DOUBRE Maxime
BERTRAND Julien

Rapport TP Systèmes ambiants et mobiles

Objectif ;

L'objectif du TP est la création d'un système ambiant destiné à gérer la luminosité dans une salle. Pour ce faire, nous avons utilisé un serveur mqtt avec Mosquitto en tant que broker. Pour les publisher et les subscriber nous avons réalisé nous même ceux associés aux actionneurs et aux capteurs utilisés dans le système dont les interfaces ont aussi été réalisées par nos soins. Pour contrôler le bon fonctionnement de ceux-ci, nous avons utilisé MQTT box. En dessous seront développés 2 TP qui sont représentatifs de l'ensemble des compétences nécessaires à la réalisation de ce TP.

TP4 :

Dans le TP4 nous avons mis en œuvre tout ce que nous avions fait avant dans le TP2, c'est à dire l'envoi et la réception de messages via le broker avec un programme python ainsi que l'utilisation de MQTT box afin de faire les envois et réceptions via interface.
Contrairement au TP2, la nouveauté de ce TP était dans la capture des données grâce au MCP9808 pour la température et au TSL2561 pour la luminosité. Pour arriver à l'objectif du TP d'envoyer les données captées via le broker nous avons eu plusieurs phases.

La première a été de détecter les capteurs grâce à une fonction qui détecte tous les capteurs et qui en fait une liste. Pour ça nous avons fait notre fonction à partir de celle du sujet de TP et d'autres trouvées sur internet. Nous avons après cela testé que la fonction nous retournait bien la liste de tous les capteurs connectés au bus i2c.

Ensuite, la deuxième étape a été d'acquérir les données grâce aux capteurs.
De la même façon que pour la détection automatique des capteurs, nous avons cherché sur internet des exemples afin de faire nos propres fonctions. Nous avons donc deux fonctions pour extraire la température, dont une utilisant une bibliothèque adafruit pour arduino et une autre faite avec écriture dans les registres de notre MCP. Pour la luminosité nous avons une fonction de capture dans laquelle on écrit dans les regs de notre capteur avant de lire la donnée dans un registre aussi.

Après cela, la troisième étape a été de reprendre notre code du TP2 et d'y incorporer notre fonction de scan du bus ainsi que nos fonctions de captures des données. Une fois l'incorporation de ces fonctions faite dans le code nous avons testé notre code afin de vérifier son bon fonctionnement. Nous l'avons donc testé avec MQTT box afin de voir plus facilement les envois et les réceptions que dans le terminal avec tous les logs de notre programme python.
D'après nos deux capteurs nous avions donc les topics suivants :

- 1R1/014/temperature (/command)
- 1R1/014/luminosity (/command)

Les topics avec /command nous ont servi à recevoir dans notre programme les demandes de captures ainsi que les changements de fréquences comme dans le TP2. Nous avions pour ça dans notre Json un paramètre "order" contenant soit "capture" soit "frequence" avec une valeur associée pour le changement de fréquence.

