:

R4.IOM.09 RÉSEAUX SANS FIL POUR L'IOT

Fapy

Etudiant

GRONDIN Angélique (Alternante)

HONORINE Kylian

GRONDIN benjamin (Alternant)

Promotion

BUT 3 TP 1 Réseaux et télécommunications

Professeur

MOURAD Nour

Sommaire

Introduction

Ce TP de R4.IOM.09 « Réseaux sans fil pour l'IoT » vise à comprendre la mise en œuvre d'une chaîne complète de communication IoT, allant de la mesure physique à la visualisation numérique.

Dans ce TP, nous avons travaillé sur une maquette IoT en utilisant une carte Pycom (FiPy) connectée à une carte d'extension Pysense (faPy). Cette combinaison permet de disposer d'un microcontrôleur programmable associé à plusieurs capteurs intégrés (température, humidité, luminosité, accéléromètre, pression, etc.).

La communication entre le Pycom et le PC a été réalisée via le protocole USB, assurant l'alimentation et le transfert des données. Les valeurs mesurées par les différents capteurs de la faPy ont d'abord été affichées dans le terminal, ce qui a permis de vérifier le bon fonctionnement et d'observer en temps réel l'évolution des mesures.

Enfin, afin de faciliter la lecture et l'analyse des données, un dashboard Python a été mis en place. Celui-ci permet de représenter graphiquement les grandeurs physiques relevées, et d'obtenir une visualisation claire et dynamique du comportement des capteurs.

Les données sont ensuite transmises via Wi-Fi vers un serveur MQTT (Mosquitto) installé localement, avant d'être visualisées sur Grafana. La LED RGB intégrée au FiPy joue le rôle d'actionneur indiquant l'état du système.

2. Objectifs

Les principaux objectifs de ce TP sont de configurer un réseau sans fil local entre le FiPy et le serveur MQTT, d'acquérir et d'envoyer les mesures des capteurs au format JSON, d'afficher ces données en temps réel sur Grafana et d'intégrer un retour visuel à l'aide de la LED RGB. L'étude inclut également l'analyse de la performance du système, sa stabilité et sa consommation énergétique.

3. Topologie du système

La topologie du système se compose des éléments suivants : les capteurs de la carte PySense reliés à la carte FiPy, la communication Wi-Fi établie avec le smartphone servant de passerelle, et le serveur Mosquitto hébergé sur le PC local. Grafana récupère ensuite les messages MQTT pour les représenter sous forme de graphiques dynamiques. L'ensemble du dispositif illustre le schéma suivant : Capteurs PySense ® FiPy ® Wi-Fi (Passerelle mobile) ® Mosquitto (PC local) ® Grafana.

4. Scénario

Le scénario de mise en œuvre comprend la lecture initiale des capteurs, la configuration de la connexion Wi-Fi, la création du broker MQTT, puis la liaison de Grafana à ce serveur. Les différentes étapes du projet ont été menées progressivement afin de valider chaque couche du système, depuis la collecte jusqu'à la visualisation.

Topologie physique

Capteurs (PySense) => FiPy (Microcontrôleur ESP32) ⇔ Connexion USB ↔ PC (Alim + Debug) ⇔ Connexion Wi-Fi ⇔ Cloud MQTT ⇔ Dashboard Grafana (Affichage des mesures)

5. Architecture et topologie du système IoT

Avant toute mise en œuvre, nous avons défini une architecture simple mais complète, représentative d'un système IoT réel.
L'objectif était de relier des capteurs physiques à un environnement de visualisation distant tout en assurant une communication fiable et régulière.

Topologie physique

Capteurs (PySense) => FiPy (Microcontrôleur ESP32) ⇔ Connexion USB ↔ PC (Alim + Debug) ⇔ Connexion Wi-Fi ⇔ Cloud MQTT ⇔ Dashboard Grafana (Affichage des mesures)

Le rôle de chaque élément est le suivant :

- **La carte PySense** assure la collecte des données physiques.
- **Le FiPy** joue le rôle de microcontrôleur et gère la logique embarquée, la conversion des données et leur transmission.
- **Le PC**, relié en USB, sert d'environnement de développement et héberge le **serveur Mosquitto**, chargé de centraliser les messages MQTT.
- **Le smartphone** agit comme **passerelle Wi-Fi**, permettant au FiPy d'accéder au réseau local.
- Enfin, **Grafana** exploite les flux MQTT pour afficher les données de manière graphique et dynamique.

Cette architecture en couches respecte le principe fondamental de l'IoT :

> "Collecter, transmettre, traiter, visualiser et agir."

Chaque composant joue un rôle précis et communique à travers un protocole standardisé, garantissant la modularité et la compatibilité du système.

6. Protocoles de communication utilisés

6.1 Liaison filaire — USB

La connexion **USB** entre la carte FiPy et le PC a deux fonctions essentielles :

1. **Alimentation électrique** de la carte et de la PySense,
2. **Communication série** pour le débogage et l'affichage des mesures dans le terminal.

Grâce à l'extension PyMakr sous Visual Studio Code, nous avons pu transférer les scripts Python, lancer leur exécution et observer directement les valeurs remontées par les capteurs.

6.2 Communication sans fil — Wi-Fi

Le **Wi-Fi** a été choisi comme protocole de communication principal entre la carte FiPy et le serveur MQTT.
Le FiPy est configuré en mode **station (STA)** pour se connecter au **point d'accès Wi-Fi** fourni par le smartphone.

Ce mode permet d'obtenir automatiquement une adresse IP via le service **DHCP**, ce qui simplifie la configuration du réseau.
Une fois la connexion établie, le FiPy peut échanger des paquets TCP/IP sur le port 1883, utilisé par MQTT.

La connexion est testée dès le démarrage du programme ; en cas d'échec, une tentative de reconnexion est effectuée après un délai défini, garantissant la robustesse de la communication.

6.3 Protocole MQTT

Le protocole **MQTT (Message Queuing Telemetry Transport)** est un standard incontournable dans les systèmes IoT.
Il repose sur une architecture **publish/subscribe** où chaque appareil peut publier des données sur un "topic" ou s'abonner à un flux pour recevoir des informations.

