SAE 3.01 Partie IOT

(Groupe 12)

Documentation Python

Khalil Ahmad - Sellou Rayan - Gibert Yannis (Groupe 2B)

Sommaire :

Tutoriel d'installation

Installer Python

Pour vérifier si votre système dispose de Python :

- ouvrir un terminal
- lancer la commande **Python –version**

Si la version de Python apparaît, vous pouvez sauter l'étape d'installation de Python.

Si Python n'est pas installé, vous pouvez le télécharger depuis **https://www.python.org/downloads/.**

Pour lancer le script Python, assurez-vous d'avoir les bibliothèques requises installées. Voici les bibliothèques nécessaires :

paho.mqtt.client : bibliothèque MQTT pour Python.

json : module Python pour travailler avec JSON.

configparser : module Python pour lire les fichiers de configuration.

os : module Python pour des fonctionnalités liées au système d'exploitation.

time (sous Windows uniquement) : module Python pour le temps.

datetime : module Python pour manipuler les dates et heures.

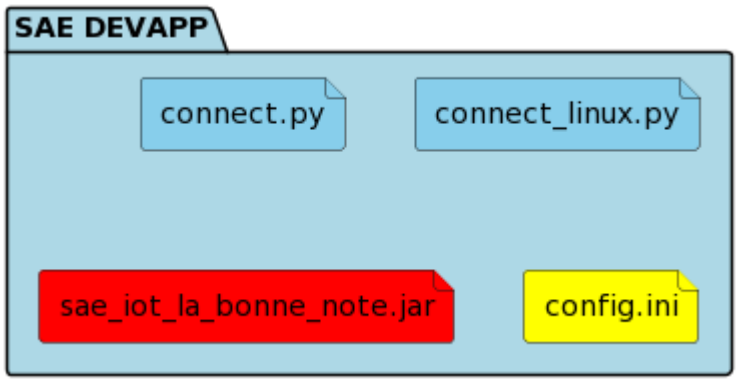
Pour installer les bibliothèques Python, ouvrez une invite de commande ou un terminal et saisissez les commandes suivantes :

- **pip install paho-mqtt**
- **pip install jsonlib-python3**
- **pip install configparser**
- **pip install datetime**

Utiliser le script

Pour lancer le script :

- Rendez-vous sur le lien du dossier de l'application finale qui est structuré comme ci-dessous :



(En bleu les script python, en rouge l'exécutable de l'application qui ne sera pas utile ici et en jaune le fichier de configuration)

- Télécharger les **fichiers Python** ainsi que le fichier de configuration **config.ini** (ces fichiers sont aussi déposés en même temps que cette documentation sur WebEtud si le GitHub n'est pas accessible car privée pendant la SAE)

- Après avoir téléchargé les fichiers, vérifier que le fichier de configuration (**config.ini**) et le script Python (**connect.py** ou **connect_linux.py** selon le système d'exploitation) sont au même endroit dans l'arborescence. Les paramètres par défaut du fichier de configuration permettent la connexion au serveur MQTT donc inutile d'y toucher.

- Ouvrez une invite de commande et exécutez la commande suivante selon votre système d'exploitation :

  Windows :

  - **Python3 connect.py**

  Linux :

  - **Python3 connect_linux.py**

Structure des fichiers

Fichiers Python

- **connect_linux.py** : version Linux du script.

- **connect.py** : version Windows du script.

Il y a deux fichiers Python, un pour la version Windows et un pour la version Linux.

Fichier de configuration

- **config.ini** : fichier de configuration qui contient tous les paramètres (serveurs, seuils d'alertes..), modifiable à travers l'application java

Fichiers de données

- **alertes.json :** un fichier Json stockant tout l'historique des alertes (le script le crée s'il n'existe pas déjà)

- **logs.json :** un fichier Json stockant tout l'historique des données (le script le crée s'il n'existe pas déjà)

- **donnees.json :** un fichier Json stockant juste les données récupérées en temps réel (fichier qui est supprimé puis crée à nouveau à chaque lancement)

(Les noms de fichiers peuvent êtres changés à travers l'application.)

