Conception et développement d'un système de supervision de la consommation énergétique

Adrien Larousse
Etudiant en ingénieurie informatique
2A ISS
Télécom Nancy
Villers-lès-Nancy, 54600
Email: adrien.larousse@telecomnancy.eu

Stanislas Mezureux
Etudiant en ingénieurie informatique
2A ISS
Télécom Nancy
Villers-lès-Nancy, 54600
Email: stanislas.mezureux@telecomnancy.eu

***Abstract*—Cet article présente un projet de collaboration entre la société de génie électrique CEGELEC Lorraine, l'ENSEM et TELECOM Nancy, visant à développer un système de supervision de la consommation énergétique pour les bâtiments universitaires. L'objectif principal du projet est de sensibiliser les personnels et les étudiants à la transition énergétique et aux écogestes en affichant en temps réel les consommations électriques, la température extérieure, l'hygrométrie, la consommation due au chauffage et l'impact carbone des différentes activités dans le hall de l'école. Le système de supervision repose sur l'utilisation de capteurs sans fil pour mesurer les données, d'une base de données pour les stocker, ainsi que sur une interface utilisateur conviviale pour visualiser et analyser les informations. La solution a été conçue pour être facilement dupliquée dans les bâtiments d'autres écoles et/ou d'autres secteurs d'activité.** [1]

Mots-clés : Développement durable, Capteurs communicants (I.O.T.), Stockage de données horodatées, Application Web, Supervision, Consommation énergétique

TABLE DES MATIÈRES

[1] https://github.com/PIDR-EnergyWatch/EnergyWatch

I. INTRODUCTION

Le changement climatique est établi comme étant l'un des grands enjeux du 21ème siècle. Pour limiter les émissions de gaz à effet de serre, il est nécessaire de réduire sa consommation énergétique. Pour cela, il faut observer et comprendre les sources de notre dépense énergétique afin de trouver des leviers. Quantifier nos

gestes et les ressources que demandent les équipements est une première étape dans la sensibilisation aux enjeux de réduction de l'empreinte carbone.

A. Contexte

En s'inscrivant dans ce contexte, les étudiants de l'ENSEM cherchent à entreprendre leur démarche de sensibilisation à la consommation énergétique de leur école. Pour cela, un premier groupe d'étudiants a été chargé d'installer des capteurs afin de récupérer des données de consommation sur leurs équipements et un autre groupe d'étudiants de proposer une manière d'afficher des données pertinentes.

En plus des groupes de travail présents à l'ENSEM, ce projet d'outil de supervision de consommation énergétique est en collaboration avec l'Université de Lorraine et l'entreprise de génie électrique CEGELEC afin de récupérer un plus large volume de données.

B. Objectifs

Dans le cadre de ce projet, les étudiants de Télécom Nancy ont été chargés de concevoir et de développer un système de supervision de la consommation énergétique, en s'appuyant sur les travaux préalablement réalisés par les étudiants de l'ENSEM. L'objectif principal de ce projet est de fournir aux élèves de l'ENSEM un tableau de bord virtuel, leur permettant de consulter facilement la consommption électrique de leur établissement. Toutefois, les ambitions de ce projet ne s'arrêtent pas là : il est également demandé aux étudiants de Télécom Nancy de concevoir un outil réutilisable, afin de faciliter le déploiement de cette solution dans d'autres établissements.

C. Étapes clés de la réalisation

Pour la conception et le développement de cet outil de supervision, trois tâches principales sont à mettre en œuvre :

1) Instancier le serveur récoltant et stockant les données issues des différents capteurs communicants dans une base de données adaptée ;
2) Extraire les informations pertinentes ;
3) Réaliser l'interface de visualisation des données sous la forme d'un tableau de bord.

II. Analyse de l'existant

La construction de cet outil se base sur des choix techniques réalisés en amont par les groupes de travail de l'ENSEM. L'objectif de leur travail était de mettre en place les capteurs afin d'envoyer les données sur un serveur.

A. Sources de données

La récolte des données est gérée en amont par le travail des étudiants, notre outil devra uniquement récupérer les données envoyées, les stocker et les afficher de manière pertinente. Pour cela les étudiants de l'ENSEM disposent de plusieurs sources de données. On note les suivantes :

1) Capteurs présents dans les salles: Ce sont les capteurs installés par les étudiants de l'ENSEM, on en note trois :

- un capteur de température présent dans une salle ;
- un capteur sur la consommation d'un ordinateur ;
- un capteur sur la puissance d'un écran d'ordinateur.

Ces capteurs sont connectés à un `EmonPi`, il sert à collecter les données et à les traiter.

2) Consommation des bâtiments fournie par l'Université de Lorraine: Pour ce projet, seules des données statiques envoyées par l'Université de Lorraine seront disponibles, à terme l'objectif est d'avoir ces données en temps réel à l'aide d'une interface.

3) La station météo de l'ENSEM: Elle a été installée sur le toit de l'école et prend des valeurs en temps réel comme : la température, l'humidité, l'indice UV...

4) API Externes:

- Réseau de Transport d'Électricité (RTE) afin d'obtenir le mix énergétique (national et régional) en temps réel
- *OpenWeatherMap* afin d'avoir les prévisions météo

B. Envoi des données

Les données récupérées par les étudiants de l'ENSEM passent d'abord par un serveur `Node-RED`. Un serveur `Node-RED` est un outil qui permet de connecter des appareils physiques, des API et des services en ligne. Il permet ici d'envoyer les données sur le serveur MQTT.

