Procédure de mise en place de la récupération de données MQTT dans le cadre de la SAE 24

PICHOT Owen,
NOACCO Lilian,
BOURGER Pierre

1. Préambule sur la génération des données

Pour la génération du signal, nous ne sommes pas passés par le topic fourni par défaut par Mr Mura, nous avons à la place créé un petit script permettant la récupération des données d'une petite station météo basée à Trois-Epis (sur un Raspberry Pi 400) ce qui permet, premièrement, d'avoir des données réelles et cohérente d'un vrai climat, mais également de ne pas dépendre de la fiabilité du topic ou bien des modifications des strings de données envoyées.

Les données sont donc ensuite envoyé sur le topic MQTT du broker 'Mosquitto' : '`OWEN/SAE24/Maison`' sous la forme suivante :

<variable_humidité>|<variable_temperature>|<variable_pression>|<date>|<heure>|<Pièce>#...

...ceci se répète autant de fois qu'il y a de pièces...

La raison de cette syntaxe d'envoi permet une déconstruction très simple lors de la gestion du message dans le script de récupération python.

Chaque pièce est séparée par un '#' et chaque variable au sein de chaque pièce est séparée par un '|'.

Passons maintenant à la section concernant la récupération de ces messages....

```python
import paho.mqtt.client as mqtt
import datetime
import os, platform, mysql.connector


def connect_brocker(client, userdata, flags, rc):
    client.subscribe("OWEN/SAE24/Maison")


def get_data(client, userdata, msg):

    clear_terminal()#fonction externe purement esthétique

    message_complet = msg.payload.decode() #Message non trié

    affichage_terminal()#fonction externe purement esthétique

```

```
18
19      messages_separés = message_complet.split("#")  # Pièce par pièce
20
21      for message in messages_separés:
22          data = message.split('|')  # Séparation de chaque item contenant un '|'
23          room = data[-1]  # Attrape le nom de la pièce à la fin de la liste séparée
24
25
26          insert_data_to_db(room.strip(), data[0], data[1], data[2], data[3], data[4])
27
28
29  client = mqtt.Client()
30  client.on_connect = connect_brocker
31  client.on_message = get_data
32
33  client.connect("test.mosquitto.org", 1883, 60)
34
35  client.loop_forever()
```

2. Réception & traitement des données :

Voyons plus en détail ce que le code fait et pourquoi :

No de ligne du code	Explication
1-3	Importation des librairies nécessaires au bon fonctionnement du code
29	Initialisation de la connection
33	Connection au broker sélectionné (Mosquitto)
30 - 6,7	Connection au topic spécifique du broker une fois la connection avec ce dernier établit
31	Lancement de la fonction de récupération des données
14	Récupération du message en soit
19	Division du message en autant de pièces qu'il y a de signes '#' dans le message.
21-26	Pour chaque item dans chaque pièce (séparés par des '|') on l'intègre dans un tableau. On prend ensuite le nom de la pièce en question (dernier item logique du message) pour l'ajouter à sa table respective par la suite.

3. Ajout des données traitées dans la base :

a. Préparation de la base SQL :

La préparation de la base est assez rapide, il n'y a besoin que du service SQL présent soit dans un conteneur soit sur la même machine hébergeant le service Nginx ou bien dans n'importe quel emplacement du VLAN 30 de notre installation réseau (sous réserve que le service Nginx ait accès à la base).

Il faut ensuite créer un nouvel utilisateur dans notre base SQL pour que notre script python puisse l'utiliser et insérer des données dans la base. Ci-dessous la commande pour créer un utilisateur :

mysql> CREATE USER '<nom_utilisateur>'@'<adresse_utilisateur>' IDENTIFIED BY '<mot_de_passe>' ;

On assigne ensuite à notre utilisateurs les droits d'accès à notre base de donnée qui sera utilisée pour le stockage des données :

mysql> CREATE DATABASE MQTT_MAISON_OWEN;
mysql> GRANT ALL PRIVILEGES ON MQTT_MAISON_OWEN TO '<nom_utilisateur>'@'<adresse_utilisateur>';

Il ne nous reste maintenant plus qu'à utiliser le reste de notre script de récupération des données pour insérer celles-ci dans la base de données nouvellement créée.

b. Insertion des données dans la base :

Ci-dessous une prise d'écran de la fonction 'insert_data_to_db' mentionnée plus tôt :

```
5  def insert_data_to_db(emplacement, temperature, humidite, pression, date, heure):
6      conn = mysql.connector.connect(
7          host='localhost',
8          user='owen',
9          password='j38fqt',
10         database='MQTT_MAISON_OWEN'
11     )
12
13     cursor = conn.cursor()
14
15     try:
16         converted_date = datetime.datetime.strptime(date, '%d/%m/%Y').date()
17
18         query = "INSERT INTO {} (emplacement, temperature, humidite, pression, date, heure) VALUES (%s, %s, %s, %s, %s, %s)".format(emplacement)
19         values = (emplacement, temperature, humidite, pression, converted_date, heure)
20         cursor.execute(query, values)
21
22         #Sauvegarde de la bdd et fermeture de celle-ci
23         conn.commit()
24         conn.close()
```

```
25      except mysql.connector.Error as error:
26          if error.errno == 1054 or error.errno == 1146: #Si la table n'existe pas (error 1054 / 1146 SQL), alors :
27              #on créé la table
28              query_create_table = "CREATE TABLE {} (id INT AUTO_INCREMENT PRIMARY KEY, emplacement TEXT, temperature REAL, humidite REAL, pression REAL, date DAT
29              cursor.execute(query_create_table)
30
31              conn.commit()
32
33              #on insert les données dans la table nouvellement créée
34              converted_date = datetime.datetime.strptime(date, '%d/%m/%Y').date()
35
36              query_insert_data = "INSERT INTO {} (emplacement, temperature, humidite, pression, date, heure) VALUES (%s, %s, %s, %s, %s, %s)".format(emplacement)
37              values = (emplacement, temperature, humidite, pression, converted_date, heure)
38              cursor.execute(query_insert_data, values)
39
40              conn.commit()
41              conn.close()
42          else:
43              print("Catastrophic error, please reload the page..\n")
44              print(error)
```