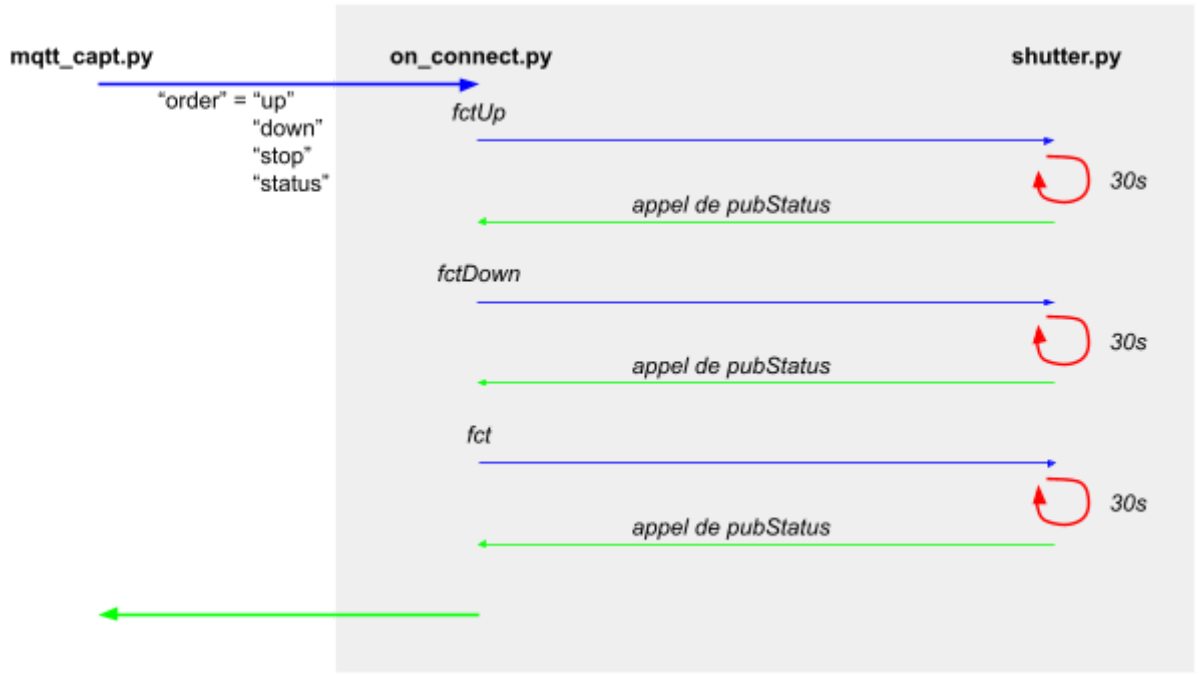
Une fois cette première grosse partie du TP réalisée, nous avons donc implémenté les interruptions permettant d'avoir les captures des données pour le TSL selon un intervalle. Nous avons fixé notre intervalle de base entre 0 et 1000. Au delà nous avons une interruption avec un envoi de la donnée capturée via le broker.
Pour configurer ces interruptions nous avons écrit dans les regs du capteur et incorporé des booléens dans le code pour gérer le mode fréquence ou le mode interrupt. Cependant nous avons eu pas mal de problèmes avec les interruptions ce qui a finalement été résolu par le fait de relancer le mode interrupt du capteur après chaque captation. Nous faisons cela dans notre fonction de capture de luminosité mais ça n'est utile que dans le cas où on est dans le mode interrupt.

TP7 :

Réutilisation des TP3 et des TP4 + implémentation d'une appli ambiante en python. Puis utilisation du TP5 et TP6 pour faire l'appli ambiante via red-node pour avoir une interface graphique.

Nous avons pour ce dernier TP réparti les tâches en deux : d'un côté le code de l'application ambiante en python et d'un autre côté les installations de docker et des différents conteneurs dont nous avions besoin pour la réalisation du TP.

Dans cette première partie nous avons donc dû utiliser tout ce que nous avions fait jusqu'à là. J'ai donc regroupé dans un dossier le TP3 et le TP4 afin d'avoir la capture de luminosité et la gestion des volets qui étaient déjà faites. J'ai ajouté à ça un petit programme python simulant la présence via un bouton poussoir et les gpio de la RPi et un autre programme python qui est l'application ambiante donc celle qui gère les différentes captures et actions des volets et de la lumière. Dans l'application on subscribe à tous les topics qui permettent d'avoir les données capturées et on publie dans tous les topics /command qui permettent d'envoyer des ordres aux capteurs et volets.
Dans notre fichier qui gère les volets, le nombre de ceux-ci peut être modifié en changeant simplement un nombre dans la boucle de création des volets dans le main de 'connect.py'. Cependant pour l'évaluation nous avons fait en sorte qu'ils soient tous reliés à la même led.