Justification du choix pour les fichiers de données

Nous avons examiné plusieurs critères pour déterminer lequel serait le plus adapté à nos besoins spécifiques. Le tableau ci-dessus met en lumière les principaux avantages et inconvénients de chaque format. Après une évaluation approfondie, nous avons conclu que le format JSON offre plusieurs avantages significatifs pour notre cas d'utilisation. Sa facilité de manipulation, sa capacité à stocker des structures complexes et sa lisibilité ont été des facteurs décisifs dans notre choix du format JSON pour le stockage de nos données.

Critère	JSON	CSV
Facilité de lecture	Facile à lire pour les humains grâce à sa structure hiérarchique	Peut être difficile à lire pour les humains en raison de sa structure tabulaire
Facilité de manipulation	Manipulation aisée grâce à sa structure d'objets et d'arrays	Peut nécessiter un traitement spécifique pour manipuler les données tabulaires
Taille du fichier	Peut être plus volumineux en raison de la nature textuelle et répétitive des données	Généralement plus compact en raison de la simplicité des données stockées
Compatibilité	Bien pris en charge par de nombreuses langages et plateformes grâce à sa popularité	Compatibilité étendue avec de nombreux logiciels et outils
Structure des données	Peut stocker des structures complexes et des types de données variés	Idéal pour des données tabulaires simples, peut nécessiter une transformation pour des structures complexes
Lisibilité des données	Les données sont souvent plus lisibles pour les humains en raison de leur structuration	Peut être moins lisible en raison des séparateurs et des éventuelles valeurs null

Explication du code

Pour le bon lancement du script, il faut que le fichier de configuration soit situé au même endroit dans l'arborescence que les scripts Python, sans

cela le script renverra une erreur. Nous allons donc commencer par une présentation du fichier de configuration et comment celui-ci est utilisé dans le code puis nous verrons les autres méthodes du code.

Fichier de configuration

Le fichier de configuration (**config.ini**) comporte plusieurs sections :

```
[MQTT]
broker=chirpstack.iut-blagnac.fr
port=1883
topic=AM107/by-room/#
[CONFIG]
fichier_alerte=alerte
fichier_donnees=donnees
fichier_logs=logs
choix_donnees=temperature,humidity,activity,co2
typeTemps=seconde(s)
frequence_affichage=0.0
[ALERT]
son_Alertes=98.0
seuil_Temperature=10.0
seuil_Humidity=50.0
seuil_CO2=0.0
seuil_Activity=2.0
```

MQTT : comporte le broker, le port et le topic sur lequel se mettre en écoute

CONFIG : se compose des noms des fichiers, des données choisies, de la fréquence et de l'unité de cette fréquence (utilisé par l'application)

ALERT : le niveau sonore des alertes (en % dans l'application java seulement) les seuils d'alertes pour chaque donnée

```
# Chemin du fichier de configuration
config_file_path = r'config.ini'

# Obtenir le répertoire du fichier de configuration
config_dir = os.path.dirname(config_file_path)

print("Répertoire de travail actuel :", os.getcwd())

# Lire les paramètres de configuration
config = configparser.ConfigParser()
found = config.read(config_file_path)

# Récupèrer la fréquence d'affichage à partir des paramètres de configuration MQTT
frequence_affichage = config.getfloat('CONFIG', 'frequence_affichage')

print("Fichiers de configuration trouvés :", found)
print("Sections trouvées :", config.sections())

broker = config.get('MQTT', 'broker')
port = config.getint('MQTT', 'port')
topic = config.get('MQTT', 'topic')
choix_donnees = config.get('CONFIG', 'choix_donnees').split(',')  # Récupère les valeurs à afficher
print("Vous avez choisi d'afficher les données suivante : ", choix_donnees)

# Dictionnaire pour stocker les valeurs
values_by_room = {}
historique_par_salle = {}

# Nom du fichier pour écrire les logs
fichier_logs = config.get('CONFIG', 'fichier_logs') + '.json'

# Nom du fichier pour écrire les données
fichier_donnees = config.get('CONFIG', 'fichier_donnees') + '.json'

# Nom du fichier pour écrire les alertes
fichier_alertes = config.get('CONFIG', 'fichier_alerte') + '.json'
```

Toutes ces données sont récupérées et utilisées dans le script Python comme ci-dessus.