De nouveau, voyons ce que le code fait plus en détail :

No de ligne du code	Explication
6-10	On se connecte à la BDD en utilisant les crédentiels de l'utilisateur créé plus tôt
15-24	Si la pièce du message en question à déjà une table dans la BDD, alors il n'y a qu'à insérer les données dans la dite table (voir page suivante pour la description de celle-ci).
25-41	Si les données viennent d'un nouveau capteur et que celui-ci n'a donc pas de table attribuée dans la base de données, il faut alors d'abord la créer puis seulement ensuite y insérer les données. La table manquante peut être détectée via l'erreur SQL 1054 ou 1146. Il y a également une partie du code qui permet la conversion des dates du format américain au format européen pour une lecture plus claire une fois les données affichées sur un médium.

N.B : Nous avons bien conscience qu'il n'y a pas, dans notre installation, d'utilisation de clefs étrangères et que l'entièreté du projet pourrait donc être réalisée à l'aide de NoSQL.

Cependant, à la vue de l'échelle du cahier des charges (uniquement quelques données provenant de moins d'une dizaine de capteurs.) Il était plus logique pour nous de diviser chaque capteur avec sa propre table.
Cela permet également une gestion plus fluide des données de chaque capteur sur le front-end et il est évident que dans le cadre d'une infrastructure plus grande (un immeuble avec plus de 1000 bureaux contenant chacun un capteur par bureau par exemple), il serait alors plus logique de mettre directement chaque capteur dans la même table étant donné qu'il serait impossible d'afficher chaque capteur sur une seule et même page web.

4. Présentation d'une table type (Salon) :

Ci-dessous une capture d'écran de la table correspondant à l'un des capteurs actifs, 'Salon'.

```
mysql> describe Salon;
+-------------+--------+------+-----+---------+----------------+
| Field       | Type   | Null | Key | Default | Extra          |
+-------------+--------+------+-----+---------+----------------+
| id          | int    | NO   | PRI | NULL    | auto_increment |
| emplacement | text   | YES  |     | NULL    |                |
| temperature | double | YES  |     | NULL    |                |
| humidite    | double | YES  |     | NULL    |                |
| pression    | double | YES  |     | NULL    |                |
| date        | date   | YES  |     | NULL    |                |
| heure       | time   | YES  |     | NULL    |                |
+-------------+--------+------+-----+---------+----------------+
7 rows in set (0,04 sec)
```

Chaque capteur possède donc les attributs suivants :

- Un id (auto-incrémenté permettant de le 'target',
- L'emplacement du capteur actuel (logiquement identique au nom de sa table), cet attribut est présent dans le cas justement où l'on souhaiterait regrouper tous les capteurs dans une seule table si il y plusieurs capteurs par pièces.
- La température relevée à un moment T par le capteur,
- L'humidité relevée à un moment T par le capteur,
- La pression atmosphérique relevée à un moment T par le capteur,
- La date à laquelle la donnée à été prise
- L'heure à laquelle la donnée à été prise.

