

SAÉ 3.01 2023-2024 - Documentation

Python - Équipe 5

Table des matières

1. Introduction	1
2. Prérequis	1
3. Installation	2
4. Utilisation	2
5. Explication de la structure du code	3
5.1. Le fichier de configuration	3
5.2. Fonctionnement de l'application python	4
6. Exemple d'utilisation	8
7. Tests	9
7.1. Test 1 : Lancement de l'application et réception des données normales	9
7.2. Test 2 : Réception des données d'alertes	10
7.3. Test 3 : Réception des données anormales	10
7.4. Test 4 : Temps d'attente	10
8. Conclusion	11

1. Introduction

Notre application python permet de récupérer les données envoyées par les différents capteurs présents dans les entrepôts de stockage de MalyArt qui les utilise pour s'assurer que les conditions de stockage sont optimales pour les oeuvres d'art qui y sont entreposées.

L'objectif est de pouvoir afficher ces données sur une application JavaFX pour que les employés de MalyArt puissent les consulter facilement.

2. Prérequis

L'application nécessite plusieurs choses qu'il vous faudra installer au préalable :

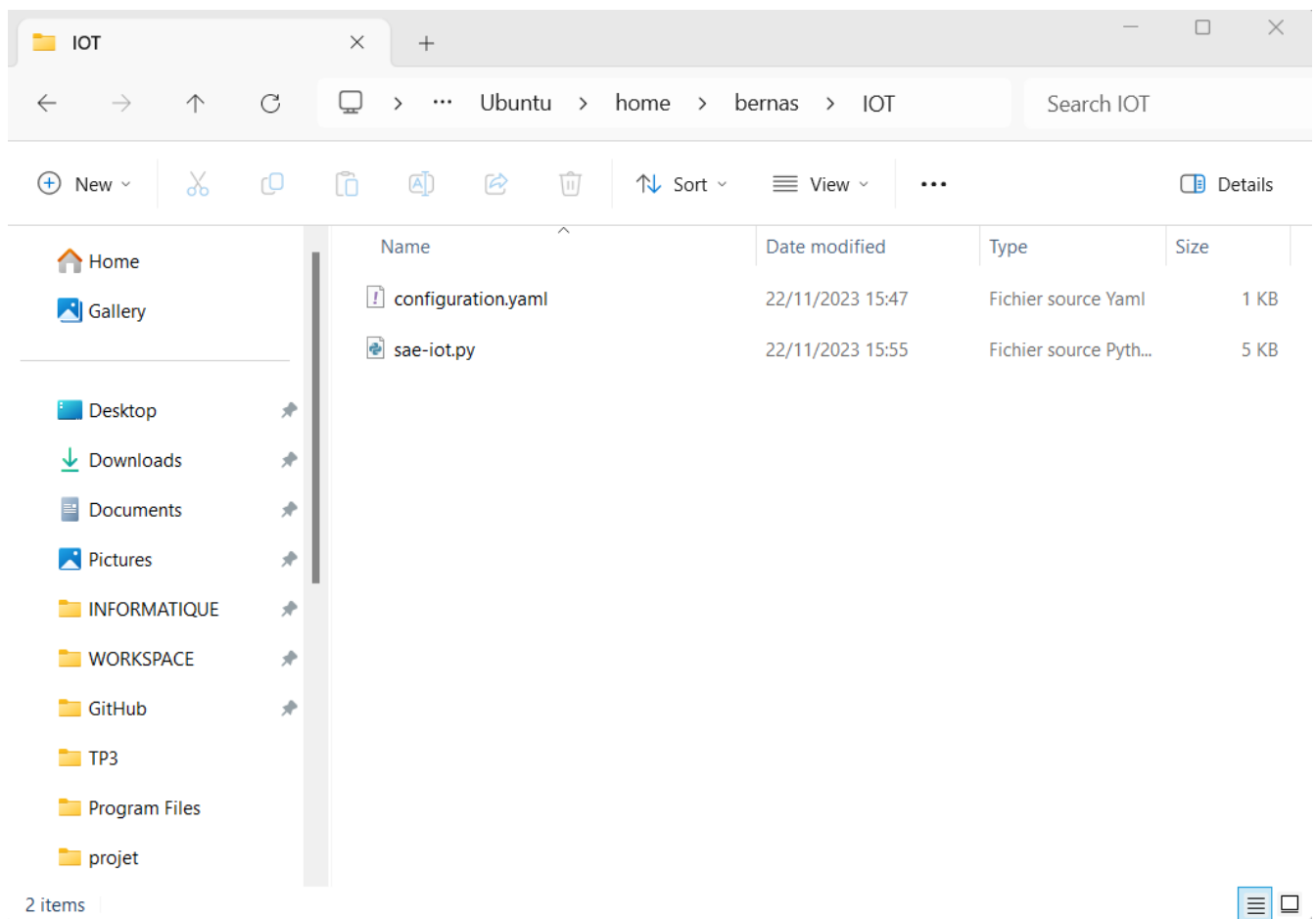
- **Python 3** ou une version ultérieure. (<https://www.python.org/downloads/>)
- **pip** pour installer les dépendances. (Inclus dans Python 3.4 et supérieur)
- **csv** pour lire les fichiers csv. (Bibliothèque Python)
- **yaml** pour lire les fichiers yaml. (**pip install pyyaml**)
- **paho.mqtt.client** pour communiquer avec le broker MQTT. (**pip install paho-mqtt**)
- **json** pour lire les fichiers json. (Bibliothèque Python)