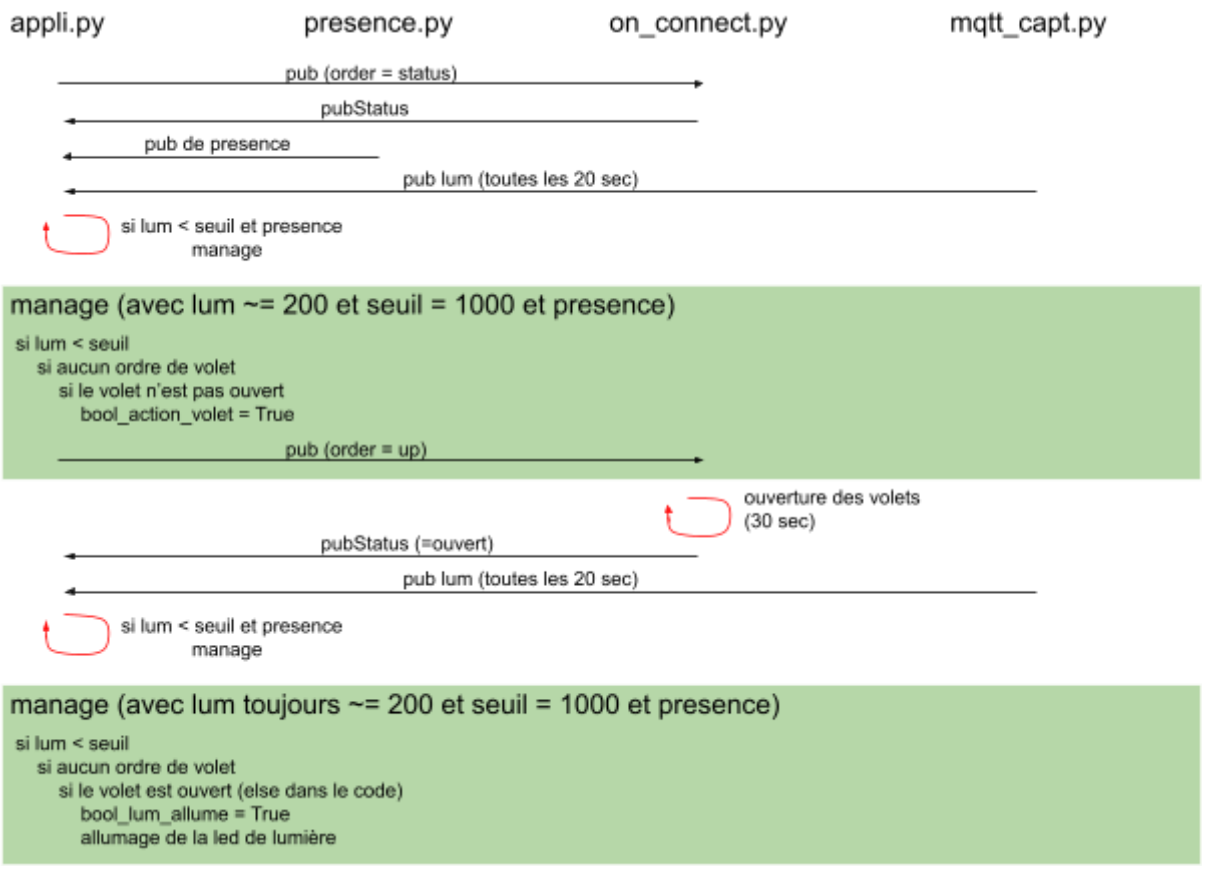
Dans mon "on_message", je gère tous les messages qui arrivent en sauvegardant les différentes données reçues dans des variables globales afin d'effectuer des traitements avec elles par la suite. A la fin de mon "on_message" j'ai fait une fonction "manage" qui permet justement d'effectuer des traitements selon les données reçues.
Par exemple, si la luminosité est en dessous d'un certain seuil fixé à l'exécutable de l'appli alors si le volet n'est pas en action et que son statut n'est pas ouvert alors j'envoie un publish aux volets afin qu'ils s'ouvrent. Une fois ouverts, si la luminosité n'est toujours pas au-dessus du seuil alors j'allume la lumière (simulée par une led).

Pour faire tourner l'application ambiante, il faut avoir 4 terminaux ouverts sur la RPi. Il faut dans un premier lancer le fichier "connect.py" qui gère les volets. Dans un deuxième le fichier "mqtt_capt.py" qui gère les captures de luminosité. Dans un troisième le fichier "presence.py" qui gère le boutons poussoir et dans le dernier le fichier "appli.py" qui est notre application python.