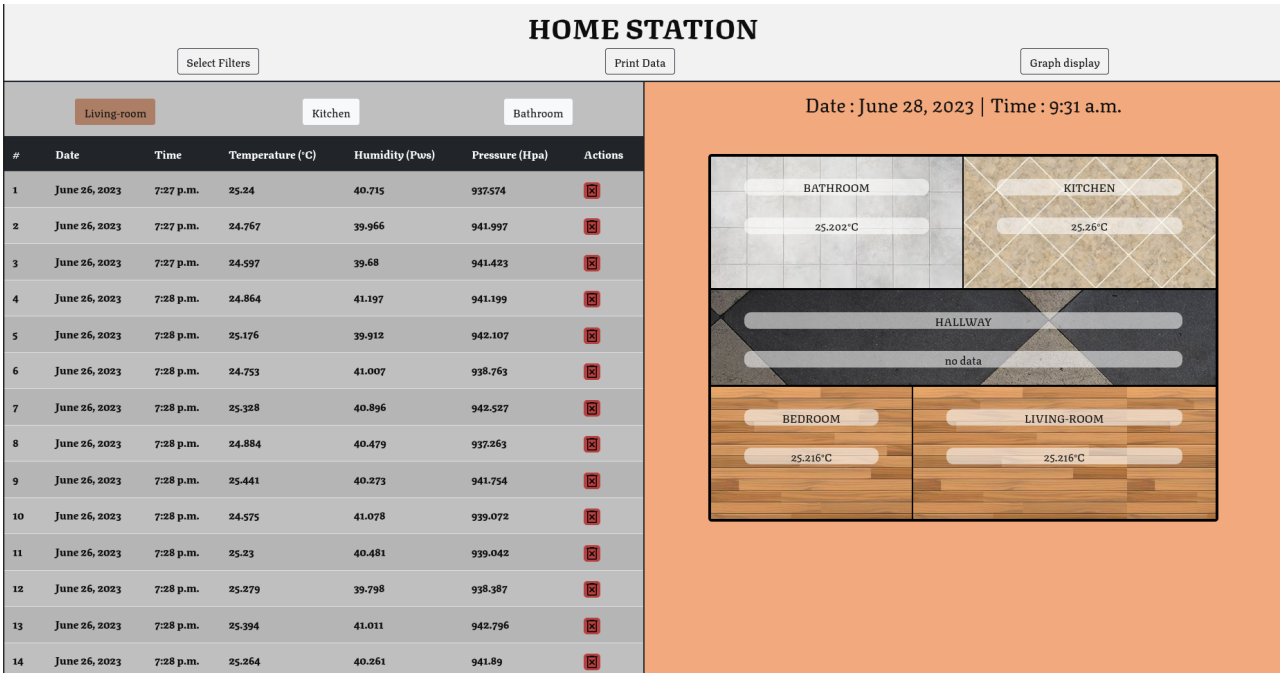
FIN

Merci pour votre attention.

Rapport de mise en place d'une application web dans le cadre de la SAE 24

PICHOT Owen,
NOACCO Lilian,
BOURGER Pierre

1. Vue générale du site :



#	Date	Time	Temperature (°C)	Humidity (Pws)	Pressure (Hpa)	Actions
1	June 26, 2023	7:27 p.m.	25.24	40.715	937.574	
2	June 26, 2023	7:27 p.m.	24.767	39.966	941.997	
3	June 26, 2023	7:27 p.m.	24.597	39.68	941.423	
4	June 26, 2023	7:28 p.m.	24.864	41.197	941.199	
5	June 26, 2023	7:28 p.m.	25.176	39.912	942.107	
6	June 26, 2023	7:28 p.m.	24.753	41.007	938.763	
7	June 26, 2023	7:28 p.m.	25.328	40.896	942.527	
8	June 26, 2023	7:28 p.m.	24.884	40.479	937.263	
9	June 26, 2023	7:28 p.m.	25.441	40.273	941.754	
10	June 26, 2023	7:28 p.m.	24.575	41.078	939.072	
11	June 26, 2023	7:28 p.m.	25.23	40.481	939.042	
12	June 26, 2023	7:28 p.m.	25.279	39.798	938.387	
13	June 26, 2023	7:28 p.m.	25.394	41.011	942.796	
14	June 26, 2023	7:28 p.m.	25.264	40.261	941.89	

Notre site web est basé sur un projet Django, ce framework permet l'utilisation de squelettes pour chaque partie de la page globale ce qui permet de modifier chaque partie de la page indépendamment.

Si l'on se base sur la prise d'écran ci dessus de la page d'accueil du site web nous pouvons voir que le site est composé de trois blocs principaux :

- Le 'header' contenant les liens de navigations vers d'autres parties / fonctionnalités comme par exemple l'impression de données ou les affichages graphiques.
- Le 'Tableau' qui permet la visualisation complète des données contenues dans chacune des tables correspondantes à chaque pièce dont on récolte des données environnementales.
- Le 'Schéma' qui permet une visualisation en vue du dessus de la maison / de l'appartement ou les données sont récoltées et qui se contente de n'afficher que les dernières données en date.

Chaque section peut donc être modifiée indépendamment sans retoucher l'entièreté du site.

Passons maintenant à l'aspect plus technique des choses..

Dans les prochaines sections nous verrons pour chaque partie du site et son organisation globale l'organisation et le code qu'il y a derrière. Ceci commence par :

2. Le squelette du site :

```
{% extends 'HEADER.html' %}

{% load static %}

{% block HEAD %}
    <link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">
    <title>OVERVIEW</title>
{% endblock %}

{% block TABLEAU %}

    {% include 'tableau.html' %}

{% endblock %}

{% block MAISON %}

    {% include 'house.html' %}

{% endblock %}
```

Ci-contre est une prise d'écran du fichier 'index.html', qui est à la base du squelette de l'application.
Il est composé de trois blocks, dont l'un des trois permet l'insertion de balises dans la section <head> des fichiers html (pour modifier le nom de l'onglet de la page par exemple ou charger des fichiers css spécifiques à un bloc).
Les deux autres blocs quant à eux représentent la partie gauche et droites de la page principale dans laquelle on peut insérer des éléments.
La page index est ensuite étendue depuis une autre page qui s'appelle 'HEADER.html', celle-ci permet uniquement la génération du bloc de navigation du haut de page du site avec notamment les liens de navigations et le titre du site qui sont identiques sur chacune des pages.
Ci-dessous une prise d'écran de la section header de 'HEADER.html' contenant tout ceci.

```
<header class="d-flex flex-nowrap flex-column pt-2 pb-2 border-bottom">
    <div class="d-flex justify-content-center">
        <a href="{% url 'IOT:home' %}"><h1 class="google_custom_font_bold">HOME STATION</h1></a>
    </div>
    <div>
        <nav class="d-flex justify-content-around">
            <a href="{% url 'IOT:Select_filters' %}"><button class="btn btn-outline-dark google_custom_font_medium">Select Filters</button></a>

            <div class="dropdown">
                <button class="dropbtn btn btn btn-outline-dark google_custom_font_medium">Print Data</button>
                <div class="dropdown-content">
                    <a href="{% url 'IOT:save_LR' 'LR' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print living room data</a>
                    <a href="{% url 'IOT:save_LR' 'KT' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print Kitchen data</a>
                    <a href="{% url 'IOT:save_LR' 'BT' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print bathroom data</a>
                </div>
            </div>

            <div class="dropdown">
                <button class="dropbtn btn btn-outline-dark google_custom_font_medium">Graph display</button>
                <div class="dropdown-content">
                    <a href="{% url 'IOT:graph_3' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 3 entries</a>
                    <a href="{% url 'IOT:graph_10' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 10 entries</a>
                    <a href="{% url 'IOT:graph_50' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 50 entries</a>
                </div>
            </div>


        </nav>
    </div>
</header>
```