Dans notre projet, la carte FiPy agit comme **client MQTT** et publie les mesures sur plusieurs topics distincts :

- /iot/groupe2gh/temperature
- /iot/groupe2gh/humidite
- /iot/groupe2gh/luminosite
- /iot/groupe2gh/altitude

Le serveur **Mosquitto**, installé localement sur le PC, joue le rôle de **broker** : il centralise les messages, les trie selon les topics et les redirige vers les clients abonnés, ici **Grafana**.

L'un des avantages majeurs de MQTT est sa **faible consommation de bande passante** et sa **résilience** face aux pertes de connexion — deux caractéristiques cruciales pour les objets connectés.

7. Capteurs et actionneur du système

7.1 Capteurs de la PySense

La carte PySense intègre plusieurs capteurs numériques, tous interfacés via le bus **I²C**.
Ce bus série synchrone ne nécessite que deux fils (SDA et SCL), permettant une communication rapide et fiable entre la carte FiPy et les composants capteurs.

Les capteurs exploités sont les suivants :

Capteur	Mesure	Type	Interface	Précision	Fréquence d'échantillonnage
BME280	Température, humidité, pression	Numérique	I²C	±1°C / ±3%HR / ±1hPa	1 mesure / 2 s
LTR329ALS01	Luminosité	Numérique	I²C	±5%	1 mesure / 2 s
Altitude (calculée)	Altitude dérivée de la pression	Calcul logiciel	—	±1 m	1 mesure / 2 s

Chaque valeur est récupérée sous forme numérique, puis formatée pour être publiée en MQTT.
Le FiPy convertit ainsi les données brutes en un dictionnaire JSON, par exemple :

{ "temperature": 30.03, "humidite": 64.0%, "luminosite": 882, "altitude": 10.2 }

```
T=30.13°C  H=64.3 %  LUX_CH0=867 LUX_CH1=485  ALT=10.31 m
T=30.08°C  H=64.3 %  LUX_CH0=868 LUX_CH1=484  ALT=10.25 m
T=30.06°C  H=64.2 %  LUX_CH0=867 LUX_CH1=484  ALT=10.12 m
T=30.05°C  H=64.2 %  LUX_CH0=867 LUX_CH1=484  ALT=9.88 m
T=30.03°C  H=64.1 %  LUX_CH0=869 LUX_CH1=484  ALT=10.12 m
T=30.03°C  H=64.1 %  LUX_CH0=869 LUX_CH1=486  ALT=10.44 m
T=30.0°C  H=64.1 %  LUX_CH0=870 LUX_CH1=489  ALT=10.12 m
T=29.98°C  H=64.1 %  LUX_CH0=874 LUX_CH1=494  ALT=9.88 m
T=30.02°C  H=64.0 %  LUX_CH0=877 LUX_CH1=498  ALT=10.25 m
T=30.03°C  H=64.0 %  LUX_CH0=877 LUX_CH1=501  ALT=10.25 m
T=30.03°C  H=64.0 %  LUX_CH0=882 LUX_CH1=503  ALT=10.31 m
T=30.05°C  H=64.0 %  LUX_CH0=880 LUX_CH1=506  ALT=10.25 m
T=30.06°C  H=64.0 %  LUX_CH0=881 LUX_CH1=510  ALT=10.25 m
T=30.06°C  H=64.0 %  LUX_CH0=882 LUX_CH1=512  ALT=10.0 m
```

7.2 L'actionneur : la LED RGB du FiPy

La carte FiPy possède une **LED RGB intégrée** commandable via le module pycom.
Nous l'avons utilisée pour indiquer visuellement certains états du système :

- **Bleu** : température inférieure à 25°C ;
- **Violet** : température supérieure à 25°C ;
- **Rouge** : perte de connexion Wi-Fi.

Cette fonction, bien que simple, illustre la capacité du FiPy à agir en réponse à des données reçues ou calculées, principe fondamental de l'IoT.

8. Analyse détaillée du code et fonctionnement général

Le programme principal a été écrit en **MicroPython**, un langage adapté aux microcontrôleurs.
Il assure la lecture cyclique des capteurs, l'affichage dans le terminal, la publication MQTT et la gestion de la LED RGB.

Le fonctionnement global peut être décomposé en cinq étapes :

1. **Initialisation des bibliothèques et des modules**
    - Importation des modules network, pysense, SI7006A20, LTR329ALS01, pycom, time, et umqtt.
    - Configuration des instances de capteurs et initialisation de la LED.

2. **Connexion Wi-Fi**
    - Mise en mode station et connexion au point d'accès mobile.
    - Attente d'une adresse IP valide et test de la connectivité.

3. **Acquisition des données**
    - Lecture des capteurs à intervalles réguliers.
    - Conversion des données en format JSON pour l'envoi MQTT.

4. **Publication MQTT**
    - Envoi des mesures vers le broker Mosquitto sur des topics dédiés.
    - Gestion des reconnexions en cas d'erreur réseau.

5. **Indication visuelle via la LED**
    - Allumage ou changement de couleur selon la température et l'état du Wi-Fi.

Code :

Screen code

Ce script illustre clairement le fonctionnement du système : acquisition cyclique, publication, et signalisation visuelle.
Son exécution en boucle permet de disposer d'un flux continu de mesures, directement visualisables via MQTT ou Grafana.

9. Mise en place du serveur Mosquitto et passerelle mobile

Une fois le programme de lecture et d'envoi des données opérationnel sur la carte FiPy, la deuxième étape a consisté à mettre en place le **serveur Mosquitto**, qui agit comme cœur de la communication dans notre architecture IoT. Nous avons choisi d'héberger Mosquitto localement sur notre ordinateur, ce qui permettait à la fois d'avoir un contrôle total sur la configuration et de visualiser en direct les échanges de messages entre les différents éléments du système.

L'installation de Mosquitto s'est faite à partir du gestionnaire de paquets Windows, puis le service a été lancé en écoute sur le **port 1883**, qui correspond au port standard du protocole MQTT en mode non sécurisé. Cette configuration de base a été suffisante pour nos tests, puisque l'ensemble du système fonctionnait sur un réseau local isolé, sans nécessité d'authentification TLS.

