SAÉ24: Collecte

Groupe 13

Louis DESVERNOIS, Alexis SCHOENN, Philippe DUBOIS

24 juin 2022

Table des matières

Table des codes

1 Connexion à la base de données

Nous avons décidé d'utiliser la deuxième option du sujet, c'est à dire un script Python ajoutant directement les valeurs récupérées [1]. En Python il existe plusieurs moyens d'accéder à une base de données, ici nous avons utilisé `mysqlclient`, la même librairie utilisée par Django.

```
try:
    db=_mysql.connect("mysql.rt13.lab","root","admin", "temp")
except OperationalError:
    db=_mysql.connect("mysql.rt13.lab","root","admin")
    db.query("CREATE DATABASE temp")
    db.query("USE temp")
```

Code 1 – Connexion BDD

Voici, en Code 1, l'extrait de notre script qui permet de se connecter à notre serveur MySQL. Le "try, except" permet ici de créer la base de données "temp" si elle n'existe pas. Après cela, nous utilisons des requêtes SQL pour créer les deux tables de notre base de données, nous nous attarderons sur cela dans la partie web.

2 Connexion au brocker MQTT

Pour nous connecter au Brocker MQTT, nous allons utiliser le paquet Python `paho-mqtt` que nous pouvons installer avec `pip install`. Une fois le paquet installé, nous pouvons créer un simple script pour nous abonner à notre topic ainsi que d'imprimer les messages dans le terminal.

```
import paho.mqtt.client as mqtt
import random

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("IUT/Colmar/SAE24/Maison1")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client(client_id=f"client-grp13-{random.randint(1, 99999)}")
client.on_connect = on_connect
client.on_message = on_message

client.connect("test.mosquitto.org", port=1883, keepalive=60)

client.loop_forever()
```

Code 2 – Simple script imprimant les messages MQTT

On remarque dans le code 2 que l'on crée deux fonctions `on_connect` et `on_message`, la première est exécutée lors de la connexion au brocker tandis que la deuxième est exécutée lors de la réception d'un message MQTT. Nous pouvons donc utiliser ces fonctions pour nos analyses et nos opérations SQL. Nous définissons également un id unique grâce à `random.randint`.

1. Nous verrons dans le rapport Web/BDD comment nous avons déployé notre serveur MySQL

3 Écriture d'un parser pour les données MQTT

Les données envoyées par les capteurs sont présentées sous un format *csv* (Comma-separated values) comme en exemple ci-dessous (Code 3).

```
Id=B8A5F3569EFF,piece=sejour,date=22/06/2022,time=21:23:53,temp=11.56
```

Code 3 – Exemple de message brut

Il est donc facile de séparer ces valeurs avec Python grâce à la méthode `split(',')`. Nous remarquons que chaque valeur est précédée de `=`. Comme nous n'avons pas besoin de tout ce qui est avant le signe égal, nous pouvons utiliser la même méthode que précédemment pour ne garder uniquement les valeurs utiles.

```
mac_addr= payload[0].split("=")[1]
piece= payload[1].split("=")[1]
dt= datetime.strptime(payload[2].split("=")[1] + " " + payload[3].split("=")[1],
                      '%d/%m/%Y %H:%M:%S')
temp= payload[4].split("=")[1]
```

Code 4 – Séparation des valeurs

En Code 4 nous utilisons également la méthode `strptime(str, format)` de l'objet `datetime`, qui nous permettra, en spécifiant le format de la date d'origine, d'importer la date en format ISO 8601 dans la base de données.

4 Envoi des requêtes SQL

Une fois les valeurs récupérées et séparés, nous pouvons nous atteler aux requêtes SQL. Cela commence à sortir du contexte de cette partie.

```
if mac_addr not in dico_macaddr.keys():
    try:
        db.query(f"INSERT INTO sensors (macaddr, piece) VALUES ('{mac_addr}',
        ↪  '{piece}')")
    except Exception:
        pass
    db.query(f"SELECT id FROM sensors WHERE macaddr='{mac_addr}'")
    id = int(db.store_result().fetch_row()[0][0])
    dico_macaddr[mac_addr] = id
```

Code 5 – Dictionnaire des adresses MAC

En code 5 nous utilisons un dictionnaire pour stocker les id correspondants aux adresses MAC, cela nous permet de limiter les requêtes à la base de données car nous avons besoin de cet id pour ajouter la température dans bonne table.

```
sql_data = f"INSERT INTO sensors_data (sensor_id, datetime, temp) VALUES
↪  ({dico_macaddr[mac_addr]}, '{dt.strftime('%Y-%m-%d %H:%M:%S')}', {temp})"
print(sql_data)

reachable = True
try:
    db.query(sql_data)
    reachable = True
except Exception:
    reachable = False
    bak = open("backup.sql", "a")
    bak.write(f"{sql_data}\n")
    bak.close()

if exists("backup.sql") and reachable:
    with open('backup.sql') as f:
        for line in f:
            db.query(line)
            remove('backup.sql')
```

Code 6 – Envoi des données dans la BDD

En code 6 nous utilisons des requêtes SQL pour simplement envoyer les données dans la base de données. Si la requête échoue après une déconnexion soudaine du serveur MySQL, les requêtes sont à la place écrite dans le fichier `backup.sql` pour être exécutées une fois que la connexion est retrouvée [2].

2. Ce code ne fonctionne pas dans certaines conditions, car dans certains cas, la librairie Python utilisée ne lance pas d'erreur, mais lance une boucle infinie qui bloque complètement l'exécution du script