SAÉ24: Web

Groupe 13

Louis DESVERNOIS, Alexis SCHOENN, Philippe DUBOIS

25 juin 2022

Table des matières

Table des figures

Table des codes

1 Introduction

Nous avons créé un site web dynamique avec le framework Django pour afficher les données récupérées par le script de collecte MQTT. Notre site doit être capable d'afficher les données avec plusieurs filtres et nous devons être capables de modifier le nom et l'emplacement de chaque capteurs.

2 Base de données

2.1 Mise en place du serveur MySQL

Nous avons utilisé MySQL Workbench pour créer le serveur ainsi que pour le configurer. Workbench n'est qu'une interface graphique à MySQL, mais n'est pas nécessaire une fois le serveur configuré. Pour commencer il faut installer le serveur MySQL sur notre machine Windows en le téléchargeant sur le site officiel de MySQL, le programme est activé automatiquement. Une fois le serveur installé et activé nous devons nous connecter avec Workbench.

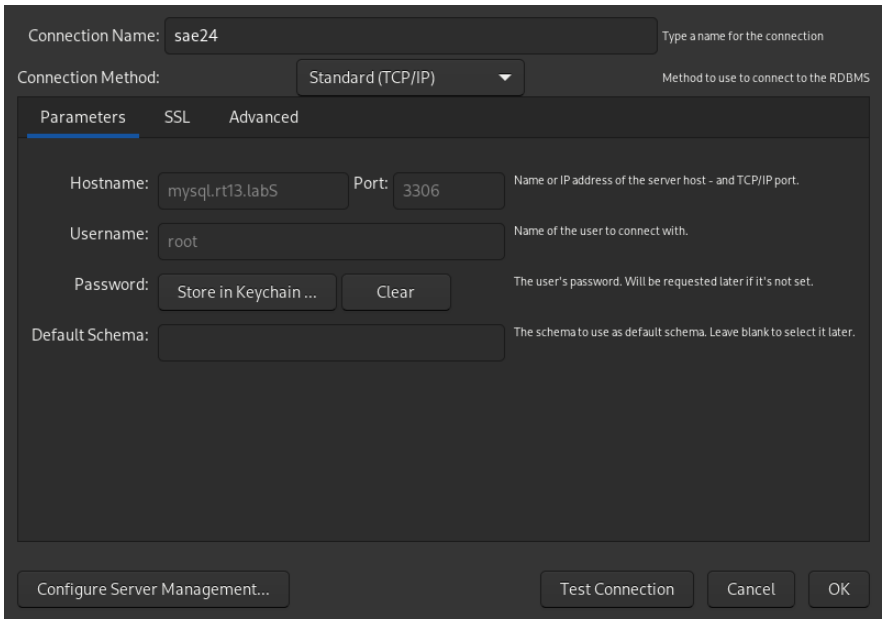

FIGURE 1 – Connexion à la BDD avec Workbench

Dans la Figure 1, le serveur MySQL est déjà configuré pour accepter les connexions extérieures, pour activer cela, il faut naviguer dans le menu *Server* puis *Users and Privileges* et configurer le paramètre *Limit to Host Matching* pour le bon utilisateur. En Figure 2 nous avons configuré l'accès au VLAN server uniquement avec le *wildcard* `%`.