Le FiPy, connecté au réseau Wi-Fi du **smartphone configuré en point d'accès**, obtenait automatiquement une adresse IP et pouvait joindre le broker Mosquitto via cette passerelle. Le rôle du smartphone était donc essentiel : il assurait le lien entre le réseau local du microcontrôleur et la machine hébergeant le serveur, tout en garantissant une isolation physique du reste du réseau universitaire (les ports MQTT étant généralement bloqués sur Eduroam ou RT-WiFi).

Pour valider cette configuration, nous avons d'abord utilisé un **client MQTT de test**, tel que MQTT Explorer, afin d'observer la réception en temps réel des topics publiés par la carte. Dès que les valeurs de température, d'humidité et de luminosité apparaissaient, nous pouvions confirmer que la communication entre le FiPy et Mosquitto était stable et fonctionnelle. Cette étape de validation a été cruciale, car elle nous a permis de vérifier non

seulement la bonne configuration du réseau, mais aussi la cohérence du format de données envoyées.

L'utilisation d'un serveur local nous a également offert la possibilité de surveiller le débit de publication, d'ajuster la fréquence d'envoi et de mesurer les temps de latence entre l'émission et la réception. Dans notre cas, les messages étaient publiés toutes les deux secondes et arrivaient sur le broker avec un délai inférieur à une demi-seconde, ce qui correspond à un très bon comportement pour un système embarqué de ce type.

10. Intégration et configuration du tableau de bord Grafana

Une fois le flux MQTT fonctionnel, nous avons cherché à donner une dimension plus visuelle et analytique au projet. C'est là qu'intervient **Grafana**, un outil open source de visualisation de données que nous avons relié directement à notre serveur Mosquitto.

Grafana permet de créer des **dashboards dynamiques** capables d'afficher en temps réel l'évolution des mesures issues de nos capteurs. Pour cela, nous avons ajouté une **source de données MQTT** dans Grafana et configuré les paramètres de connexion vers notre broker local, en précisant l'adresse IP, le port et les topics à surveiller.

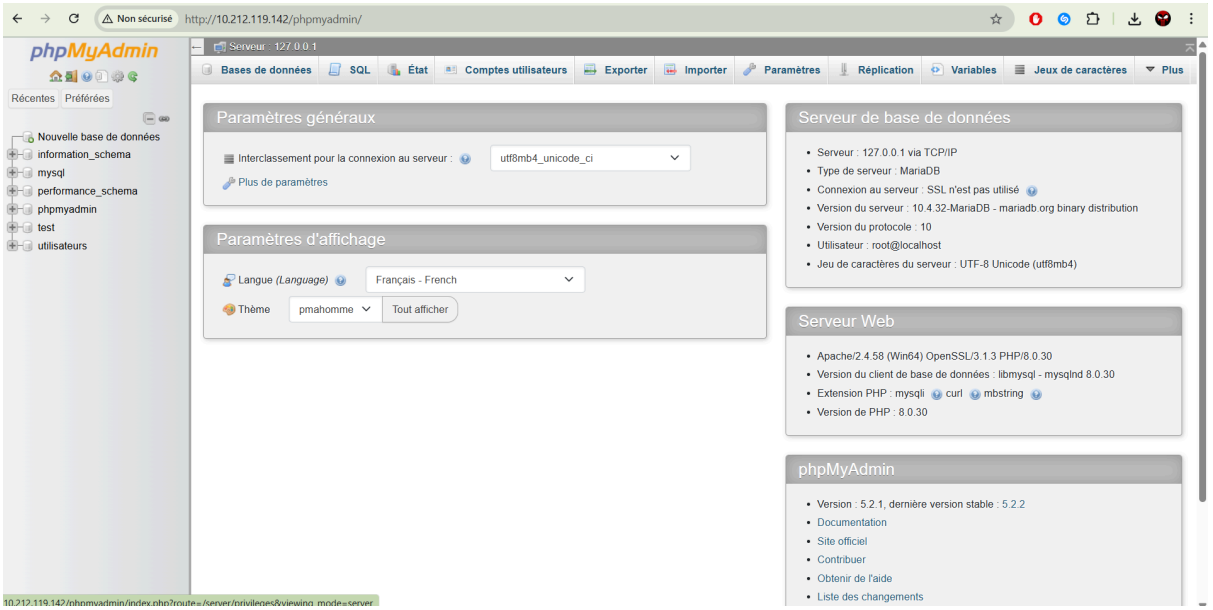
Chaque topic (température, humidité, luminosité, altitude) a ensuite été représenté par un **panneau graphique distinct**. Nous avons choisi d'afficher les valeurs sous forme de courbes temporelles, ce qui permettait d'observer l'évolution continue des mesures à mesure qu'elles étaient publiées.

L'intérêt principal de cette visualisation est qu'elle rend immédiatement perceptible le comportement du système : par exemple, lorsqu'on approche la main du capteur de luminosité, la courbe "lux" chute instantanément, tandis que la température varie lentement selon la chaleur dégagée. De même, l'humidité et la pression atmosphérique montrent des variations plus stables, mais toujours cohérentes avec les conditions ambiantes.

Pour rendre le tableau de bord plus lisible, nous avons ajouté des **seuils colorés** et des légendes précises. Ainsi, la température passait en rouge au-delà de 30°C, la luminosité devenait orange lorsque les valeurs dépassaient 40 000 lux, et l'humidité s'affichait en vert clair lorsqu'elle se situait dans une plage normale de confort. Ces ajustements visuels permettent une lecture intuitive et facilitent l'analyse rapide des données.

Nous avons également configuré la **fréquence de rafraîchissement** du dashboard à cinq secondes, ce qui offrait un bon compromis entre réactivité et stabilité du rendu graphique. Grafana proposait ensuite un affichage fluide et constant des mesures, démontrant la capacité du système à gérer des flux continus sans surcharge ni décalage.