On retrouve dans cette section chacun des liens de navigation du site web et si l'on souhaite, par exemple, changer le nom d'affichage de l'onglet ou bien insérer un fichier statique type css ou Js, il n'y aurait alors qu'à déclarer ceux ci en haut de page dans la partie du squelette liée à l'index 'HEAD'.

{% block HEAD %}{% load static %}<script src='mon_script'></script>{% endblock %}

Parlons maintenant des sections que l'on va insérer dans notre page 'index.html'.

3. Affichage complet des données :

Il est possible sur notre site, via des boutons, de naviguer entre les différents capteurs installés dans la maison afin d'en afficher toutes ses données (non filtrées pour l'instant).
Pour des raisons pratiques je ne montrerais ici que la partie tableau du capteur 'Salon' étant donné que les deux autres sont globalement identiques et que seul le Javascript permettant de switcher entre les trois est intéressant.

La première vue à être appelé lorsque l'url que l'utilisateur entre pour accéder au site (donc url vide), est notre page d'index qui est va se charger de charger l'index ainsi que toutes les données qui seront nécessaires à afficher.

```python
def home(request):
    donnees_salon_all = Salon.objects.all()
    donnees_cuisine_all = Cuisine.objects.all()
    donnees_sdb_all = SDB.objects.all()

    donnees_salon_latest = Salon.objects.order_by('-id').first()
    donnees_cuisine_latest = Cuisine.objects.order_by('-id').first()
    donnees_sdb_latest = SDB.objects.order_by('-id').first()

    context = {
                'donnees_salon_latest':donnees_salon_latest,
                'donnees_cuisine_latest': donnees_cuisine_latest,
                'donnees_sdb_latest':donnees_sdb_latest,
                'donnees_salon_all': donnees_salon_all,
                'donnees_cuisine_all': donnees_cuisine_all,
                'donnees_sdb_all':donnees_sdb_all,
               }

    return render(request, 'index.html', context)
```

On va donc pour se faire aller chercher les différents objets dont nous aurons besoin qui sont déclarés dans notre fichier models.py de l'application (voir ci-dessous avec un exemple de Salon).

```python
class Salon(models.Model):
    emplacement = models.TextField(blank=True, null=True)
    temperature = models.FloatField(blank=True, null=True)
    humidite = models.FloatField(blank=True, null=True)
    pression = models.FloatField(null=True)
    date = models.DateField(blank=True,null=True)
    heure = models.TimeField(null=True)
```

On charge alors tous les modèles de chacune des tables afin de les insérer par la suite dans leurs tableaux respectifs. On isole également le dernier item de chaque table pour afficher les dernières données sur le schéma et l'heure/date de la dernière prise de données.

On appel ensuite dans notre template 'index' le fichier 'tableau.html' (voir prise d'écran début de page suivante) dans laquelle on pourra alors insérer les données que notre contrôleur 'home' aura permit d'acheminer.

Il y a également pour chaque item de chaque capteur la possibilité de supprimer l'une des lignes de données en passant par un contrôleur 'delete_<nom_capteur>_item' nous pouvons voir ci contre un exemple pour effacer un item de la catégorie de capteurs du Salon mais cela fonctionne de la même manière pour un capteur de la cuisine ou d'un autre endroit.

```python
def delete_salon_item(request, id):
    item = Salon.objects.get(id=id)

    item.delete()

    return home(request)
```

```
<div class="rounded d-flex justify-content-around mt-4 mb-3">
    <span class="btn btn_custom mr-1 google_custom_font_medium" id="salon">Living-room</span>
    <span class="btn btn-light ml-1 google_custom_font_medium" id="cuisine">Kitchen</span>
    <span class="btn btn-light ml-1 google_custom_font_medium" id="sdb">Bathroom</span>
</div>



<table class="table table-striped table-hover" id="table_salon" style="display: table;">
    <thead class="thead-dark sticky-top">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Date</th>
            <th scope="col">Time</th>
            <th scope="col">Temperature (°C)</th>
            <th scope="col">Humidity (Pws)</th>
            <th scope="col">Pressure (Hpa)</th>
            <th scope="col">Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for donnees in donnees_salon_all %}
            <tr>
                <th scope="row" class="text-align">{{donnees.id}}</th>
                <td class="text-align">{{donnees.date}}</td>
                <td class="text-align">{{donnees.heure}}</td>
                <td class="text-align">{{donnees.humidite}}</td>
                <td class="text-align">{{donnees.temperature}}</td>
                <td class="text-align">{{donnees.pression}}</td>
                <td class="text-align"><a href="{% url 'IOT:delete_salon_item' donnees.id %}"><
            </tr>
        {% endfor %}
    </tbody>
</table>
```

Ci-contre l'affichage d'un des tableaux (pour le Salon), qui, de nouveau ne diffère pas énormément des deux autres concernant les autres capteurs.