Exemple d'échange entre les capteurs et actionneurs à travers le TP7 :

Au début du programme on commence par demander le statut des volets afin de savoir si on peut les baisser ou les ouvrir ou les deux. Après ça on attend tout simplement les captures de luminosité qui arrivent toutes les 20 secondes au départ du programme :
A chaque fois qu'on reçoit un message, dans on_message on teste si la luminosité de la salle est dans l'intervalle décidé et si quelqu'un est présent dans la salle et si il y a quelqu'un mais que la luminosité n'est pas dans l'intervalle alors on passe par la fonction manage qui va gérer les différents cas.

Pour le lancement de l'appli on aura le schéma suivant :



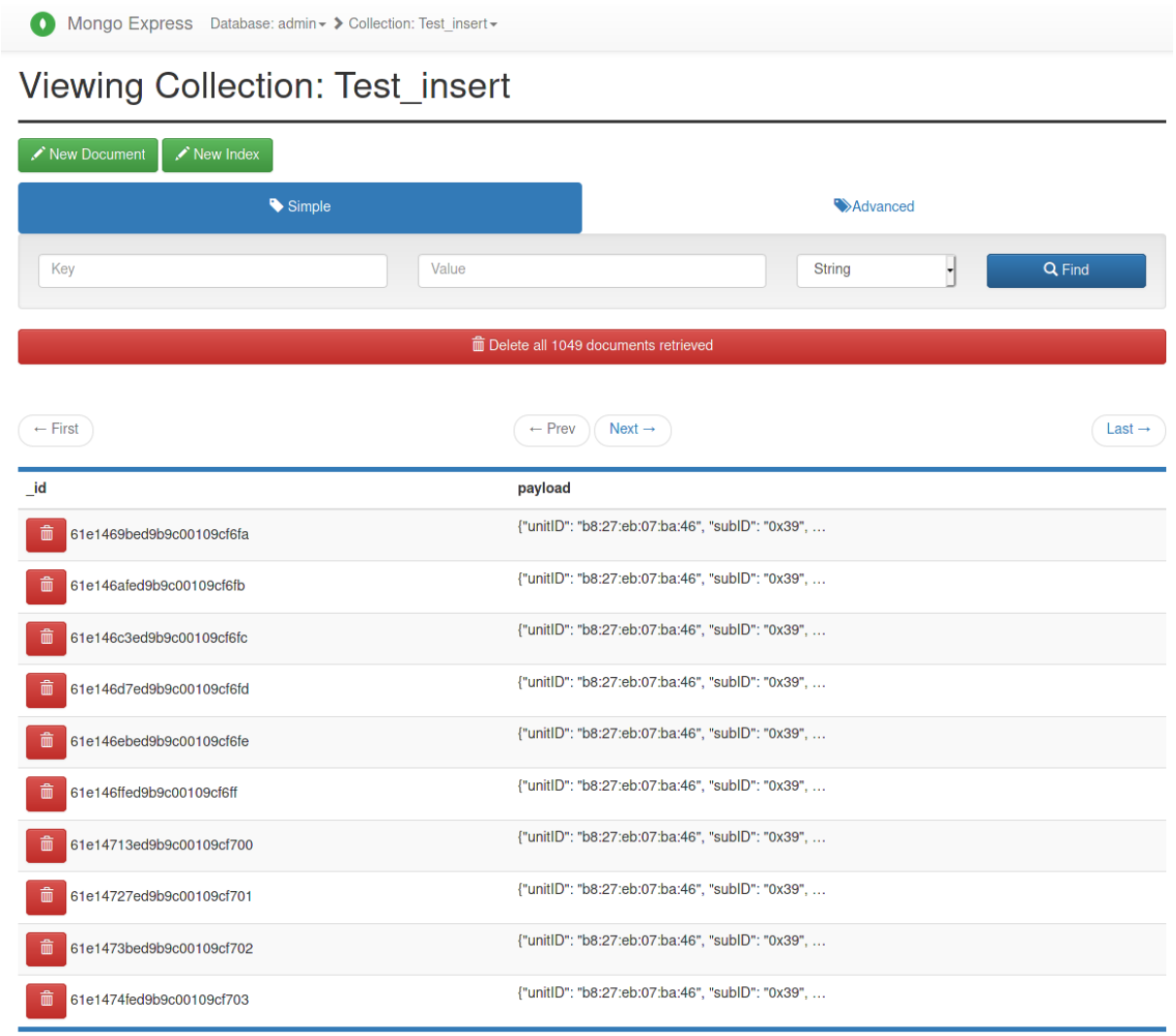
Dans cet exemple au début on demande les statuts des volets que l'on reçoit juste après. On stocke les valeurs reçues dans des variables globales. En parallèle, on reçoit la luminosité toutes les 20 secondes.
Tant qu'on ne reçoit pas de présence alors on reçoit les luminosités mais aucun traitement n'a lieu dans le programme.
Dès qu'on reçoit 'presence' alors si la dernière luminosité reçue n'était pas dans l'intervalle attendu on rentre dans 'manage'.
Une fois dans manage, on est dans le cas où luminosité < seuil et au début du programme les volets n'ont aucune action et ne sont pas ouverts (car between à l'init). Alors on envoie un publish afin d'ouvrir les volets et après 30 secondes on reçoit les statuts des volets qui sont maintenant 'open'
A réception des statuts si la luminosité n'est toujours pas dans l'intervalle alors on rentre à nouveau dans 'manage'.
Cette fois-ci les volets sont ouverts alors on va allumer la led qui simule notre lumière de salle.