- **time** pour gérer le temps. (Bibliothèque Python)
- **signal** pour gérer les signaux. (Bibliothèque Python)

WARNING | L'application ne fonctionne que sur **Linux**.

3. Installation

1. Téléchargez le fichier python à partir du lien suivant : [sae-iot.py](#)
2. Téléchargez le fichier de configuration à partir du lien suivant : [configuration.yaml](#)
3. Regroupez les deux fichiers dans un même dossier.

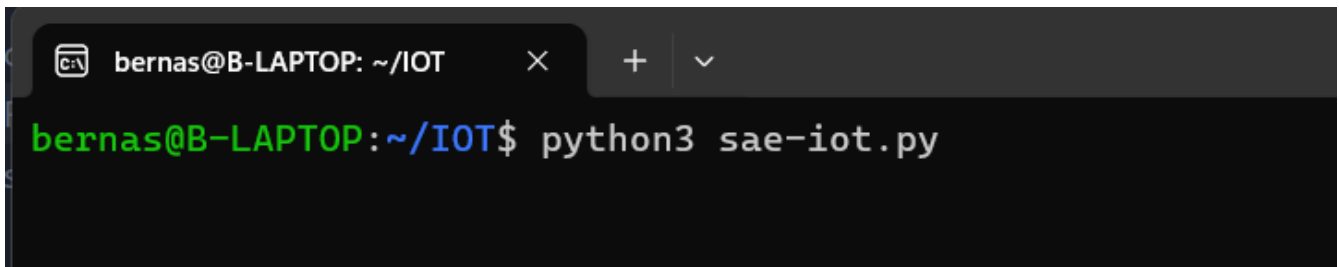


WARNING | Il est important que les noms des fichiers soient respectés.

4. Utilisation

Pour utiliser l'application, il vous suffit d'ouvrir un terminal dans le dossier contenant les deux fichiers et de lancer la commande suivante :

```
python3 sae-iot.py
```



```
bernas@B-LAPTOP: ~/IOT
bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
```

5. Explication de la structure du code

Le programme python est divisé en deux parties : - Le fichier de configuration (configuration.yaml)
- Le fichier principal (sae-iot.py)

5.1. Le fichier de configuration

Le fichier de configuration est un fichier yaml qui permet de configurer l'application. Il est divisé en plusieurs parties :

5.1.1. Le broker

Cette partie permet de configurer le broker MQTT. Il est possible de modifier l'adresse du broker et le port utilisé.

```
url: "chirpstack.iut-blagnac.fr"
port: 1883
keepalive: 60
```