On utilise notamment dans les tableaux la variable 'donnees_salon_all' qui aura été transmise en tant que contexte depuis notre contrôleur 'home'.

On peut ensuite afficher via une boucle for une ligne pour chaque sample de chaque capteur toujours en se basant sur le modèle que l'on a et de l'agencement de notre table dans la bdd.

Ci-dessous, nous avons le script (Javascript) qui permet de passer du tableau d'un capteur à un autre.
La procédure est assez simple et le script se contente de masquer ou d'afficher le tableau correspondant au boutons que l'utilisateur aura pressé. On commence par identifier les éléments du DOM (boutons, tableaux, etc..) avec lesquels on souhaite interagir par le biais d'id ou classes et on peut ensuite modifier leurs différents styles en fonction d'une situation X ou Y.

Par exemple : *Si bouton A cliqué, ALORS, change affichage pour le tableau A et retire les tableaux B & C.*

```
<script>
    const select_salon = document.getElementById("salon");
    const select_cuisine = document.getElementById("cuisine");
    const select_sdb = document.getElementById('sdb');

    function update_display(table_to_show, table_to_hide_a, table_to_hide_b, button_to_activate, button_to_hide_a, button_to_hide_b, title_name){
        document.getElementById(table_to_show).style.display = 'table';
        document.getElementById(table_to_hide_a).style.display = 'none';
        document.getElementById(table_to_hide_b).style.display = 'none';
        document.getElementById(button_to_activate).classList.remove('btn-light');
        document.getElementById(button_to_hide_a).classList.remove('btn_custom');
        document.getElementById(button_to_hide_b).classList.remove('btn_custom');
        document.getElementById(button_to_activate).classList.add('btn_custom');
        document.getElementById(button_to_hide_a).classList.add('btn-light');
        document.getElementById(button_to_hide_b).classList.add('btn-light');
    };

    select_salon.addEventListener('click', update_display('table_salon', 'table_cuisine', 'table_sdb', 'salon', 'cuisine', 'sdb', 'Living room data'))

    select_cuisine.addEventListener('click', update_display('table_cuisine', 'table_salon', 'table_sdb', 'cuisine', 'salon', 'sdb', 'Kitchen data'))

    select_sdb.addEventListener('click', update_display('table_sdb', 'table_salon', 'table_cuisine', 'sdb', 'cuisine', 'salon', 'Bathroom data'))
```

On peut aussi ajouter à une section du squelette ou bien à une page complète directement le script ci-dessous qui permet tout simplement de rafraîchir la page automatiquement toutes les X millisecondes (20 secondes dans notre cas).

```
<script>
    setInterval(function() {
        location.reload();
    }, 20000); // 5000 milliseconds = 5 seconds
</script>
```

4. Filtrages des données de la page principale :

Pour filtrer les données affichées par défaut (toutes) sur la page principal, l'utilisateur peut avoir accès au menu ci-contre :

Chaque filtre est indépendant de l'autre et il est donc possible de n'en renseigner qu'un, deux ou tous..

La seule exception est la température ou il est impératif de renseigner un minimum ET un maximum sans quoi le contrôleur ne pourra pas aboutir avec sa requête.


```python
def Select_filters(request):
    return render(request, 'select_filters.html')

def home_filtered(request):
    donnees_salon_latest = Salon.objects.order_by('-id').first()
    donnees_cuisine_latest = Cuisine.objects.order_by('-id').first()
    donnees_sdb_latest = SDB.objects.order_by('-id').first()

    donnees_salon = Salon.objects.all()
    donnees_cuisine = Cuisine.objects.all()
    donnees_sdb = SDB.objects.all()

    date = request.GET.get('date')
    date_fin = request.GET.get('date_fin')
    heure = request.GET.get('heure')
    heure_fin = request.GET.get('heure_fin')
    temp = request.GET.get('temp')
    temp_max = request.GET.get('temp_max')

    if date:
        if date and date_fin:
            donnees_salon = donnees_salon.filter(date__range=(date, date_fin))
            donnees_cuisine = donnees_cuisine.filter(date__range=(date, date_fin))
            donnees_sdb = donnees_sdb.filter(date__range=(date, date_fin))
        else:
            donnees_salon = donnees_salon.filter(date=date)
            donnees_cuisine = donnees_cuisine.filter(date=date)
            donnees_sdb = donnees_sdb.filter(date=date)


    if heure:
        if heure and heure_fin:
            donnees_salon = donnees_salon.filter(heure__range=(heure, heure_fin))
            donnees_cuisine = donnees_cuisine.filter(heure__range=(heure, heure_fin))
            donnees_sdb = donnees_sdb.filter(heure__range=(heure, heure_fin))
        else:
            donnees_salon = donnees_salon.filter(heure=heure)
            donnees_cuisine = donnees_cuisine.filter(heure=heure)
            donnees_sdb = donnees_sdb.filter(heure=heure)

    if temp and temp_max:
        donnees_salon = donnees_salon.filter(humidite__range=(temp, temp_max))
        donnees_cuisine = donnees_cuisine.filter(humidite__range=(temp, temp_max))
        donnees_sdb = donnees_sdb.filter(humidite__range=(temp, temp_max))


    context = {
        'donnees_salon_latest':donnees_salon_latest,
        'donnees_cuisine_latest':donnees_cuisine_latest,
        'donnees_sdb_latest':donnees_sdb_latest,
        'donnees_salon':donnees_salon,
        'donnees_cuisine':donnees_cuisine,
        'donnees_sdb':donnees_sdb,
    }

    return render(request, 'index.html', context)
```