Les différentes méthodes

Le script python se compose de plusieurs méthodes :

calculer_moyenne : permet de calculer la moyenne pour les 10 dernières valeurs, prend en paramètre le tableau de l'historique des valeurs

```
50  # Fonction pour calculer la moyenne des 10 dernières valeurs
51  def calculer_moyenne(historique):
52      return sum(historique[-10:]) / min(len(historique), 10)
```

ecrire : permet d'écrire les logs / données dans le fichier, prend en paramètre un booléen pour savoir s'il s'agit d'une écriture de log ou de données, le nom du fichier dans lequel écrire, l'identifiant de la room (salle), cet identifiant permettra d'ajouter les données au même endroit dans le fichier Json au lieu de les ajouter n'importe où et enfin la data à écrire en elle même

```python
def ecrire(ecrire_log, nom_fichier, room, data):
    try:
        # Ouverture du fichier JSON en mode Lecture / Ecriture, le créer s'il n'existe pas avec les droits 644
        # (User rw, Group rx, Others rx)
        fichier = os.open(nom_fichier, os.O_RDWR | os.O_CREAT, 0o644)

        # Écriture d'un objet JSON vide si le fichier est nouveau
        if os.path.getsize(fichier) == 0:
            os.write(fichier, json.dumps({}).encode())

        # Lecture des données JSON existantes
        os.lseek(fichier, 0, os.SEEK_SET)
        contenu = os.read(fichier, os.path.getsize(fichier)).decode()
        if contenu:
            donnees = json.loads(contenu)
        else:
            donnees = {}
        # Mise à jour des données avec les nouvelles informations
        if room not in donnees:
            donnees[room] = []
        # Ajout des nouvelles données
        donnees[room].append(data)
        # Écriture des données mises à jour dans le fichier sans effacer le contenu existant
        os.lseek(fichier, 0, os.SEEK_SET)
        os.write(fichier, json.dumps(donnees, indent=4).encode())
        # Fermeture du descripteur de fichier
        os.close(fichier)
        # Mise en "sommeil" avant la prochaine écriture
        if(ecrire_log):
            time.sleep(frequence_affichage)
    except Exception as e:
        print(f"Erreur lors de l'écriture dans le fichier données : {e}")
```

ecrire_alerte : permet d'écrire les alertes dans le fichier, prend en paramètre l'identifiant de la room (salle) ainsi que l'alerte à écrire

```python
def ecrire_alerte(room, alerte):
    try:
        # Ouverture du fichier JSON en mode Lecture / Ecriture, le créer s'il n'existe pas avec les droits
        # 644(User rw, Group rx, Others rx)
        fic_alertes = os.open(fichier_alertes, os.O_RDWR | os.O_CREAT, 0o644)

        # Écriture d'un objet JSON vide si le fichier est nouveau
        if os.path.getsize(fichier_alertes) == 0:
            os.write(fic_alertes, json.dumps({}).encode())

        # Lecture des données JSON existantes
        os.lseek(fic_alertes, 0, os.SEEK_SET)
        contenu = os.read(fic_alertes, os.path.getsize(fichier_alertes)).decode()
        if contenu:
            donnees = json.loads(contenu)
        else:
            donnees = {}
        # Mise à jour des données avec les nouvelles informations
        if room not in donnees:
            donnees[room] = []
        donnees[room].append(alerte)

        os.lseek(fic_alertes, 0, os.SEEK_SET)
        os.write(fic_alertes, json.dumps(donnees, indent=4).encode())
        # Fermeture du descripteur de fichier
        os.close(fic_alertes)
    except Exception as e:
        print(f"Erreur lors de l'écriture dans le fichier d'alerte : {e}")
```