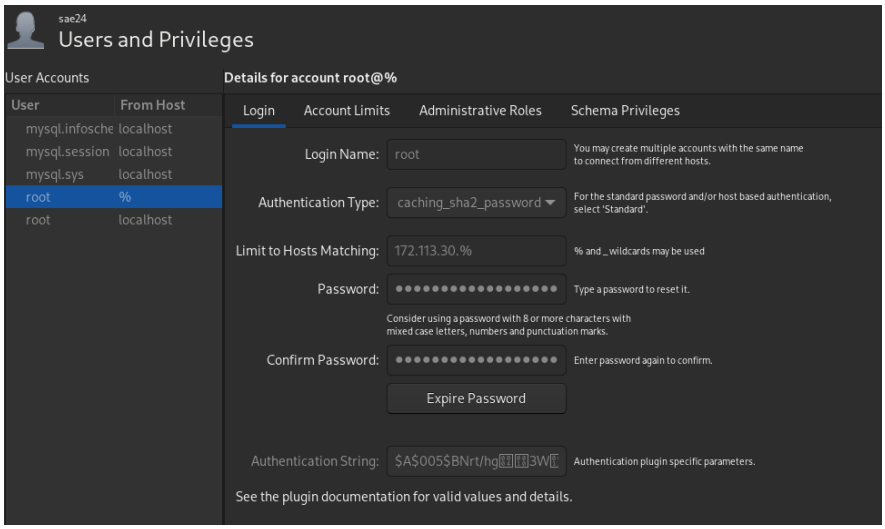

FIGURE 2 – Paramétrage de l'accès à distance

2.2 Création de la base de données et des tables

La base de données ainsi que les tables sont créés dans notre script de collecte MQTT avec des requêtes SQL.

```
CREATE DATABASE temp;
USE temp;

CREATE TABLE IF NOT EXISTS temp.sensors (
    id INT NOT NULL AUTO_INCREMENT,
    macaddr VARCHAR(12) NOT NULL,
    piece VARCHAR(50) NOT NULL,
    emplacement VARCHAR(50),
    nom VARCHAR(50),
    UNIQUE (macaddr),
    PRIMARY KEY (id));

CREATE TABLE IF NOT EXISTS temp.sensors_data (
    id INT NOT NULL AUTO_INCREMENT,
    sensor_id INT NOT NULL,
    CONSTRAINT sensorFK
        FOREIGN KEY (sensor_id)
        REFERENCES temp.sensors(id),
    datetime DATETIME NOT NULL,
    temp FLOAT NOT NULL,
    PRIMARY KEY (id));
```

Code 1 – Création de la base de données et des tables

3 Création projet Django

3.1 Initialisation

Pour commencer, nous avons besoin d'initialiser notre projet Django. Pour cela nous allons d'abord créer un environnement virtuel Python avec la commande `python3 -m venv .venv` nous pouvons ensuite l'activer avec `source .venv/bin/activate`. Une fois dans l'environnement virtuel, nous pouvons utiliser `pip install` pour installer les paquets dont nous avons besoin, c'est-à-dire django, django-admin et mysqlclient [1]. Avec tous les paquets installer, nous pouvons, exécuter les commandes `django-admin startprojet sae24` pour initialiser le projet et `django-admin startapp temp` pour créer l'application.

3.2 Connexion de Django à la base de données

Pour connecter Django à notre de base de données, nous devons modifier le fichier `settings.py` du projet.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'temp',
        'USER': 'root',
        'PASSWORD': 'admin',
        'HOST': 'mysql.rt13.lab',
        'PORT': '3306',
    }
}
```

Code 2 – settings.py : connexion à la BDD

Maintenant que nous sommes connectés à la base de données nous pouvons créer automatiquement le fichier `models.py` en utilisant la commande `./manage.py inspectdb > temp/models.py`.

3.3 Création du form

Nous pouvons maintenant créer le form pour les capteurs, qui nous permet de modifier le nom et l'emplacement du capteurs.

```
class SensorsForm(ModelForm):
    class Meta:
        model = models.Sensors
        fields = ('emplacement', 'nom')
        labels = {
            'nom': 'Nom',
            'emplacement': 'Emplacement'
        }
```

Code 3 – Formulaire pour les capteur

En code 3 nous n'ajoutons pas les fields qui ne doivent pas changer dans la base de données.

1. Nous avons également besoin d'installer le paquet mariadb-clients avec apt pour que l'installation fonctionne

3.4 Création des views

Maintenant que notre formulaire est créé, nous pouvons nous atteler à la création des views de notre projet.

3.4.1 Affichage

Les views d'affichage sont les plus simples à créer, en effet, il suffit de récupérer tous les objets d'un models et de les envoyer dans les templates HTML.

```
global refresh
global refresh_time
refresh = True
refresh_time = 20

def liste_sensors(request):
    sensors = Sensors.objects.all()
    return render(request, 'sensors/liste.html', {'sensors': sensors})

def liste_data(request):
    data = SensorsData.objects.all()
    return render(request, 'data/liste.html', {'data': data, 'refresh': refresh,
    ↪  'refresh_time': refresh_time})
```

Code 4 – Views d'affichage

Les variables `refresh` et `refresh_time` permettent d'activer ou de désactiver le rafraîchissement automatiquement. Les deux templates appelées ici sont quasiment identique, les seules différences sont le nombre de colonnes dans le tableau.

3.4.2 Modification

La modification des capteur se fait avec deux views, un de modification qui affiche le formulaire (code 5) et un autre qui enregistre les modifications dans la base de données (code 6).