Ci-contre sont les contrôleurs permettant la gestion des filtres en fonction des actions de l'utilisateur.
Nous avons tout d'abord le contrôleur 'Select_filters' qui permet simplement de charger la page de sélection des filtres.

Une fois le bouton de validation des filtres pressé, le contrôleur 'home_filtered' prend le relais.
Il commence tout d'abord par récupérer chacun des champs du formulaire de filtrage et si l'un des champs à été complété alors on va affiner la liste complète des objets en retirant de la liste ceux qui ne correspondent pas au filtre appliqué.

On continue comme ceci pour chaque type de filtres jusqu'à avoir les listes d'objets finales que l'on peut alors renvoyer vers la page index qui se chargera de mettre à jour notre tableau avec uniquement les objets respectant les règles de filtrage que l'utilisateur aura renseigné.

Ci-dessous nous avons le balisage HTML du formulaire pour sélectionner les types de filtres que l'on souhaite.

```html
<form action="{% url 'IOT:home_filtered' %}" method="GET" class="d-flex flex-column flex-nowrap">
    <label for="date">Date :</label>
    <input type="date" name="date" id="date">
    <label for="date_fin">Date de fin :</label>
    <input type="date" name="date_fin" id="date_fin">
    <label for="heure">Heure :</label>
    <input type="time" name="heure" id="heure" step="1">
    <label for="heure_fin">Heure de fin :</label>
    <input type="time" name="heure_fin" id="heure_fin" step="1">
    <label for="temp">Temperature (min/max) : </label>
    <input type="number" name="temp" id="temp">
    <input type="number" name="temp_max" id="temp_max">
    <button type="submit">Filter</button>
</form>
```

5. Impression des données d'une des tables :

Il y a dans le header de notre site une section permettant l'impression des données dans un format .csv afin de pouvoir visualiser les données dans une feuille de calcul.

Tout commence par un menu déroulant qui permet à l'utilisateur de sélectionner laquelle des tables l'on souhaite télécharger. Ce menu est composé de liens qui chacun renvoi vers le même contrôleur avec simplement un paramètre qui permet au contrôleur de déterminer quelle table on souhaite télécharger.

```
<a href="{% url 'IOT:save_LR' 'LR' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print living room data</a>
<a href="{% url 'IOT:save_LR' 'KT' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print Kitchen data</a>
<a href="{% url 'IOT:save_LR' 'BT' %}" class="btn btn-secondary dropped_button google_custom_font_light">Print bathroom data</a>
```

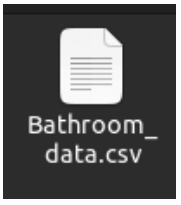
Lorsque l'un des trois boutons est pressé, le contrôleur commence par déterminer quelle table il faut imprimer par rapport au paramètre fourni à l'aide d'une déclaration 'if, else' basique.
On sélectionne ensuite la liste de tous les objets de la table sélectionnée que l'on stocke dans une variable, on prépare ensuite une autre variable 'response', qui se chargera de lancer le téléchargement du fichier généré.

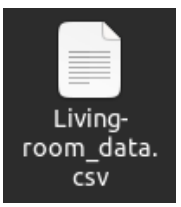
On utilise ensuite la librairie python 'csv', qu'il est impératif d'importer dans votre script, pour créer la première ligne du tableau qui contiendra la description de chaque colonne.
On prend ensuite l'objet contenant toutes les données de la table et à l'aide d'une boucle 'for' on ajoute chacun des items de la table en tant que ligne au sein du fichier que l'on crée.

Il ne reste ensuite plus qu'à retourner la réponse et cela poussera le navigateur à télécharger ce dernier.
Voici ci-dessous le contrôleur en question et le résultat de ces fichiers générés.

```python
def save_LR(request, param):
    if param == 'LR':
            salon_data = Salon.objects.all()
            response = HttpResponse(content_type='text/csv')
            response['Content-Disposition'] = 'attachment; filename="Living-room_data.csv"'
            csv_writer = csv.writer(response)
            csv_writer.writerow(['Temperature', 'Humidity', 'Pressure', 'Date', 'Time'])
            for lines in salon_data:
                csv_writer.writerow([lines.humidite, lines.temperature, lines.pression, lines.date, lines.heure])
    elif param == 'KT':
            cuisine_data = Cuisine.objects.all()
            response = HttpResponse(content_type='text/csv')
            response['Content-Disposition'] = 'attachment; filename="Kitchen_data.csv"'
            csv_writer = csv.writer(response)
            csv_writer.writerow(['Temperature', 'Humidity', 'Pressure', 'Date', 'Time'])
            for lines in cuisine_data:
                csv_writer.writerow([lines.humidite, lines.temperature, lines.pression, lines.date, lines.heure])
    elif param == 'BT':
            sdb_data = Cuisine.objects.all()
            response = HttpResponse(content_type='text/csv')
            response['Content-Disposition'] = 'attachment; filename="Bathroom_data.csv"'
            csv_writer = csv.writer(response)
            csv_writer.writerow(['Temperature', 'Humidity', 'Pressure', 'Date', 'Time'])
            for lines in sdb_data:
                csv_writer.writerow([lines.humidite, lines.temperature, lines.pression, lines.date, lines.heure])
    else:
        print('Incoherent parameter')

    return response
```



