

J3 : Parafoudre

J4 : Détecteur de luminosité

J5 : Lampadaire

J6 : Annexe souhaité

J7 : J8 : Alimentation 12V

La communication entre l'application fournie et le luminaire peut maintenant être testé. La carte est prête à l'usage.

PLAN DE DEVELOPPEMENT

N° itération	Activités	Justification
Revue 1	Faire communiquer et configurer des modules XBee afin de crée un réseau fonctionnel sur XCTU. Identification de la topologie réseau.	Cette action était une priorité car le matériel devait être fonctionnel afin de pouvoir commencer les applications des différentes parties.
Revue 2	Installation et remplissage des bases de données (InfluxDB) ainsi que des annexes tel que : Grafana, MQTT.fx ou encore Telegraf. Conception de l'application de configuration des modules XBee.	Afin de pouvoir avoir toutes les indications et les adresses pour développer nos différentes applications qui communiqueront avec les bases de données nous nous voyons alors obligés d'installer l'ensemble des bases de données, car les informations sur les ordres donnés communiqueront par celles-ci.
Revue 3	Développement des applications communicantes (dont l'application GTC) avec les deux bases de données et test du contrôle du lampadaire.	Enfin les applications des différentes parties ont étaient créées et sont opérationnelles. Le test du lampadaire s'est fait avec le changement de la luminosité à partir de l'application GTC en passant par l'application de la Raspberry.