Groupe 2B12 Date : 01/12/2024

SAE S3.01 – Développement d'Application

Compétence 3 – Système

Compte Rendu

Architecture Système

Sommaire

Introduction

Contexte

Ce projet consiste à développer un système IoT qui récupère et gère les données provenant de capteurs et de panneaux solaires. Les données sont alors stockées en utilisant le protocole MQTT pour leur transmission en temps réel vers des fichiers CSV. Cela permet une surveillance continue via une interface utilisateur dédiée.

Ce projet utilise Python pour mettre en place un système IoT capable de recevoir des données via le protocole MQTT, de les traiter et de les enregistrer dans des fichiers CSV pour un usage ultérieur dans une interface utilisateur. Ci-dessous, une explication détaillée du fonctionnement, des fonctionnalités et des blocs de code.

Objectif

Développer une architecture adaptable pour collecter, traiter et utiliser les données IoT collectés, pour permettre leur exploitation durant leur suivi en temps réel et leur analyse.

Problématique

Comment concevoir un système IoT facilement configurable, capable de gérer des flux de données en temps réel, garantissant leur stockage et leur utilisation pour différents usages ?

Objectifs du projet

Réception des données

- Les données proviennent de deux sources :
  - Capteurs de salle AM107 : température, humidité, taux de CO2.
  - Panneaux solaires Solaredge : puissance et heure de mise à jour.
- Ces données sont transmises via un serveur MQTT.

Traitement des données

- Le programme identifie le type de données en fonction du sujet (topic).
- Il affiche les données dans la console pour un suivi en temps réel.
- Les données sont sauvegardées dans des fichiers CSV pour une consultation ultérieure ou pour la partie IHM.

Modularité

- La configuration est paramétrable via un fichier configuration.ini.
- Les fichiers CSV servent de passerelle vers d'autres modules ou outils (ex. IHM).

Explication des composants

Import des bibliothèques

Le programme utilise plusieurs bibliothèques Python pour gérer les différentes fonctionnalités :

- `json` : Pour interpréter les messages JSON reçus via MQTT.

- `configparser` : Pour lire les paramètres dynamiques depuis le fichier `configuration.ini`.
- `csv` : Pour créer et gérer les fichiers CSV contenant les données reçues.
- `paho.mqtt.client` : Pour gérer la communication MQTT entre le programme et le serveur.
- `logging` : Pour consigner les erreurs et les informations.

Fichier de configuration

Le fichier *configuration.ini* est utilisé pour configurer le programme sans modifier directement le code. Cela permet de rendre l'application flexible et facile à adapter à différents contextes.

```
[MQTT]
broker                  = mqtt.iut-blagnac.fr  # Adresse du serveur MQTT
port                    = 1883                 # Port utilisé pour la connexion
topic                   = AM107/by-room/       # Sujet des messages MQTT

[DATA_SOURCE]
salle_ou_panneau_solaire = data                # Indique le type de donnée à recevoir

[DATA_TYPE]

[SEUILS]
temperature             = 10
co2                     = 100
humidity                = 20
tvoc                    = 10
infrared_and_visible    = 10

[PARAMS]
frenquence_s            = 60                 # Fréquence de collecte des données

[SUJET]
salle1                   = B002
salle2                   = B106
salle3                   = B202
salle4                   = C004
```

Les fonctions principales

1. `on_connect(client, userdata, flags, reason_code, properties)`

Cette fonction est appelée automatiquement lorsque le programme se connecte au serveur MQTT.

- **Rôle** :
  - Vérifier si la connexion est réussie.
  - S'abonner au sujet (`topic`) défini dans le fichier de configuration.

Code : python

```
def on_connect(client, userdata, flags, reason_code,
properties):
    print(f"Connecté au broker MQTT avec le code de résultat :
{reason_code}")
    client.subscribe(topic)
```

- **Sortie** : Si la connexion est réussie, le programme s'abonne au sujet, ce qui permet de recevoir des messages.

2. `on_message(client, userdata, message)`

Cette fonction est déclenchée chaque fois qu'un message est reçu.

- **Rôle** :
    - Interpréter le message JSON reçu.
    - Vérifier de quel type de données il s'agit (capteurs ou panneaux solaires).
    - Afficher les données dans la console.
    - Sauvegarder les données dans le fichier CSV approprié.

Code python :

```
def on_message(client, userdata, message):
    try:
        data = json.loads(message.payload)
        if message.topic.startswith("AM107/by-room"):
            print("\nDONNÉES CAPTEUR AM107")
```

```
            print("====================")
            print("Salle :", data[1]['room'])
            print("Température :", data[0]['temperature'])
            print("Humidité :", data[0]['humidity'])
            print("Taux de CO2 :", data[0]['co2'])

            # Enregistrement dans un fichier CSV
            with open('dataCapteur.csv', 'w', newline='') as
file:
                writer = csv.writer(file, delimiter=';')
                writer.writerow(["Salle", "Température",
"Humidité", "Taux de CO2"])
                for sensor_data in data:
                    writer.writerow([sensor_data['room'],
sensor_data['temperature'], sensor_data['humidity'],
sensor_data['co2']])

        elif message.topic.startswith("solaredge/blagnac"):
            print("\nDONNÉES PANNEAUX SOLAIRES")
            print("========================")
            print("Dernière mise à jour :",
data['lastUpdateTime'])
            print("Power (puissance) :",
data['currentPower']['power'])

            # Enregistrement dans un fichier CSV
            with open('dataSolar.csv', 'w', newline='') as
file:
                writer = csv.writer(file, delimiter=';')
                writer.writerow(["Date dernière mise à jour",
"Puissance"])
                writer.writerow([data['lastUpdateTime'],
data['currentPower']['power']])

    except (json.JSONDecodeError, KeyError) as e:
        logging.error("Erreur dans les données reçues : %s",
e)
```

- **Explication** :

Les données des capteurs sont enregistrées dans `dataCapteur.csv` :

```
Salle;Température;Humidité;Taux de CO2
A101;22.5;45;400
B203;21.0;50;420
```

Les données des panneaux solaires sont enregistrées dans `dataSolar.csv` :

```
Date dernière mise à jour;Puissance
2024-12-08T10:00:00Z;500
```

Connexion au broker et boucle principale

Le programme utilise un objet `mqtt.Client()` pour se connecter au broker et activer les fonctions de callback définies précédemment.

Code python :

```
mqttc = mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.connect(serveurMQTT, port=port, keepalive=60)
mqttc.loop_forever()
```

- **Explication** :
    - La méthode `connect()` établit une connexion avec le serveur MQTT.
    - La méthode `loop_forever()` maintient la connexion et écoute les messages en continu.

---

3. Points d'amélioration

1. **Gestion des seuils** :
    - Ajouter un module pour comparer les données reçues à des seuils prédéfinis. En cas de dépassement, une alerte pourrait être générée.

2. **Agrégation des données** :
    - Actuellement, toutes les données reçues sont sauvegardées. Il serait utile d'ajouter une option pour filtrer les données (par ex. garder uniquement la moyenne horaire).

3. **Interface utilisateur (IHM)** :
    - Intégrer une interface graphique pour visualiser les données en temps réel à partir des fichiers CSV.

4. Conclusion

Le système présenté ici est un prototype fonctionnel pour gérer des données IoT avec MQTT. Il est conçu pour être extensible et adaptable. Les données collectées sont centralisées dans des fichiers CSV, ce qui facilite leur exploitation par d'autres modules, comme une IHM ou des outils d'analyse.