5.1.2. Les topics

Cette partie permet de configurer les topics MQTT. Il est possible de modifier les topics auxquels l'application va s'abonner.

```
topics: ["AM107/by-room/+data", "AM107/by-room/E003/data", "AM107/by-room/E006/data"]
```

5.1.3. Les fichiers

Cette partie permet de configurer les fichiers csv où se trouvent les données des capteurs. Il est possible de modifier les fichiers csv utilisés par l'application.

```
dataFile: "data.csv"
alertFile: "alert.csv"
```

5.1.4. Les données des capteurs à récupérer

Cette partie permet de configurer les données des capteurs à récupérer. Il est possible de modifier

les données des capteurs à récupérer.

```
selectedData:
["temperature","humidity","co2","activity","tvoc","illumination","infrared","infrared_
and_visible","pressure"]
```

5.1.5. Les temps d'attente entre chaque écoute

Cette partie permet de configurer les temps d'attente entre chaque écoute. Il est possible de modifier les temps d'attente entre chaque écoute. (Le temps ou l'application va écouter les données des capteurs est aussi présent mais il n'est pas possible de le modifier dans l'application)

```
rest_duration : 30
running_time : 10
```

5.1.6. Les seuils d'alertes

Cette partie permet de configurer les seuils d'alertes. Il est possible de modifier les seuils d'alertes.

```
thresholds:
  temperature : 10
  humidity : 45
  co2 : 10000
  activity : 300
  tvoc : 500
  illumination : 100
  infrared : 100
  infrared_and_visible : 100
  pressure : 1100
```

5.2. Fonctionnement de l'application python

Notre application python fonctionne de la manière suivante :

5.2.1. Récupération des paramètres

L'application récupère les paramètres du fichier de configuration.

```
# Lecture du fichier de configuration
with open("configuration.yaml", "r") as file:
    print("~ Retrieving configuration file")
    config = yaml.safe_load(file)

print("~ Selected data : " + str(config["selectedData"]))
```

5.2.2. Connexion au broker MQTT

L'application se connecte au broker MQTT. Cela se fait grâce à la fonction `on_connect` qui est appelée lorsque l'application se connecte au broker MQTT.

Dans la même fonction, l'application s'abonne aux topics configurés dans le fichier de configuration.

```
def on_connect(client, userdata, flags, rc):
    print("~ Connected with result code " + str(rc))

    # Abonnement aux différents topics MQTT définis dans le fichier de configuration
    for topic in config["topics"]:
        try:
            client.subscribe(topic)
            print("~ Subscribed to " + topic)
        except Exception as e:
            print("~ Failed to subscribe to {}: {}".format(topic, str(e)))
```

5.2.3. Création des fichiers de données

Dans la fonction `on_connect`, l'application crée les fichiers de données configurés dans le fichier de configuration s'ils n'existent pas déjà.

```
# Creation du fichier de données si il n'existe pas
try:
    csvFile = open(config["dataFile"], "r", newline="")
except IOError:
    csvFile = open(config["dataFile"], "w", newline="")
    csv_writer = csv.writer(csvFile)
    csv_writer.writerow(["room", "time"] + config["selectedData"])
    print("~ Creating file " + config["dataFile"])
    csvFile.close()

# Creation du fichier d'alerte si il n'existe pas
try:
    alertFile = open(config["alertFile"], "r", newline="")
except IOError:
    alertFile = open(config["alertFile"], "w", newline="")
    alert_writer = csv.writer(alertFile)
    alert_writer.writerow(["room", "time", "alert"])
    print("~ Creating file " + config["alertFile"])
    alertFile.close()
```

5.2.4. Récupération des données

L'application reçoit les données des capteurs grâce à la fonction `on_message` qui est appelée lorsque l'application reçoit un message du broker MQTT.