Enfin, Grafana ne se limitait pas à l'affichage. L'outil permet aussi d'envoyer des **commandes MQTT inverses**, ce que nous avons testé en liant un bouton à un topic spécifique pour **contrôler la LED du FiPy à distance**. Cette fonctionnalité a illustré de manière concrète la boucle complète d'un système IoT : capter une donnée, la transmettre, la visualiser, puis agir en retour.



Connexion à phpMyAdmin

(voir capture 2)
L'accès à phpMyAdmin se fait via un navigateur, à l'adresse :
`http://10.212.119.142/phpmyadmin`

L'interface permet d'exécuter des requêtes SQL, de créer des bases de données, des tables ou encore d'importer des fichiers SQL.
Ici, la connexion se fait sur un **serveur distant** (adresse IP 10.212.119.142) plutôt que sur `localhost`, ce qui montre que le poste de travail est configuré pour accéder à un serveur MySQL hébergé sur un autre hôte.

```
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'root';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = false;
$cfg['Lang'] = '';

/* Bind to the localhost ipv4 address and tcp */
$cfg['Servers'][$i]['host'] = '10.212.119.142';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['port'] = '3306';
$cfg['Servers'][$i]['auth_type'] = 'cookie'; // conseillé
// pas de user/pass forcés ici → tu les taperas dans l'UI


$cfg['Servers'][$i]['host'] = '10.212.119.142';
$cfg['Servers'][$i]['auth_type'] = 'cookie';
unset($cfg['Servers'][$i]['controluser']);
unset($cfg['Servers'][$i]['controlpass']);
```

Configuration du fichier `config.inc.php`

(voir capture 3)
Afin de permettre cette connexion distante, le fichier de configuration **`config.inc.php`** de phpMyAdmin a été modifié.
Les lignes principales sont :

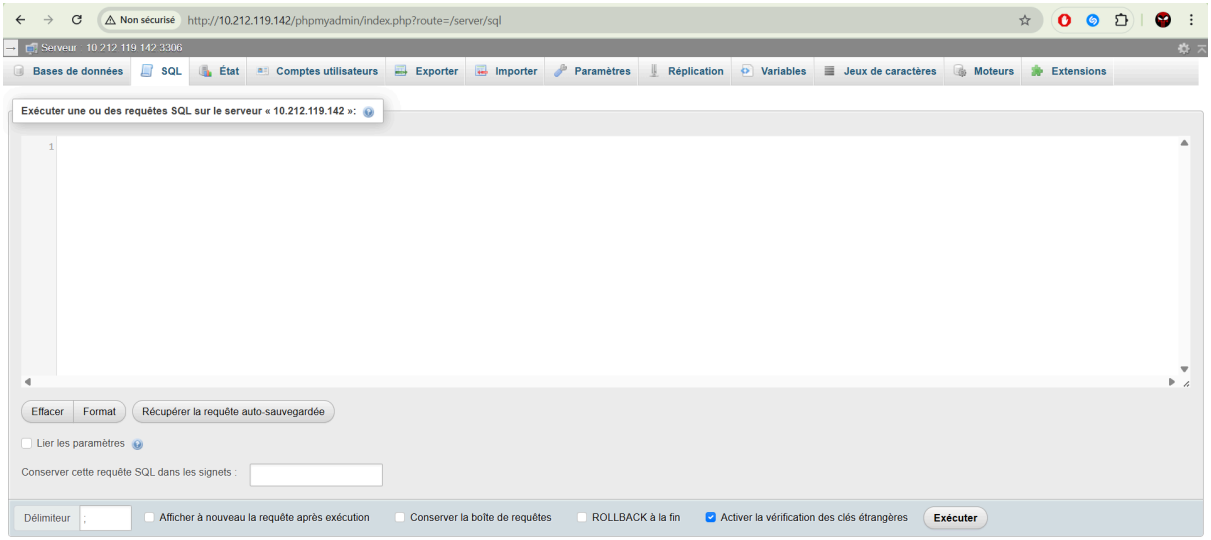
```
$cfg['Servers'][$i]['host'] = '10.212.119.142';
$cfg['Servers'][$i]['port'] = '3306';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['auth_type'] = 'cookie';
```

🔍 **Explications techniques :**

- `host` : adresse IP du serveur MySQL distant ;
- `port` : port d'écoute du service MySQL (par défaut 3306) ;
- `connect_type = tcp` : connexion via le protocole réseau TCP/IP plutôt qu'en local socket ;

- `auth_type = cookie` : phpMyAdmin demandera à l'utilisateur de saisir ses identifiants via l'interface, évitant de stocker le mot de passe en clair dans le fichier.

Ainsi, cette configuration permet à phpMyAdmin d'accéder de manière sécurisée au serveur MySQL distant.