on_connect : méthode qui est appelé lors de la connection au serveur MQTT, on se met en “écoute” sur le topic dans cette méthode

```python
# Fonction appelée lors de la connexion au broker
def on_connect(client, userdata, flags, rc):
    print("Connecté avec le code résultat " + str(rc))
    client.subscribe(topic)
```

on_message : méthode permettant de récupérer et traiter les nouveaux messages arrivant sur le bus MQTT

```python
# Fonction appelée à la réception d'un message
def on_message(client, userdata, msg):
    try:
        print(f"Message reçu sur le topic {msg.topic}")
        payload = json.loads(msg.payload)
        print("Payload brut:", payload)

        # Traitement des données
        sensor_data = payload[0]
        device_info = payload[1]

        room = device_info['room']
        if room not in historique_par_salle:
            historique_par_salle[room] = {key: [] for key in sensor_data.keys()}

        maintenant = datetime.now()
        date_heure = maintenant.strftime("%d-%m-%Y %H:%M:%S")

        envoyer_alerte = False

        alerte = {
            "date": date_heure,
            "donnees": {}
            }
        alerte_texte = ""

        salle_donnees = {
            "date": date_heure,
            "donnees": {}
            }

        donnees_logs = {
            "date": date_heure,
            "donnees": {}
            }
```

Récupère le message, vérifie si la salle concernée est déjà dans l'historique des salles (pour le calcul de la moyenne), l'ajoute si c'est pas le cas, on récupère ensuite la date actuelle et on initialise la structure JSON de la nouvelle donnée à écrire.

```
158         for key, value in sensor_data.items():
159             if key.lower() in choix_donnees:  # Vérifie si la clé est dans les valeurs à afficher
160                 historique_par_salle[room][key].append(value)
161                 # S'assurer que la liste ne contient que les 10 dernières valeurs
162                 if len(historique_par_salle[room][key]) > 10:
163                     historique_par_salle[room][key].pop(0)  # Supprime la valeur la plus ancienne
164                 moyenne = calculer_moyenne(historique_par_salle[room][key])
165                 salle_donnees["donnees"][key] = {
166                     "valeur": value,
167                     "moyenne": moyenne
168                 }
169                 donnees_logs["donnees"][key] = value
170                 # Vérification des seuils pour déclencher une alerte
171                 seuil_key = f"seuil_{key.lower()}"
172                 if config.has_option('ALERT', seuil_key):
173                     seuil_max = config.getfloat('ALERT', seuil_key)
174                     if value > seuil_max:
175                         alerte["donnees"][key] = {
176                             "valeur": value,
177                             "seuil_max": seuil_max
178                             }
179                         alerte_texte += f"- {key} a dépassé le seuil maximum de {seuil_max}, valeur actuelle : {value}\n"
180                         envoyer_alerte = True
181 
182         print(salle_donnees)
183         if(envoyer_alerte):
184             print ("Alertes : \n" + alerte_texte)
185             ecrire_alerte(room, alerte)  # S'il y a des alertes, les écrire dans le fichier
186         ecrire(False, fichier_donnees, room, salle_donnees)
187         ecrire(True, fichier_logs, room, salle_donnees)
188 
189         # Mise en "sommeil" avant la prochaine écriture
190         if(frequence_affichage > 0):
191             time.sleep(frequence_affichage)
192 
193     except Exception as e:
194         print(f"Erreur lors du traitement du message: {e}")
```

Pour chaque donnée reçue, vérifie s'il s'agit d'une donnée souhaitée si oui l'ajoute dans l'historique par salle (pour le calcul de la moyenne) puis met à jour la moyenne. Les nouvelles données sont ensuite écrites dans le bon fichier JSON. Après chaque écriture, si la fréquence d'affichage choisie est > 0, le programme se met en pause le temps de la fréquence puis reprend. Le programme ne récupère pas les données durant ce temps. L'écriture du message est annulée en cas d'erreur.