```
def modif_sensors(request, id):
    obj = Sensors.objects.get(id=id)
    objform = SensorsForm(model_to_dict(obj))
    if request.method == "POST":
        form = SensorsForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect("/sensor/liste")
    else:
        return render(request, "sensors/modif.html", {"form": objform, "id": id})
```

Code 5 – Views de Modification 1

```python
def save_modif_sensors(request, id):
    objform = SensorsForm(request.POST)
    bak = Sensors.objects.get(id=id)
    sensors = Sensors.objects.all()
    if objform.is_valid():
        objform = objform.save(commit=False)
        objform.id = id
        objform.macaddr = bak.macaddr
        objform.piece = bak.piece
        for i in sensors:
            if i.nom == objform.nom:
                return HttpResponseRedirect(f"/sensors/modif/{id}")
        objform.save()
        return HttpResponseRedirect("/sensors/liste")
    else:
        return render(request, "sensors/modif.html", {"form": objform, "id": id})
```

Code 6 – Views de Modification 2

En code 6, nous créons une copie du capteur que nous souhaitons modifier sans cela, notre bouton modifier supprimerais l'adresse MAC et la pièce du capteur modifié, car notre form ne contient pas ces informations. Nous vérifions également si le nom choisi n'est pas déjà utilisé par un autre capteur, si cela est le cas, nous ne sauvegardons pas la modification.

3.4.3 Filtre

Pour créer des filtres facilement, nous pouvons utiliser la fonction `filter()` des objets de model Django. Le filtre le plus simple est celui en code 7, qui permet de filtrer les données par capteur.

```python
def filtre_par_sensor(request, id):
    data = SensorsData.objects.filter(sensor = id)
    return render(request, 'data/liste.html', {'data': data})
```

Code 7 – Filtre par capteur

Nous pouvons cependant créer d'autres filtres plus complexes, par exemple, nous pouvons filtrer entre deux dates utilisant `__range` et des objets `datetime`. La création d'une template contenant un `<form>` HTML est nécessaire.

```html
<form method="post" action="/data/liste_filtre">
    {% csrf_token %}
    <input type="datetime-local" class="form-control" id="startDate" name="startDate"
    ↪  placeholder="Date de début">
    <input type="datetime-local" class="form-control" id="endDate" name="endDate"
    ↪  placeholder="Date de fin">
    <input type="submit" class="btn btn-success" value="Valider">
</form>
```

Code 8 – Template de filtre entre deux dates et heures

```python
def filtre_par_date(request):
    start = datetime.fromisoformat(request.POST["startDate"])
    end = datetime.fromisoformat(request.POST["endDate"])
    data = SensorsData.objects.filter(datetime__range=(start, end))
    return render(request, 'data/liste.html', {'data': data, 'refresh': False})
```

Code 9 – Filtre entre deux dates et heures

Nous pouvons également utiliser ces deux filtres en même temps en mettant en argument les deux conditions.

```python
def filtre_par_date_et_capteur(request, id):
    start = datetime.fromisoformat(request.POST["startDate"])
    end = datetime.fromisoformat(request.POST["endDate"])
    data = SensorsData.objects.filter(datetime__range=(start, end), sensor = id)
    return render(request, 'data/liste.html', {'data': data, 'refresh': False})
```

Code 10 – Filtre par date et par capteur

3.4.4 Exportation en CSV

Maintenant que nous pouvons lister et filter nos capteurs et données, nous pouvons utiliser le module `csv` mis à disposition par Python pour facilement créer dynamiquement des fichiers de valeurs séparées par des virgules.

```python
def export_csv(request, id):
    data = SensorsData.objects.filter(sensor = id)
    response = HttpResponse(
        content_type='text/csv',
        headers={'Content-Disposition': 'attachment; filename="export.csv"'},
    )

    writer = csv.writer(response)
    writer.writerow(['timestamp', 'macaddr', 'piece', 'emplacement', 'nom', 'temp'])
    for i in data:
        writer.writerow(
            [i.datetime,
            i.sensor.macaddr,
            i.sensor.piece,
            i.sensor.emplacement,
            i.sensor.nom,
            i.temp])

    return response
```

Code 11 – Génération d'un fichier csv

En code 11, nous utilisons une boucle for pour générer ce fichier ligne par ligne avec la méthode `writerow()` de l'objet `writer` du module importé.

3.4.5 Templates

Les templates sont à écrire en même temps ou même avant les views. Elles utilisent des boucles for pour permettre à Django de créer des pages dynamiquement. Voici un exemple simplifié de template utilisant un modèle avec `{% extends MODELE %}` pour simplifier la création de la mise en page.