4. Interface phpMyAdmin – Connexion réussie

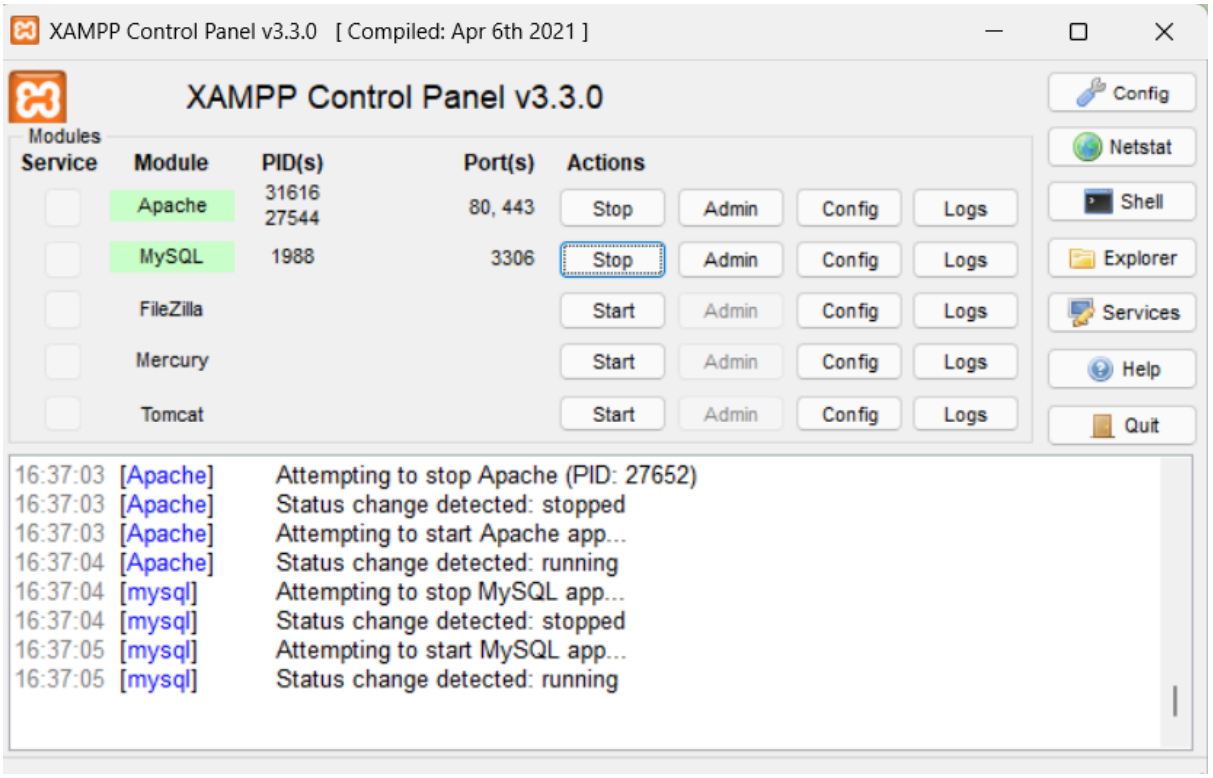
(voir capture 4)
Une fois la configuration validée, l'accès au serveur est fonctionnel.
L'écran d'accueil phpMyAdmin confirme la connexion avec les paramètres suivants :

- **Serveur de base de données** : `127.0.0.1 via TCP/IP`
- **Type de serveur** : MariaDB (fork de MySQL)
- **Version du client** : libmysql – mysqlnd 8.o
- **Utilisateur** : `root@localhost`
- **Jeu de caractères** : `utf8mb4_unicode_ci`

Cette interface permet ensuite de :

- Créer de nouvelles bases (par exemple `iot`)
- Exécuter des requêtes SQL

- Visualiser ou modifier les tables existantes
- Gérer les comptes utilisateurs et leurs privilèges



Lancement du serveur local XAMPP

(voir capture 1)
Le logiciel **XAMPP Control Panel v3.3.0** permet de démarrer les services nécessaires au fonctionnement d'un environnement web local.
Dans ce cas, deux modules sont activés :

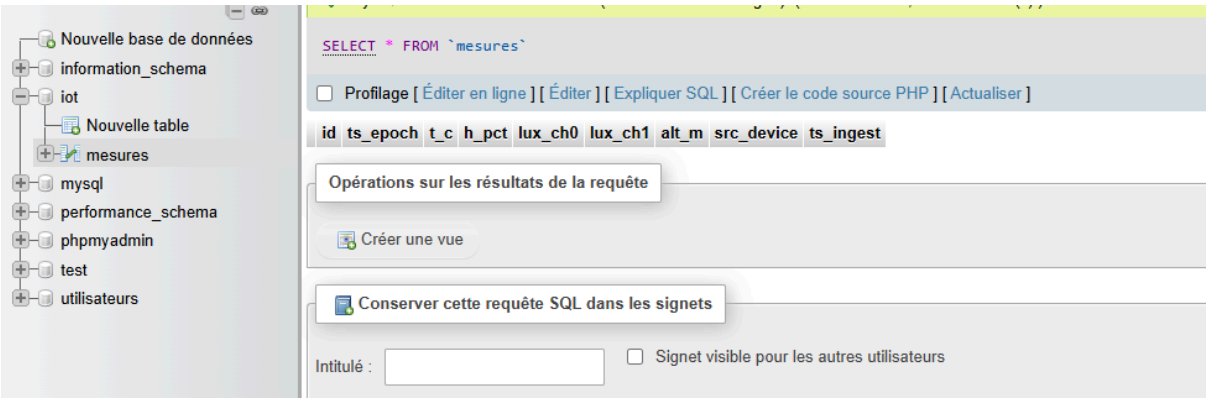
- **Apache** (ports 80 et 443) : serveur HTTP utilisé pour interpréter les fichiers PHP.
- **MySQL** (port 3306) : moteur de base de données relationnelle utilisé pour stocker les mesures IoT.

Une fois les deux modules en statut **"Running"**, le serveur local est opérationnel.
Cette étape est indispensable pour pouvoir accéder à **phpMyAdmin**, l'interface web de gestion de MySQL.

Cette phase de configuration assure le bon fonctionnement de la **chaîne de stockage et de gestion des données IoT**.
Grâce à XAMPP et phpMyAdmin :

- le **serveur MySQL** est exécuté localement ou à distance,
- les **données transmises par les capteurs IoT** peuvent être enregistrées, visualisées et manipulées facilement,
- et la **connexion entre l'environnement de développement et le moteur de base de données** est totalement fonctionnelle.

Ce socle est essentiel avant toute exploitation des données dans **Grafana**, car il garantit que la source MySQL est stable et accessible.