Dans un premier temps, l'application récupère les données du message et les transforme en dictionnaire json. Puis elle ne retient que les données des capteurs configurés dans le fichier de configuration. Cela se fait grâce à un dictionnaire créé en compréhension qui ne retient que les données des capteurs configurés.

```
def on_message(client, userdata, msg):
    my_data = msg.payload.decode("utf-8")
    my_json = json.loads(my_data)

    # check si le nom de la salle est présent et si non, on passe à la suite
    try:
        room = my_json[1]["room"]
    except Exception as e:
        print("~ Nom de salle absent")
        return

    # check si il y a des données dans le json
    if len(my_json[0]) == 0:
        return

    # enregistrement des données dans un dictionnaire avec le temps
    # (si les données sont présentes dans le fichier de configuration et dans le json)
    data_values = {
        key: my_json[0][key] for key in config["selectedData"] if key in my_json[0]
    }
    data_values["time"] = time.time()
```

5.2.5. Enregistrement des données

L'application enregistre ensuite les données dans le fichier de données configuré dans le fichier de configuration.

```
# Écriture des données dans le fichier CSV
with open(config["dataFile"], "a", newline="") as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(
        [room, data_values["time"]]
        + [data_values[key] for key in config["selectedData"]]
    )
```

5.2.6. Check et enregistrement des alertes

L'application vérifie ensuite si les données reçues sont en dehors des limites configurées dans le fichier de configuration. Si c'est le cas, elle enregistre une alerte dans le fichier d'alertes.

```

text = ""
# Check si les données dépassent les seuils et écriture dans le fichier d'alerte le cas échéant
with open(config["alertFile"], "a", newline="") as alertfile:
    alert_writer = csv.writer(alertfile)
    for key, threshold in config["limites"].items():
        if key in data_values and data_values[key] > threshold:
            print(
                "Threshold exceeded - {}: {} (Threshold: {})".format(
                    key, data_values[key], threshold
                )
            )
            text += (
                str(key)
                + " "
                + str(data_values[key])
                + " ("
                + str(threshold)
                + ") "
            )
    alert_writer.writerow([room, data_values["time"], text])

```

5.2.7. Moyenne des données

L'application calcule ensuite la moyenne des données reçues et l'affiche dans la console.

```

# Affichage de la moyenne des données de la pièce
def affichage_Moyenne(room):
    with open(config["dataFile"], "r") as csvfile:
        csv_reader = csv.DictReader(csvfile)
        data = [row for row in csv_reader if row["room"] == room]
        if data:
            print("Moyenne des données de la pièce " + room + " :")
            message = ""
            for key in config["selectedData"]:
                if key != "time":
                    values = [
                        float(row[key])
                        for row in data
                        if key in row and row[key].strip()
                    ]
                    if values:
                        moyenne = round(sum(values) / len(values), 2)
                        message += key + " : " + str(moyenne) + " "
            print(message + "\n")

```

Cette moyenne est calculée en ouvrant le fichier de données, en lisant les données et en calculant la moyenne de chaque type de données pour la pièce.

5.2.8. Fréquence de récupération des données

L'application attend ensuite la durée de pause configurée dans le fichier de configuration avant de récupérer de nouveau les données. Pour cela, nous utilisons la librairie `signal` qui permet de gérer les signaux.

```
running_time = config["running_time"] * 60 # temps d'exécution transformé en minutes
rest_duration = config["rest_duration"] * 60 # temps de repos transformé en minutes

def handle_execution(signum, frame):
    print("Exécution pendant {} secondes...".format(running_time))
    start_time = time.time()
    end_time = start_time + running_time

    while time.time() < end_time:
        client.loop(timeout=1.0, max_packets=1)

    # Repos après la période d'exécution
    signal.alarm(rest_duration) # Définition de l'alarme pour la période de repos
    print("Pause pendant {} secondes...".format(rest_duration))

# Définition des signaux d'alarme pour les périodes d'exécution et de repos
signal.signal(signal.SIGALRM, handle_execution)

# Activation de l'alarme pour la première période d'exécution
# (2 secondes le temps que la connection se fasse)
signal.alarm(2)

# Boucle infinie pour attendre les signaux d'alarme
while True:
    signal.pause() # Pause jusqu'à ce qu'un signal d'alarme se déclenche
```