Dans cette deuxième partie, les différents containers sont destinés au contrôle et à la récupération des données produites par le système. Un container contient une base de données mongodb destinés à stocker les données produites par les capteurs du système. Pour lier cette base de données au système nous utilisons un autre container contenant une instance de red node. De plus, afin de permettre un contrôle ergonomique du système, un container contenant une de domoticz est utilisé.
Tous les containers ont été créés sur la VM à partir de notre fichier yml et pour y accéder sur internet nous devions donc utiliser l'adresse IP de notre VM associé au port du container docker.
Nous avions donc les adresses suivantes :

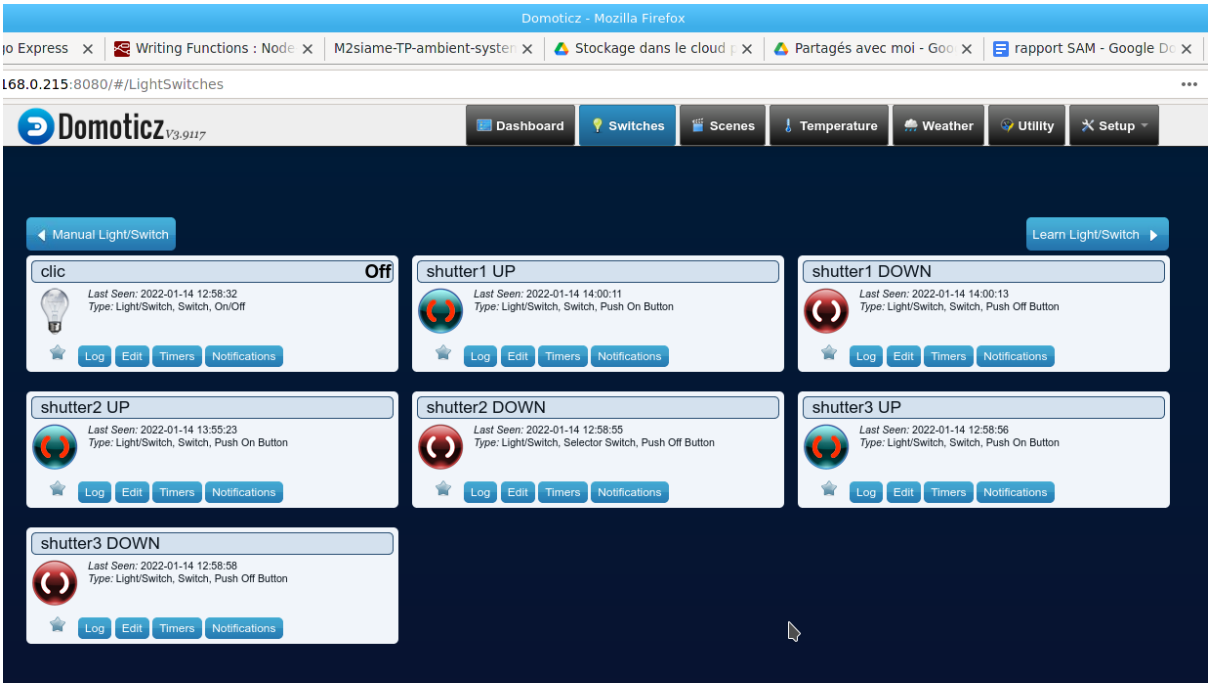
- Mongo Express : 192.168.0.215:27016
- MongoDB : 192.168.0.215:27017
- Domoticz : 192.168.0.215:8080
- NodeRed : 192.168.0.215:1880

Ci-dessous, nous avons fait des screens de notre database, de Domoticz et de Node Red.