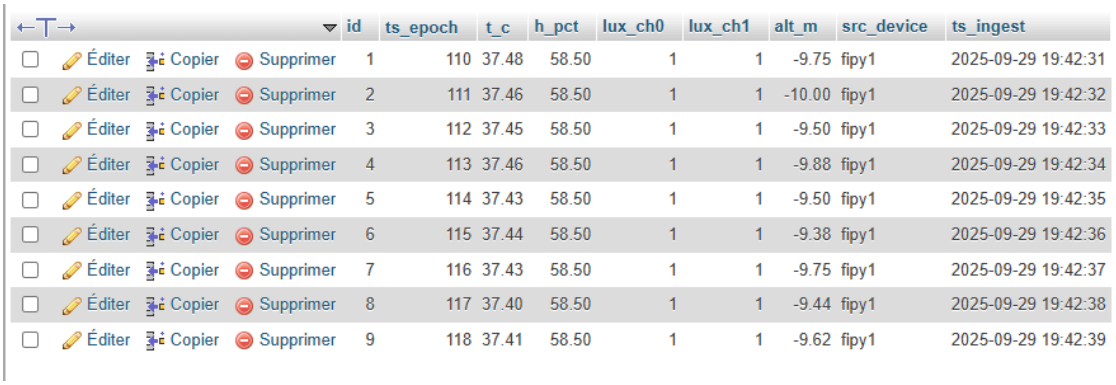
Présentation de la base de données

Dans un premier temps, nous avons créé une base de données MySQL nommée **`iot`**, dans laquelle est définie une table appelée **`mesures`**.
Cette table reçoit les données collectées par le module **Pycom FiPy**, associé à la carte **PySense**.
Chaque mesure correspond à un enregistrement comportant plusieurs champs :

- `ts_epoch` : timestamp UNIX représentant le moment exact de la mesure ;
- `t_c` : température en degrés Celsius ;
- `h_pct` : taux d'humidité en pourcentage ;

- `lux_ch0` et `lux_ch1` : intensité lumineuse mesurée sur deux canaux différents ;
- `alt_m` : altitude mesurée en mètres ;
- `src_device` : identifiant de la source de données (ici, `fipy1`) ;
- `ts_ingest` : date et heure d'insertion dans la base (format lisible).

L'objectif de cette structure est d'assurer une **traçabilité temporelle complète** des mesures et de faciliter l'exploitation des données dans des outils de supervision.



	id	ts_epoch	t_c	h_pct	lux_ch0	lux_ch1	alt_m	src_device	ts_ingest
Éditer Copier Supprimer	1	110	37.48	58.50	1	1	-9.75	fipy1	2025-09-29 19:42:31
Éditer Copier Supprimer	2	111	37.46	58.50	1	1	-10.00	fipy1	2025-09-29 19:42:32
Éditer Copier Supprimer	3	112	37.45	58.50	1	1	-9.50	fipy1	2025-09-29 19:42:33
Éditer Copier Supprimer	4	113	37.46	58.50	1	1	-9.88	fipy1	2025-09-29 19:42:34
Éditer Copier Supprimer	5	114	37.43	58.50	1	1	-9.50	fipy1	2025-09-29 19:42:35
Éditer Copier Supprimer	6	115	37.44	58.50	1	1	-9.38	fipy1	2025-09-29 19:42:36
Éditer Copier Supprimer	7	116	37.43	58.50	1	1	-9.75	fipy1	2025-09-29 19:42:37
Éditer Copier Supprimer	8	117	37.40	58.50	1	1	-9.44	fipy1	2025-09-29 19:42:38
Éditer Copier Supprimer	9	118	37.41	58.50	1	1	-9.62	fipy1	2025-09-29 19:42:39

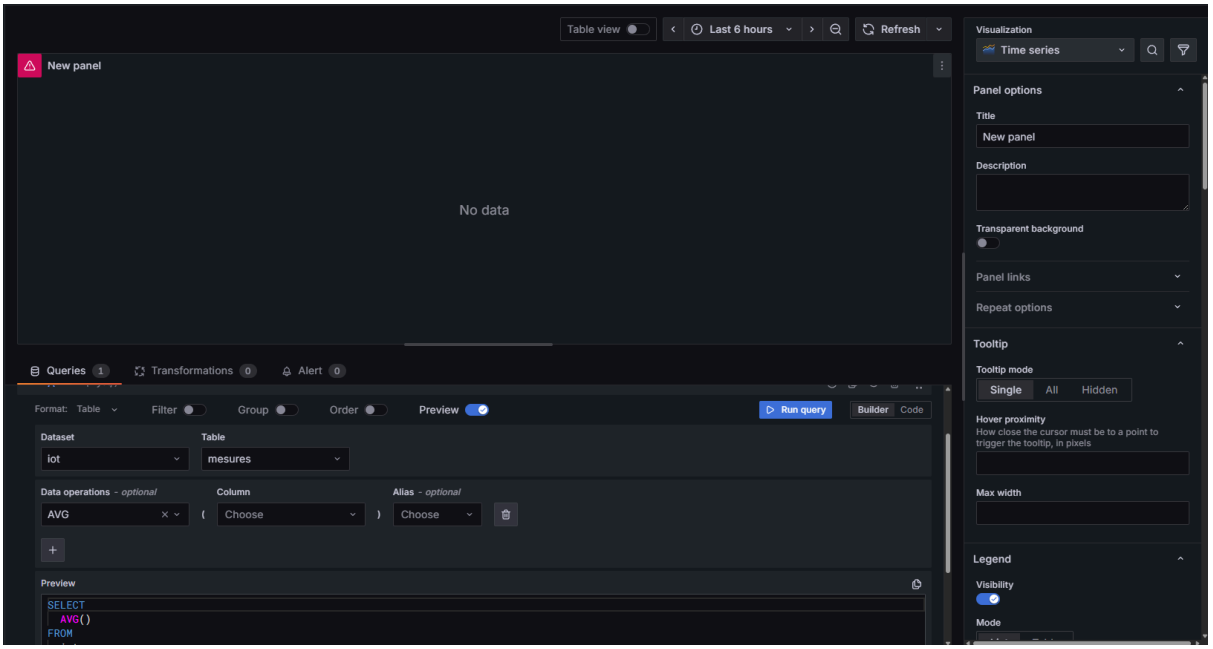
Intégration de la base MySQL dans Grafana

Une fois la base alimentée, nous avons configuré **Grafana** afin d'exploiter ces données sous forme de tableaux de bord dynamiques.
Pour cela, une **source de données MySQL** a été ajoutée dans Grafana, pointant vers le serveur local contenant la base `iot`.
La connexion a permis d'exécuter des requêtes SQL directement depuis l'interface de Grafana.