handler (uniquement sur la version linux) : permet de gérer le signal d'alarme SIGALARM (aucune action spécifique à faire ici)

```
56  # Gestion du système d'alarme
57  def handler(signum, frame):
58      # Cette fonction sera appelée lorsque le signal SIGALRM sera déclenché
59      pass
```

Pour se connecter au serveur MQTT et associer les évènements aux méthodes :

```
194 # Création et configuration du client MQTT
195 client = mqtt.Client()
196 client.on_connect = on_connect
197 client.on_message = on_message
198
199 # Connexion au broker
200 try:
201     client.connect(broker, port, 60)
202 except Exception as e:
203     print(f"Echec de la connexion, code d'erreur : {e}")
204     exit(1)
```

Essaye de se connecter au serveur MQTT, l'utilisateur est informé du résultat de la connexion (voir la méthode **on_connect** en cas de succès), en cas d'échec, le programme est arrêté avec une sortie 1 (code d'erreur standard.

Si la connexion au serveur MQTT est établie, le programme tournera en boucle pour recevoir les messages :

```
206 # Boucle de traitement des messages
207 client.loop_forever()
```

Cas de tests

Cas normal

Lors du lancement du programme :

```
PS C:\Users\1\Documents\GitHub\sae3-solo> & C:/Users/1/AppData/Local/Microsoft/WindowsApps/python3.11
ocuments/GitHub/sae3-solo/connect.py
Répertoire de travail actuel : C:\Users\1\Documents\GitHub\sae3-solo
Fichiers de configuration trouvés : ['config.ini']
Sections trouvées : ['MQTT', 'CONFIG', 'ALERT']
Vous avez choisi d'afficher les données suivante :  ['temperature', 'humidity', 'activity', 'co2']
Connecté avec le code résultat 0
```

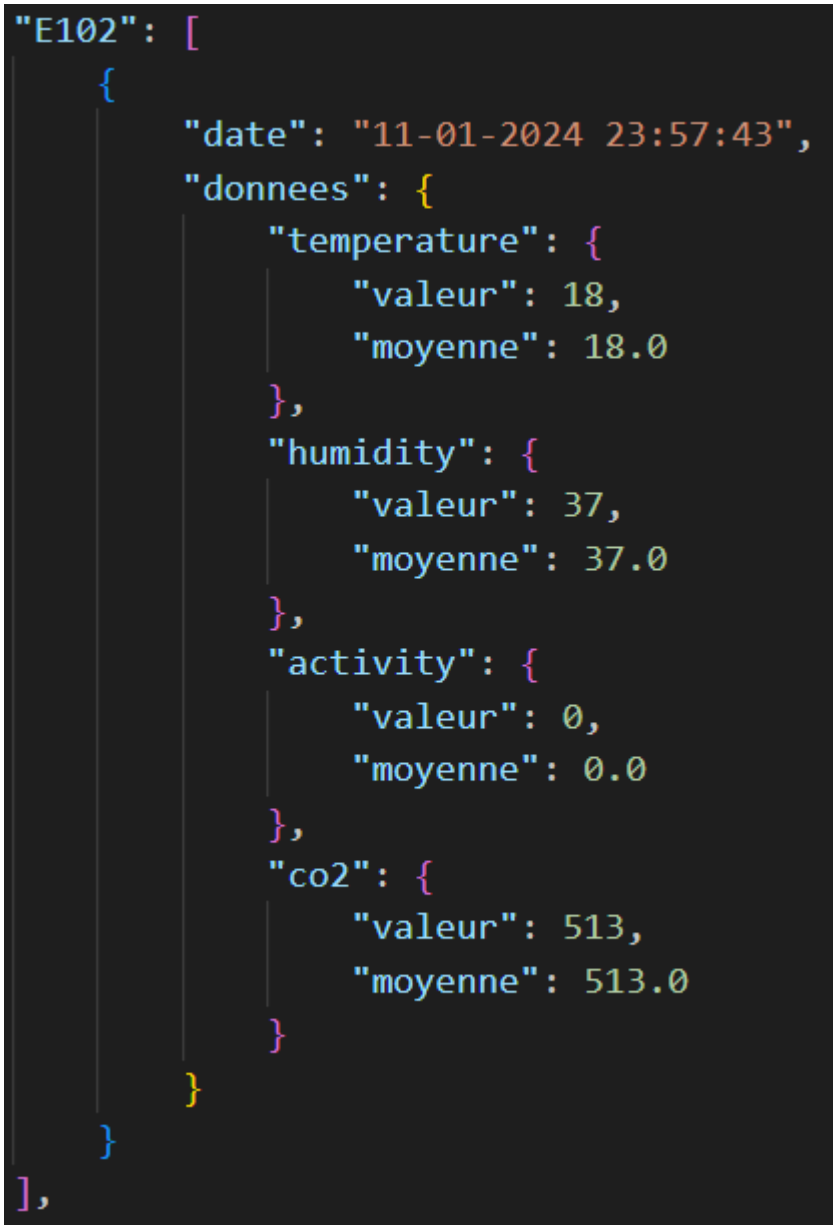
Si aucune erreur n'est rencontrée lors de la récupération de la configuration ou lors de la connexion au serveur MQTT, le programme affichera le répertoire de travail actuel, le fichier de configuration trouvé avec ses différentes sections ainsi que le choix des données à récupérer.