Le fonctionnement est le suivant :

- L'application crée un signal `SIGALRM` qui est appelé lorsque la durée de pause est écoulée, ce signal appelle la fonction `handler`.
- Lors de l'appel de la fonction `handler` (ici `handle_execution`), celle-ci débute l'écoute des données du broker MQTT tant que la durée de récupération des données n'est pas écoulée.
- Lorsque la durée de récupération des données est écoulée, la fonction `handle_execution` lance une nouvelle alarme qui débute le temps de pause.
- Pour l'attente de l'alarme nous avons une boucle infinie qui attend la réception du signal `SIGALRM`.

6. Exemple d'utilisation

Pour tester notre application, nous avons utilisé le broker MQTT `chirpstack.iut-blagnac.fr` et le topic `AM107/by-room/+data`.


```

bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
~ Retrieving configuration file
~ Selected data : ['temperature', 'humidity', 'co2', 'activity', 'tvoc', 'illumination', 'infrared', 'infrared_and_visible', 'pressure']
Exécution pendant 600 secondes...
~ Connected with result code 0
~ Subscribed to AM107/by-room/+data
~ Saving data on : data.csv
~ Waiting for data

[B103] temperature: 23.1, humidity: 43.5, co2: 3763, activity: 0, tvoc: 246, illumination: 43, infrared: 7, infrared_and_visible: 34, pressure: 985.4,
time: 1701095206.7183073
Threshold exceeded - humidity: 43.5 (Threshold: 40)
Moyenne des données de la pièce B103 :
temperature : 23.05 humidity : 43.0 co2 : 3664.5 activity : 0.5 tvoc : 237.5 illumination : 45.0 infrared : 7.0 infrared_and_visible : 35.0 pressure :
985.6

[E106] temperature: 17.1, humidity: 47, co2: 513, activity: 0, tvoc: 174, illumination: 0, infrared: 1, infrared_and_visible: 2, pressure: 985.4, time:
1701095207.2361467
Threshold exceeded - humidity: 47 (Threshold: 40)
Moyenne des données de la pièce E106 :
temperature : 17.1 humidity : 46.0 co2 : 517.5 activity : 0.0 tvoc : 158.0 illumination : 0.5 infrared : 1.5 infrared_and_visible : 3.0 pressure : 985.
6

```

L'application récupère les données du capteur **AM107** et les enregistre dans le fichier **data.csv** et les alertes dans le fichier **alert.csv**.

data.csv	alert.csv
1 room,time,temperature,humidity,co2,activity,tvoc,illumination,infrared,infrared_and_visible,pressure	1 room,time,alert
2 E006,1701092014.8094735,17.4,50,478,0,315,27,12,36,985.6	2 E006,1701092014.8094735,humidity 50 (40)
3 C006,1701092028.526804,24.1,30.5,521,112,59,109,17,85,985.6	3 C006,1701092028.526804,temperature 24.1 (24) illumination 109 (100)
4 B202,1701092045.6584003,20.5,43,1155,0,167,2,2,5,985.6	4 B202,1701092045.6584003,humidity 43 (40)
5 E004,1701092047.164517,18.4,45,567,0,227,23,10,30,986.5	5 E004,1701092047.164517,humidity 45 (40)
6 E001,1701092050.7600217,18.6,54.5,1352,261,1413,88,15,71,986.6	6 E001,1701092050.7600217,humidity 54.5 (40) tvoc 1413 (500)
7 E003,1701092078.364234,19.7,45,681,0,329,14,8,22,986.2	7 E003,1701092078.364234,humidity 45 (40)
8 B106,1701092079.969332,21.5,44,1697,185,229,32,6,27,985.8	8 B106,1701092079.969332,humidity 44 (40)
9 Serveurs,1701092081.127198,24.6,35.5,556,0,195,1,1,1,985.8	9 Serveurs,1701092081.127198,temperature 24.6 (24)
10 B234,1701092105.4975646,23.2,33.5,1200,0,67,1,1,1,985.2	10 B234,1701092105.4975646,
11 E209,1701092139.680621,15.5,47,496,0,124,9,4,12,985.3	11 E209,1701092139.680621,humidity 47 (40)
12 hall-amphi,1701092168.4173813,18.4,49,640,128,338,10,7,18,985.7	12 hall-amphi,1701092168.4173813,humidity 49 (40)
13 E103,1701092168.8570578,20.3,43,1040,393,257,17,4,16,986	13 E103,1701092168.8570578,humidity 43 (40) activity 393 (300)
14 E207,1701092182.143176,17.3,46,597,0,202,208,64,221,985.6	14 E207,1701092182.143176,humidity 46 (40) illumination 208 (100) infrared_and_visible 221
15 B217,1701092196.06342,19.7,39,805,0,87,0,1,2,985.2	15 B217,1701092196.06342,
16 B212,1701092196.332042,18.2,45,972,0,133,38,12,41,985.4	16 B212,1701092196.332042,humidity 45 (40)
17 B113,1701092206.9553888,21.2,47,2026,85,238,34,7,30,985.1	17 B113,1701092206.9553888,humidity 47 (40)