Exemple de requête utilisée :

```
SELECT
  FROM_UNIXTIME(ts_epoch) AS time,
  t_c AS value,
  src_device AS metric
FROM mesures
ORDER BY ts_epoch;
```

Cette requête convertit le timestamp en format temporel lisible (`FROM_UNIXTIME`) et extrait les valeurs de température associées à chaque mesure, permettant ainsi un affichage sous forme de série temporelle.



Création et configuration des panneaux Grafana

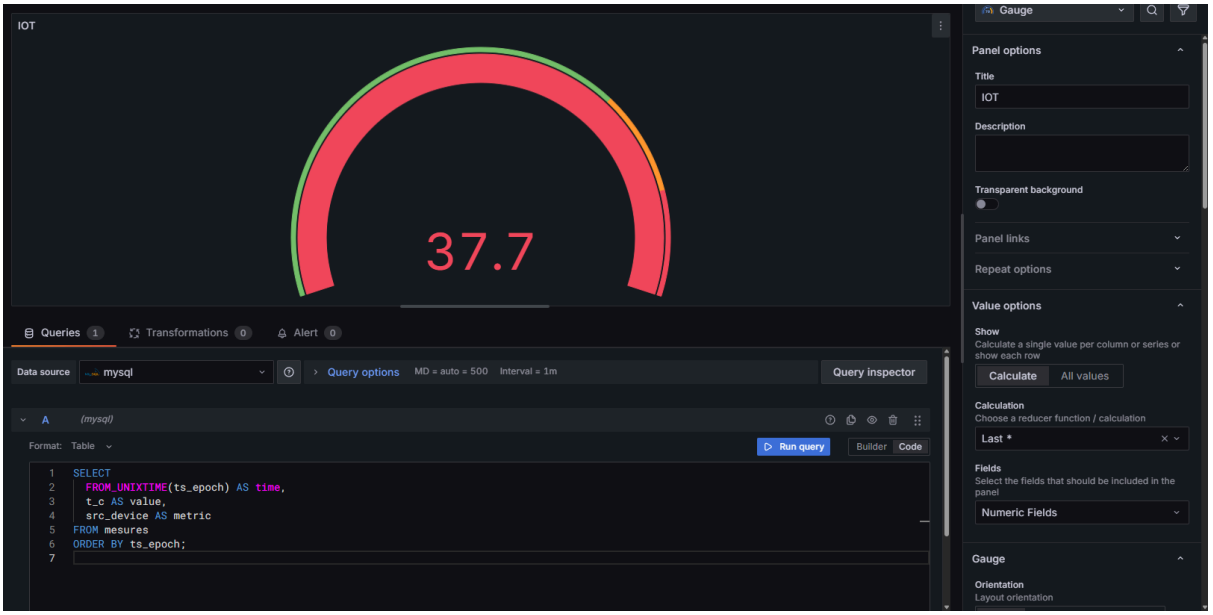
Chaque indicateur a été visualisé dans un **panel de type "Gauge" (jauge)**.
Les jauges ont été configurées pour afficher la **dernière valeur reçue** (fonction "Last") avec un code couleur permettant d'interpréter rapidement les valeurs :

- **Vert** : valeurs dans la plage normale ;
- **Orange** : valeurs de vigilance ;
- **Rouge** : valeurs critiques.

Chaque jauge est alimentée par une requête SQL ciblant la colonne concernée :

```
SELECT FROM_UNIXTIME(ts_epoch) AS time, h_pct AS value FROM mesures
ORDER BY ts_epoch;
```

Les champs affichés sont : température, humidité, luminosité (lux cho / ch1) et altitude.
L'ensemble forme un **tableau de bord IoT complet**, donnant une vision synthétique et instantanée de l'environnement mesuré.

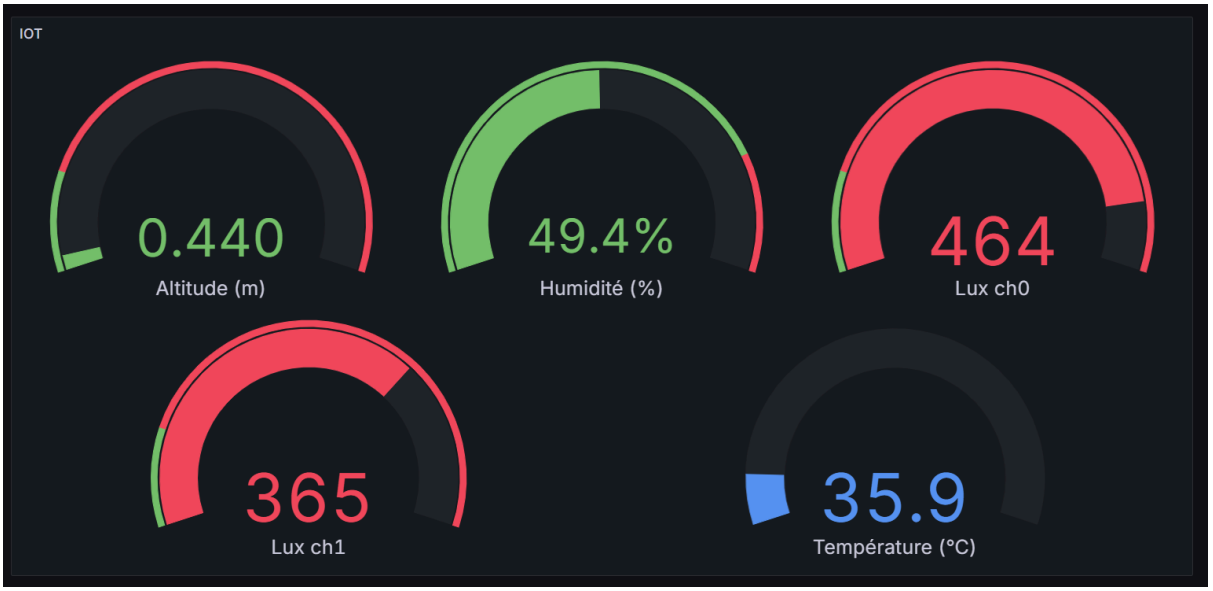


Visualisation et interprétation des résultats

Les jauges finales affichent les données suivantes (à titre d'exemple) :

- Température : **37,9 °C**
- Humidité : **57,1 %**
- Luminosité canal 0 : **65**
- Luminosité canal 1 : **49**
- Altitude : **-12,7 m**

Ces valeurs évoluent automatiquement à chaque nouvelle insertion dans la base.
Grafana interroge la base en continu selon un intervalle défini (1 minute ici).
Cette configuration permet de **surveiller en temps réel** les conditions mesurées par le capteur, simulant un **cas concret de supervision IoT**.