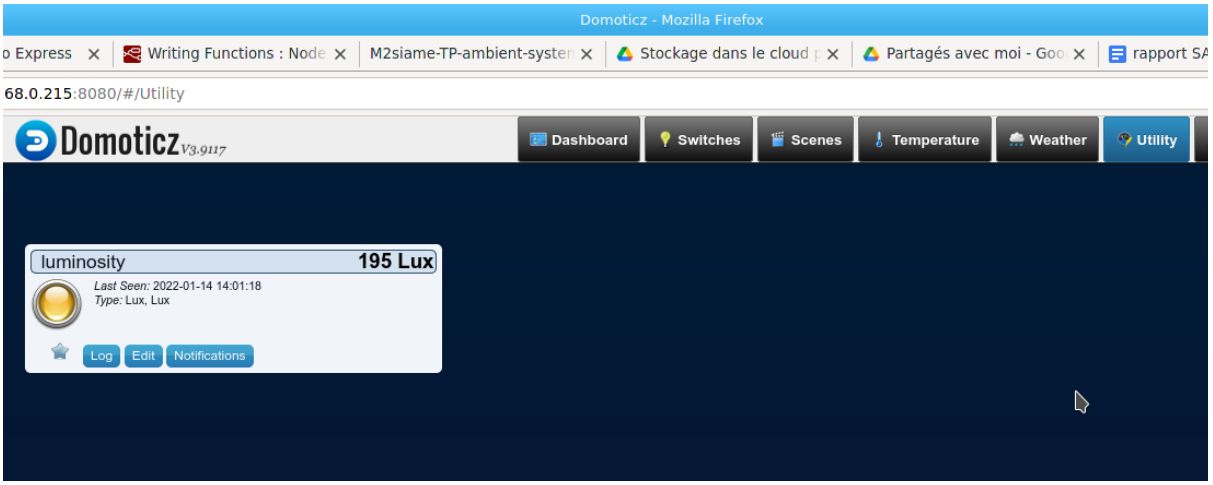
Le premier est un screen de notre data base Mongo Express qui montre le bon fonctionnement de l'envoi des données capturées vers notre base de données.
Nos données sont captées par le TSL avant d'être envoyées à travers le broker à red node qui ensuite les fait parvenir à la BDD.



Les deux prochaines captures d'écran sont des captures de notre Domoticz.
Il y a deux onglets différents.
Le premier est l'onglet des switchs dans lequel nous avons créé des interrupteurs pour la lumière et nos trois volets.
L'interrupteur pour la lumière publish sur le topic spécifique à domoticz (domoticz/out) quand il est activé ou qu'il est éteint. Notre instance de nodered a subscribe à ce topic et envoie un autre message mqtt sur le topic auquel est abonné le programme qui gère la lumière. Nous avons décidé de passer par cette intermédiaire car notre système n'a pas été conçu pour fonctionner exclusivement avec domoticz.
Les boutons pour les volets fonctionnent selon le même principe que celui qui est utilisé pour l'interrupteur pour la lumière à ceci près qu'il y a 2 boutons qui envoient un message uniquement quand ils sont activé car pour le contrôle des volets l'utilisateur peut être amené à appuyer 2 fois d'affilé sur le même boutons ce qui est un impossible avec un interrupteur.