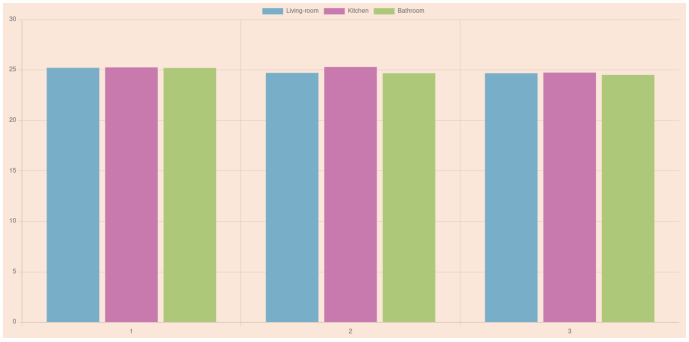


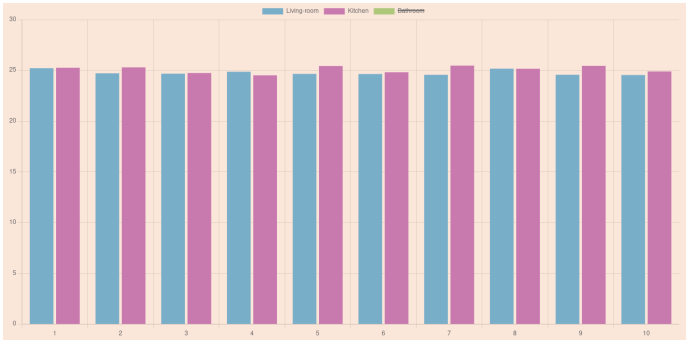
6. Affichage des données de manière graphique :

Toujours dans la section header de notre site, il y a un menu déroulant permettant l'affichage des données dans une forme plus digeste que des données brutes.
Pour des raisons de visibilité nous avons limité les options à l'affichage des 3 dernières données les plus récentes, les 10 dernières données les plus récentes et les 50 dernières données les plus récentes de chaque table.

La réalisation des graphiques en soi a été possible via la librairie Charts.js disponible gratuitement et avec une doc très complète et un système plutôt simple d'utilisation.

Lorsque l'utilisateur sélectionne l'un des trois affichages il obtient le résultat ci-dessus, respectivement avec 3,10 et 50 samples de données :

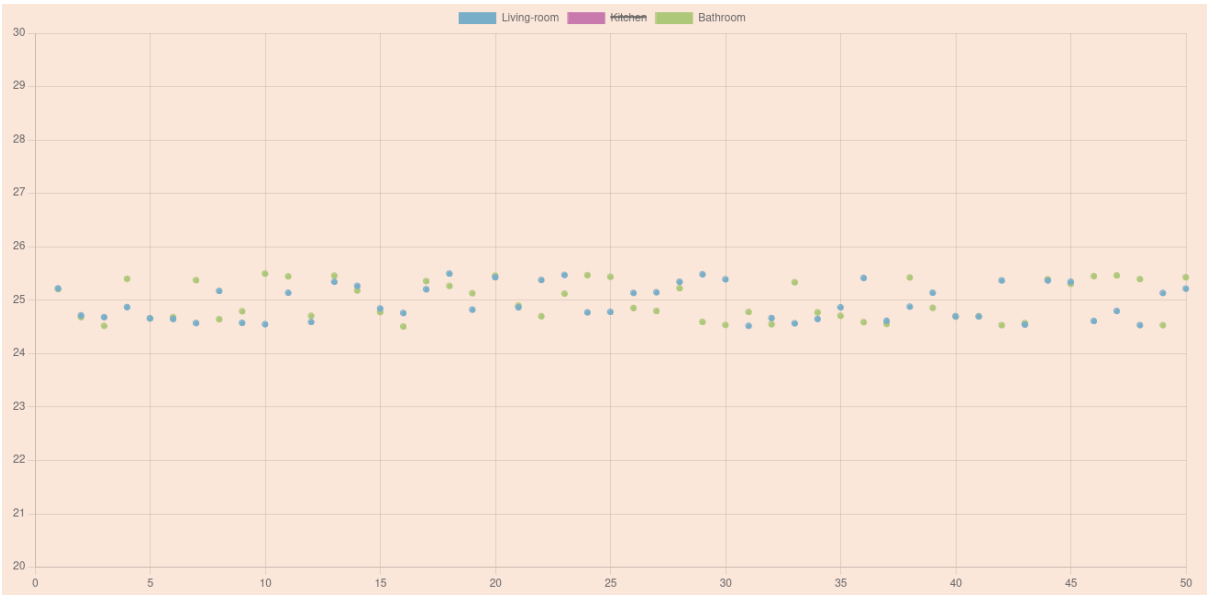




Quelques notes :

Comme vous le voyez, lorsque l'on dépasse les 10 entrées, le type de graph change pour passer à un nuage de point, en effet les graphs à colonne deviennent bien trop illisibles dans le cas échéant.