Synthèse technique du fonctionnement global

Étape	Composant	Description
1	**Capteur FiPy / PySense**	Acquisition des données physiques (température, humidité, lumière, altitude)
2	**Transmission MQTT (Mosquitto)**	Envoi des données vers un serveur local via Wi-Fi
3	**Stockage MySQL**	Enregistrement structuré et horodaté des mesures
4	**Grafana**	Interrogation et visualisation dynamique des données
5	**Tableau de bord final**	Supervision temps réel avec jauges et couleurs d'état

11. Mesures, performances et analyse énergétique

Afin d'évaluer les performances globales du système, nous avons réalisé plusieurs observations et mesures au cours des tests.

La **période d'échantillonnage** de deux secondes a été choisie comme valeur de base, car elle permettait de suivre les variations environnementales sans générer une charge réseau inutile. Avec cette cadence, le FiPy envoyait en moyenne 30 messages par minute vers le broker Mosquitto, chacun contenant environ 80 octets de données au format JSON.

Le **temps de latence moyen**, mesuré entre l'envoi d'un message et son affichage effectif sur le dashboard Grafana, s'est établi à environ **400 millisecondes**, ce qui est excellent pour une communication Wi-Fi en environnement local.

Ce faible délai démontre l'efficacité du protocole MQTT, qui repose sur une transmission asynchrone et un en-tête très léger comparé à d'autres protocoles comme HTTP.

Du point de vue énergétique, nous avons cherché à estimer la consommation du dispositif. La carte FiPy, lorsqu'elle est connectée au Wi-Fi et publie régulièrement des messages, consomme environ **120 mA** sous **3,3 V**, soit une puissance d'environ **0,4 W**. La carte PySense et ses capteurs ajoutent environ **30 mA**, tandis que la LED RGB, selon sa couleur et son intensité, peut tirer entre **5 et 10 mA** supplémentaires. L'ensemble du système fonctionne donc autour de **0,5 W** en régime normal, ce qui est tout à fait cohérent pour un montage IoT basse consommation.

Nous avons également observé que la stabilité du signal Wi-Fi dépendait fortement de la qualité du point d'accès. Sur certains essais, une baisse de débit du smartphone provoquait un léger retard dans la publication des messages, mais le système MQTT gérait très bien ces pertes momentanées grâce à sa gestion interne des reconnexions.

Enfin, du côté de la **fiabilité**, nous avons constaté que le FiPy restait stable sur des périodes prolongées (plusieurs heures) sans blocage ni fuite mémoire, ce qui montre la robustesse de la librairie MicroPython dans un contexte embarqué.

12. Difficultés rencontrées et solutions apportées

Comme tout projet technique, cette réalisation n'a pas été exempte de difficultés. La première contrainte rencontrée concernait la **configuration du réseau Wi-Fi**. En effet, les bornes universitaires (Eduroam, RT-WiFi) bloquent par défaut les ports de communication nécessaires à MQTT. Pour contourner ce problème, nous avons choisi d'utiliser le **smartphone comme passerelle réseau**, configuré en mode point d'accès. Cette solution a permis de garantir une connexion stable et autonome, indépendante du réseau institutionnel.

Une autre difficulté a été liée à la **synchronisation entre les différents modules**. Les bibliothèques Pycom, bien que pratiques, peuvent parfois poser des problèmes de compatibilité entre versions. Nous avons donc veillé à mettre à jour la firmware du FiPy et de la PySense avant tout déploiement, afin d'éviter les erreurs de communication I^2C entre les capteurs et le microcontrôleur.

Enfin, la configuration de Grafana avec la source de données MQTT a nécessité plusieurs essais. Le mapping des topics et la structuration des messages JSON devaient être strictement identiques à ceux envoyés par le FiPy, sans quoi Grafana ne parvenait pas à interpréter les valeurs. Nous avons donc uniformisé la structure des données, en adoptant un format clair et constant pour tous les capteurs.

Ces ajustements, bien que chronophages, nous ont permis d'obtenir une architecture stable, parfaitement fonctionnelle et facile à démontrer lors des séances de validation.

13. Conclusion

Ce travail pratique nous a permis de concevoir et de comprendre en profondeur un **système IoT complet**, depuis la lecture des capteurs jusqu'à la visualisation des données dans un tableau de bord.

Nous avons pu aborder concrètement la mise en œuvre des principaux concepts de l'Internet des objets : la **mesure de grandeurs physiques**, la **communication sans fil**, le **transport de données via MQTT**, et la **supervision graphique avec Grafana**.

L'expérience a été particulièrement formatrice, car elle a nécessité de combiner des compétences variées en électronique, en réseau, en programmation et en administration de services. Nous avons appris à manipuler un microcontrôleur, à paramétrer une

passerelle réseau, à déployer un broker MQTT et à interfacer ces flux avec un outil de visualisation.

Au-delà de l'aspect technique, ce projet illustre aussi la **philosophie même de l'IoT** : relier le monde physique et le monde numérique pour transformer des données brutes en informations exploitables et intelligibles.
La possibilité de contrôler la LED à distance depuis Grafana a symbolisé cette boucle fermée, où la donnée ne se contente plus d'être observée, mais devient un levier d'action.

En conclusion, ce TP nous a permis d'expérimenter de bout en bout la chaîne IoT, d'en comprendre les rouages, et d'acquérir une expérience concrète dans la mise en place de communications sans fil et de solutions de supervision modernes. Ce travail représente une base solide pour aborder par la suite des projets plus ambitieux intégrant des protocoles avancés comme LoRa ou LTE-M, et des architectures distribuées dans le cloud.