7. Tests

Nous allons maintenant tester l'application en utilisant le broker MQTT de l'IUT de Blagnac. Pour cela, nous allons utiliser le topic **AM107/by-room/E003/data** qui contient les données du capteur de la salle E003.

7.1. Test 1 : Lancement de l'application et réception des données normales

Pour ce premier test, nous allons lancer l'application et vérifier que celle-ci reçoit bien les données du capteur de la salle E003.

```

bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
~ Retrieving configuration file
~ Selected data : ['temperature', 'humidity', 'co2', 'activity', 'tvoc', 'illumination', 'infrared', 'infrared_and_visible', 'pressure']
Exécution pendant 600 secondes...
~ Connected with result code 0
~ Subscribed to AM107/by-room/E003/data
~ Saving data on : data.csv
~ Waiting for data

[E003] temperature: 9, humidity: 37, co2: 442, activity: 0, tvoc: 249, illumination: 1, infrared: 1, infrared_and_visible: 1, pressure: 989.2, time: 1705266393.932716
Moyenne des données de la pièce E003 :
temperature : 16.79 humidity : 48.25 co2 : 580.92 activity : 26.25 tvoc : 458.75 illumination : 11.75 infrared : 4.92 infrared_and_visible : 14.0 pressure : 995.83

```

Comme nous pouvons le voir sur l'image ci-dessus, l'application a bien reçu les données du capteur de la salle E003 et les a affichées dans la console.

```

Ubuntu > home > bernas > IOT > data.csv > data
1 room,time,temperature,humidity,co2,activity,tvoc,illumination,infrared,infrared_and_visible,pressure
384 E003,1705266393.932716,9,37,442,0,249,1,1,1,989.2
385

```

Nous pouvons également voir que les données ont bien été enregistrées dans le fichier csv.

7.2. Test 2 : Réception des données d'alertes

Pour ce deuxième test, nous allons modifier les seuils d'alertes dans le fichier de configuration pour que l'application reçoive des données d'alertes.

```
bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
~ Retrieving configuration file
~ Selected data : ['temperature', 'humidity', 'co2', 'activity', 'tvoc', 'illumination', 'infrared', 'infrared_and_visible', 'pressure']
Exécution pendant 600 secondes...
~ Connected with result code 0
~ Subscribed to AM107/by-room/E003/data
~ Saving data on : data.csv
~ Waiting for data

[E003] temperature: 25, humidity: 70, co2: 442, activity: 0, tvoc: 249, illumination: 1, infrared: 1, infrared_and_visible: 1, pressure: 989.2, time: 1705266743.2031426
Threshold exceeded - temperature: 25 (Threshold: 10)
Threshold exceeded - humidity: 70 (Threshold: 45)
Moyenne des données de la pièce E003 :
temperature : 17.42 humidity : 49.92 co2 : 570.23 activity : 24.23 tvoc : 442.62 illumination : 10.92 infrared : 4.62 infrared_and_visible : 13.0 pressure : 995.32
```