Lors de la réception d'un message :

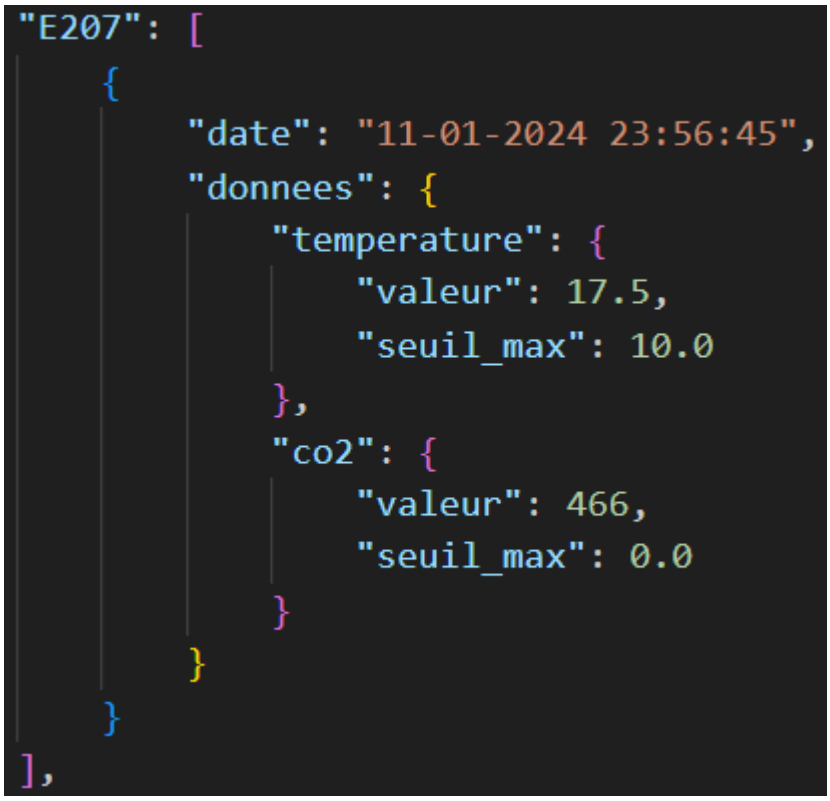
```
Message reçu sur le topic AM107/by-room/B212/data
Payload brut: [{'temperature': 17, 'humidity': 40.5, 'activity': 0, 'co2': 551, 'tvoc': 15
red': 0, 'infrared_and_visible': 1, 'pressure': 1002.4}, {'deviceName': 'AM107-19', 'devEU
m': 'B212', 'floor': 2, 'Building': 'B'}]
{'date': '11-01-2024 23:57:20', 'donnees': {'temperature': {'valeur': 17, 'moyenne': 17.0}
5, 'moyenne': 40.5}, 'activity': {'valeur': 0, 'moyenne': 0.0}, 'co2': {'valeur': 551, 'mo
Alertes :
- temperature a dépassé le seuil maximum de 10.0, valeur actuelle : 17
- co2 a dépassé le seuil maximum de 0.0, valeur actuelle : 551
```