Également, il est possible de masquer les éléments de l'une des tables en cliquant sur la légende de celle-ci (voir ci-contre ou la table cuisine est masquée).



Voici ci-dessus la liste des liens menant à leur contrôleur responsable de l'affichage des graphiques :

```
<a href="{% url 'IOT:graph_3' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 3 entries</a>
<a href="{% url 'IOT:graph_10' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 10 entries</a>
<a href="{% url 'IOT:graph_50' %}" class="btn btn-secondary dropped_button google_custom_font_light">Last 50 entries</a>
```

Ceux ci nous envoie vers les contrôleurs ci-dessous :

```
def graph_3(request):
    donnees_salon_3_derniers = Salon.objects.order_by('-date', '-heure')[:3]
    donnees_cuisine_3_derniers = Cuisine.objects.order_by('-date', '-heure')[:3]
    donnees_sdb_3_derniers = SDB.objects.order_by('-date', '-heure')[:3]

    context = {
        'donnees_salon_3_derniers':donnees_salon_3_derniers,
        'donnees_cuisine_3_derniers':donnees_cuisine_3_derniers,
        'donnees_sdb_3_derniers':donnees_sdb_3_derniers,
    }
    return render(request, 'graph_3.html', context)


def graph_10(request):
    donnees_salon_10_derniers = Salon.objects.order_by('-date', '-heure')[:10]
    donnees_cuisine_10_derniers = Cuisine.objects.order_by('-date', '-heure')[:10]
    donnees_sdb_10_derniers = SDB.objects.order_by('-date', '-heure')[:10]

    context = {
        'donnees_salon_10_derniers':donnees_salon_10_derniers,
        'donnees_cuisine_10_derniers':donnees_cuisine_10_derniers,
        'donnees_sdb_10_derniers':donnees_sdb_10_derniers,
    }
    return render(request, 'graph_10.html', context)


def graph_50(request):
    donnees_salon_50_derniers = Salon.objects.order_by('-date', '-heure')[:50]
    donnees_cuisine_50_derniers = Cuisine.objects.order_by('-date', '-heure')[:50]
    donnees_sdb_50_derniers = SDB.objects.order_by('-date', '-heure')[:50]

    context = {
        'donnees_salon_50_derniers':donnees_salon_50_derniers,
        'donnees_cuisine_50_derniers':donnees_cuisine_50_derniers,
        'donnees_sdb_50_derniers':donnees_sdb_50_derniers,
    }
    return render(request, 'graph_50.html', context)
```

Chaque graph à donc son contrôleur associé et se contente tout d'abord de prendre les données de chaque table dont il aura besoin pour créer le graph, puis de les passer en contexte à la page 'graph_X.html'.

Une fois sur la dite page, un <canvas> est présent pour accueillir le graph qui va être généré.

Pour le générer, on utilise un script par page dont vous pouvez voir des exemples ci-dessous.

```
<script>
    let ctx = document.getElementById("chart").getContext("2d");

    let chart = new Chart(ctx, {
    type: "bar",
    data: {
        labels: ["1","2","3"],
        datasets: [
            {
            label: "Living-room",
            backgroundColor: "#79AEC8",
            borderColor: "#417690",
            data: [{% for donnees in donnees_salon_3_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
            {
            label: "Kitchen",
            backgroundColor: "#c879ae",
            borderColor: "#417690",
            data: [{% for donnees in donnees_cuisine_3_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
            {
            label: "Bathroom",
            backgroundColor: "#aec879",
            borderColor: "#417690",
            data: [{% for donnees in donnees_sdb_3_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
        ]
    },
    });
</script>
```

```
<script>
    let ctx = document.getElementById("chart").getContext("2d");

    let chart = new Chart(ctx, {
    type: "bubble",
    data: {
        labels: Array.from({length: 50}, (_, index) => (index + 1).toString()),
        datasets: [
            {
            label: "Living-room",
            backgroundColor: "#79AEC8",
            borderColor: "#79AEC8",
            data: [{% for donnees in donnees_salon_50_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
            {
            label: "Kitchen",
            backgroundColor: "#c879ae",
            borderColor: "#c879ae",
            data: [{% for donnees in donnees_cuisine_50_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
            {
            label: "Bathroom",
            backgroundColor: "#aec879",
            borderColor: "#aec879",
            data: [{% for donnees in donnees_sdb_50_derniers %}{{ donnees.humidite }},{% endfor %}]
            },
        ]
    },
    options: {
        scales: {
            y: {
                // beginAtZero: true
                suggestedMin: 20,
                suggestedMax: 30,
            }
        }
    }
    });
</script>
```

On renseigne dans le 'type' de graph 'bar' ou 'bubble' suivant le résultat que l'on souhaite, il ne reste ensuite qu'à renseigner les labels en abscisses et d'y joindre les données transmise par le contrôleur par le biais d'une boucle toute simple comme on l'a utilisé d'autres fois dans le site.

Il est ensuite possible de gérer ou d'ajouter des styles au graphs en fonctions de ce que l'on souhaite et il est fortement conseillé de se fier à la doc de Chart.js notamment pour les options esthétiques et d'échelles de taille au niveau des graphs.

FIN

Merci pour votre attention