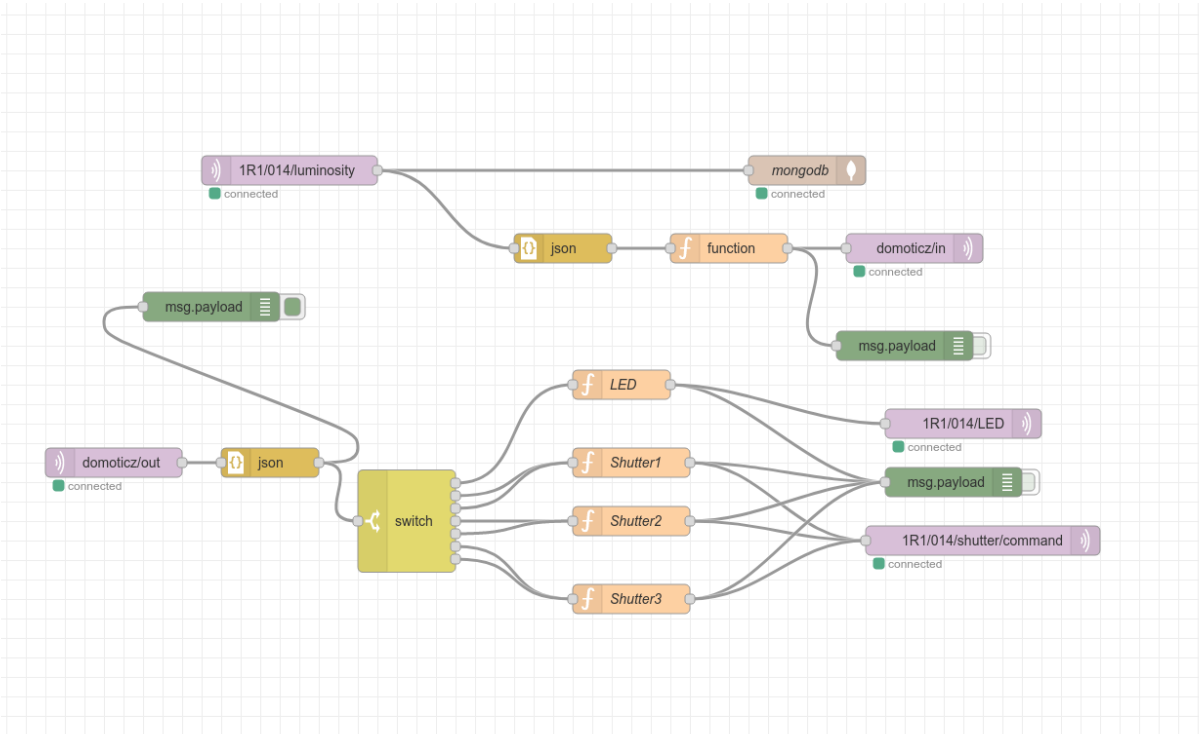


Le deuxième onglet est celui des utilitaires dans lequel nous avons fait un affichage de notre luminosité. C'est en fait les valeurs captées par notre capteur de lumière qui sont publish sur notre broker, récupérées par rednode puis envoyées par rednode vers Domoticz par MQTT sur le topic domoticz/in avant d'être affichées.



La dernière capture d'écran est une capture d'écran de notre red node.
On peut y voir les différentes connexions entre notre broker par exemple et mongoDB ainsi que Domoticz.

Par exemple, la partie du bas avec 'domoticz/out' fait en sorte que quand on appuie sur un bouton dans Domoticz on envoie le bon publish à travers le broker. Dans le cas du bouton d'allumage de la lumière, on appuie sur le bouton de Domoticz qui lui envoie un publish à travers le broker sur le topic domoticz/out, le message alors en JSON est alors converti en objet javascript plus facilement manipulable. Celui passe par un switch qui permet d'une part d'écarter les messages qui ne nous intéresse pas car domoticz n'a pas d'option de filtre sur les données envoyées sur domoticz/out et d'autre part de diriger les messages reçus. Le message est ensuite transformé afin qu'il puisse être lu par le programme qui va le recevoir et enfin, il est envoyé par MQTT sur le topic 1R1/014/LED. Ce message est reçu par le programme python LED.py dans notre RPi qui va allumer ou éteindre la led simulant la lumière selon son statut actuel.


1R1/014/luminosity
connected
mongodb
connected
json
function
domoticz/in
connected
msg.payload
msg.payload
LED
1R1/014/LED
connected
domoticz/out
connected
json
switch
Shutter1
msg.payload
Shutter2
1R1/014/shutter/command
connected
Shutter3