Affiche le message reçu en brut, affiche les alertes pour les seuils dépassés par la nouvelle donnée. L'écriture dans le fichier se fait dans le même temps.

Voici à quoi ressemble une donnée en Json :



Et une alerte :



Cas d’erreurs

Champ du fichier de configuration vide :

Si un champ du fichier de configuration est vide comme ceci (le topic ici) :

```
[MQTT]
broker=chirpstack.iut-blagnac.fr
port=1883
topic=
[CONFIG]
fichier_alerte=alerte
fichier_donnees=donnees
fichier_logs=logs
choix_donnees=temperature,humidity,activity,co2
typeTemps=seconde(s)
frequence_affichage=0.0
[ALERT]
son_Alertes=98.0
seuil_Temperature=10.0
seuil_Humidity=50.0
seuil_CO2=0.0
seuil_Activity=2.0
```

Le script Python va planter avec comme erreurs :

```
  File "c:\Users\1\Documents\GitHub\sae3-solo\connect.py", line 120, in on_connect
    client.subscribe(topic)
  File "C:\Users\1\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5
    raise ValueError('Invalid topic.')
ValueError: Invalid topic.
```

Nous avons choisi de ne pas gérer les cas d'erreurs ici car cela sera fait à travers l'application Java. Si dans l'application par exemple l'utilisateur a oublié ou décidé de ne pas remplir le champ topic, le champ sera automatiquement rempli par défaut.

Fichier de configuration non trouvé :

Cette erreur se produit lorsque aucun fichier de configuration n'est trouvé par le script Python, comme indiqué dans les documentations le fichier de configuration doit se trouver dans le même répertoire que le script Python. Ce fichier de configuration peut être recréé à travers l'application Java-FX.

```
PS C:\Users\1\Documents\GitHub\sae3-solo> & C:/Users/1/AppData/Local/Microsoft/WindowsApps/python3
Répertoire de travail actuel : C:\Users\1\Documents\GitHub\sae3-solo
Traceback (most recent call last):
  File "c:\Users\1\Documents\GitHub\sae3-solo\connect.py", line 21, in <module>
    frequence_affichage = config.getfloat('CONFIG', 'frequence_affichage')
                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Informations MQTT incorrects :

Au cas où les informations de connexion au serveur MQTT ne sont pas bons :

```
PS C:\Users\1\Documents\GitHub\sae3-solo> & C:/Users/1/AppData/Local/Microsoft/WindowsApps/python3.1
ocuments/GitHub/sae3-solo/connect.py
Répertoire de travail actuel : C:\Users\1\Documents\GitHub\sae3-solo
Fichiers de configuration trouvés : ['config.ini']
Sections trouvées : ['MQTT', 'CONFIG', 'ALERT']
Vous avez choisi d'afficher les données suivante :  ['temperature', 'humidity', 'activity', 'co2']
Echec de la connexion, code d'erreur : timed out
PS C:\Users\1\Documents\GitHub\sae3-solo>
```

Un message indiquant que la connexion a échoué apparaît et le script prend fin.

Erreur lors de l'écriture des données :

Une erreur peut apparaître lors de l'écriture des données dans les fichiers, cela peut être dû à plusieurs raisons :

- l'élément reçu et que le script tente d’écrire comporte une erreur
- le fichier Json sur lequel écrire est cassé (caractères mal placées tels que des ‘{‘ (doublons)

Dans ces cas, le fichier sur laquelle a eu lieu l’erreur est indiquée ainsi que plus d’informations sur l’erreur est affichée.