```
{% block content %}
<table class="table is-fullwidth is-hoverable">
    <thead>
    <tr>
        <!-- Titres colonnes avec des <td>-->
    </tr>
    </thead>
    {% for temps in data %}
    <tbody>
    <tr>
        <!-- Données avec des <td> et en utiliant temps.VAL -->
    </tr>
    </tbody>
    {% endfor %}
</table>
{% endblock content %}
```

Code 12 – Exemple de template de liste

4 Déploiement

Pour le déploiement, nous avons utilisé un serveur Debian 11 sans interface graphique avec le serveur web Nginx ainsi que Gunicorn. Il n'est impossible de directement utiliser Nginx pour distribuer les pages de notre site, dans notre cas, nous allons le configurer comme un *reverse proxy* à Gunicorn. **Pour la partie qui suit, nous avons cloné le dépôt git de notre projet dans `/home/toto/CLONE`.** Avant de déployer nous devons ajouter `STATIC_ROOT = os.path.join(BASE_DIR, 'static')` au fichier `settings.py`.

4.1 Gunicorn

Pour commencer, nous devons créer deux fichiers pour interfacer systemd à Gunicorn, `gunicorn.socket` et `gunicorn.service`.

```
[Unit]
Description=gunicorn socket
[Socket]
ListenStream=/run/gunicorn.sock
[Install]
WantedBy=sockets.target
```

Code 13 – /etc/systemd/system/gunicorn.socket

Ce fichier permet de créer un *socket* (ou une *interface de connexion*), qui est, pour faire simple, un moyen que plusieurs applications peuvent utiliser pour communiquer. Nous allons utiliser cela pour connecter Gunicorn à Nginx.

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target
[Service]
User=toto
Group=www-data
WorkingDirectory=/home/toto/CLONE/django/SAE24
ExecStart=/home/toto/CLONE/django/SAE24/.venv/bin/gunicorn --access-logfile - --workers 3
↪  --bind unix:/run/gunicorn.sock SAE24.wsgi:application
[Install]
WantedBy=multi-user.target
```

Code 14 – /etc/systemd/system/gunicorn.service

La création du service nous permet de lancer Gunicorn sous la forme d'un *daemon*. Dans ce fichier, nous précisons l'utilisateur et le groupe avec lesquels le service doit être exécuté ainsi que le service doit être exécuté après l'initialisation du réseau. Nous pouvons maintenant activer Gunicorn au démarrage avec les commandes `systemctl start gunicorn.socket` et `systemctl enable gunicorn.socket`.

4.2 Nginx

Une fois Gunicorn correctement paramétré, nous pouvons configurer Nginx pour rediriger les connexions sur le socket. Pour cela il faut créer un fichier *site* en suivant la bonne syntaxe.

```
server {
    listen 80;
    server_name django-serv.rt13.lab;
    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/toto/CLONE/django/SAE24;
    }
    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

Code 15 – /etc/nginx/sites-available/sae24

Pour cet exemple, Nginx écoute sur le port 80 et si un client se connecte en utilisant le domaine `django-serv.rt13.lab`, il redirige /static vers le répertoire configuré dans, le reste est redirigé vers le socket Gunicorn. Une fois ce fichier crée, nous devons l'activer avec un lien symbolique créé par la commande `ln -s /etc/nginx/sites-available/sae23 /etc/nginx/sites-enabled/`. Nous devons ensuite utiliser `manage.py` pour faire un `makemigration`, un `migrate` ainsi qu'un `collectstatic`.

Pour finir, nous pouvons redémarrer le service Nginx avec la commande `systemctl restart nginx` pour prendre en compte les changements.

Nous avons également écrit un script exécutant automatiquement toutes les étapes du déploiement, il est donc possible d'utiliser `curl` pour l'exécuter dans une machine virtuelle sans interface graphique.