Comme nous pouvons le voir sur l'image ci-dessus, l'application a bien reçu les données d'alertes du capteur de la salle E003 et les a affichées dans la console et en plus nous avons des données d'alertes affichées dans le terminal et dans le fichier csv d'alertes.

```
Ubuntu > home > bernas > IOT > alert.csv > data
1 room,time,alert
531 E003,1705266743.2031426,temperature 25 (10) humidity 70 (45)
532
```

7.3. Test 3 : Réception des données anormales

Pour le troisième test, nous allons modifier le message reçu par l'application pour que celle-ci reçoive des données anormales.

```
bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
~ Retrieving configuration file
~ Selected data : ['temperature', 'humidity', 'co2', 'activity', 'tvoc', 'illumination', 'infrared', 'infrared_and_visible', 'pressure']
Exécution pendant 600 secondes...
~ Connected with result code 0
~ Subscribed to AM107/by-room/E003/data
~ Saving data on : data.csv
~ Waiting for data

~ Erreur lors de la réception des données
```

Nous pouvons voir sur l'image ci-dessus que l'application a signalé que les données reçues étaient anormales.

7.4. Test 4 : Temps d'attente

Pour le quatrième test, nous allons modifier le temps d'attente entre chaque écoute pour que le temps d'attente soit réduit.

```

bernas@B-LAPTOP:~/IOT$ python3 sae-iot.py
~ Retrieving configuration file
~ Selected data : ['temperature', 'humidity', 'co2', 'activity', 'tvoc', 'illumination', 'infrared', 'infrared_and_visible', 'pressure']
Exécution pendant 60 secondes...
~ Connected with result code 0
~ Subscribed to AM107/by-room/E003/data
~ Saving data on : data.csv
~ Waiting for data

[E003] temperature: 9.4, humidity: 82, co2: 544, activity: 0, tvoc: 2644, illumination: 1, infrared: 1, infrared_and_visible: 1, pressure: 988.8, time: 1705267518.9345012
Threshold exceeded - humidity: 82 (Threshold: 45)
Threshold exceeded - tvoc: 2644 (Threshold: 500)
Moyenne des données de la pièce E003 :
temperature : 16.89 humidity : 51.77 co2 : 561.2 activity : 21.0 tvoc : 585.13 illumination : 9.6 infrared : 4.13 infrared_and_visible : 11.4 pressure : 994.52

[E003] temperature: 17.4, humidity: 45.5, co2: 459, activity: 0, tvoc: 378, illumination: 1, infrared: 1, infrared_and_visible: 1, pressure: 989.9, time: 1705267531.821227
Threshold exceeded - temperature: 17.4 (Threshold: 10)
Threshold exceeded - humidity: 45.5 (Threshold: 45)
Moyenne des données de la pièce E003 :
temperature : 16.92 humidity : 51.38 co2 : 554.81 activity : 19.69 tvoc : 572.19 illumination : 9.06 infrared : 3.94 infrared_and_visible : 10.75 pressure : 994.23

Pause pendant 60 secondes...
Exécution pendant 60 secondes...
[E003] temperature: 16.6, humidity: 40.5, co2: 534, activity: 0, tvoc: 36, illumination: 1, infrared: 1, infrared_and_visible: 1, pressure: 989.8, time: 1705267634.908242
Threshold exceeded - temperature: 16.6 (Threshold: 10)
Moyenne des données de la pièce E003 :
temperature : 16.9 humidity : 50.74 co2 : 553.59 activity : 18.53 tvoc : 540.65 illumination : 8.59 infrared : 3.76 infrared_and_visible : 10.18 pressure : 993.97

```

Nous pouvons voir sur l'image ci-dessus que l'application a bien effectué les écoutes avec une pause de 60 secondes.

8. Conclusion

Notre application python permet de récupérer les données envoyées par les différents capteurs présents dans les entrepôts de stockage de MalyArt qui les utilise pour s'assurer que les conditions de stockage sont optimales pour les oeuvres d'art qui y sont entreposées.