Le serveur MQTT: Le serveur ou *broker* MQTT est l'un des éléments centraux de ce projet, il est le pont entre le travail des étudiants de l'ENSEM et celui des étudiants de TELECOM Nancy.

MQTT est un protocole de messagerie de type *publish and subscribe* basé sur le protocole *TCP/IP*, le principe est simple, un capteur envoie des données en continu sur un *topic* du *broker MQTT*. Un utilisateur s'abonne au *topic* en question et il reçoit par l'intermédiaire du

broker MQTT les données en temps réel. La figure 1 résume ce fonctionnement.

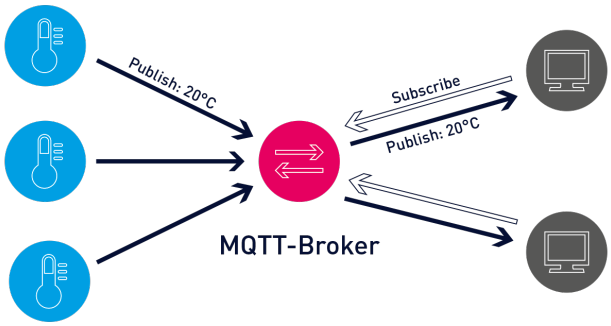

Fig. 1. Fonctionnement d'un *broker MQTT*[2]

Le *broker-MQTT* est le dernier maillon de la chaîne avant de devoir stocker et afficher les données. On peut donc résumer l'existant à l'aide de la figure 2.

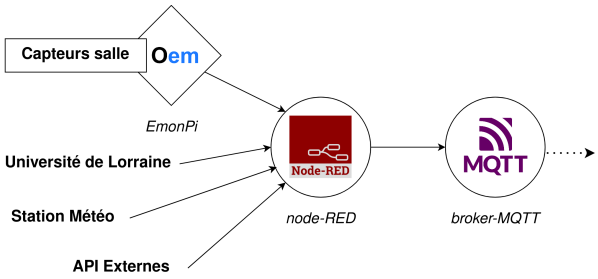

Fig. 2. Architecture initiale mise en place par l'ENSEM

Les étudiants de l'ENSEM ont utilisé un serveur public pour soumettre les données, il est accessible à l'adresse suivante :

`mqtt://broker.hivemq.com:1883`

À l'aide de ce serveur public, on peut s'abonner à différents *topics* paramétrés par les étudiants et alors récupérer des données qui proviennent des capteurs de l'ENSEM.

Pour la mise en place d'un *broker MQTT*, il est recommandé d'utiliser *Mosquitto*[3]. En effet, *Mosquitto* est le serveur MQTT le plus répandu. Il s'agit d'un logiciel open-source qui permet l'utilisation du protocole MQTT entre différents appareils.

C. Maquette de tableau de bord

Dans leurs travaux, les étudiants de l'ENSEM ont produit des maquettes de tableau de bord (Figure 3) afin de donner une inspiration sur la manière d'afficher les données. On retrouve alors les sources évoquées précédemment affichées sous différentes formes : graphiques, jauges, camemberts...

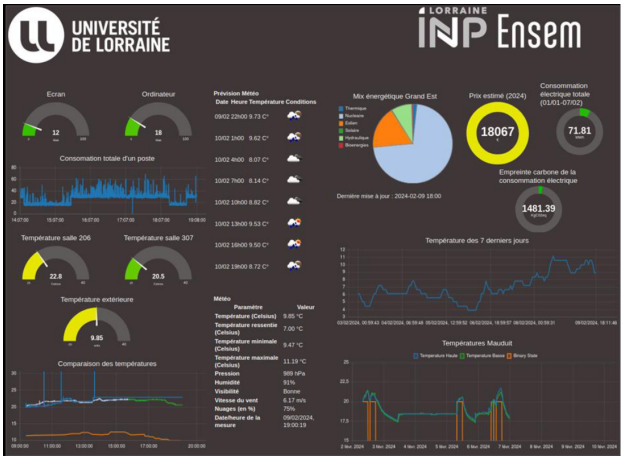

Fig. 3. Tableau de bord existant

III. Socle technique

À partir des choix réalisées en amont de la conception de l'outil de supervision, de nouveaux choix techniques s'imposent sur la manière de stocker les données ainsi que la manière de réaliser l'interface du tableau de bord.

A. Systèmes équivalents

Tout d'abord, il est intéressant de regarder d'autres outils de supervision de la consommation énergétique pour savoir ce qu'il est possible de réaliser. Voici une liste non exhaustive de systèmes équivalents.

- Wave (par Cegelec) : supervision de la température, de la consommation électrique, ... mais aussi gestion (*e.g.* temporisation des éclairages, pas de recharge des véhicules électriques si la production des panneaux ne dépasse pas un certain seuil.
- éco2mix (par RTE)[4] : répartition de la production d'électricité par filière et par région
- Université de Lorraine : supervision température, consommation, ventilation, ...

Ces systèmes ne sont pas adaptés à un affichage convivial de données. En effet, ils sont trop spécifiques et pas adaptés à notre besoin d'affichage. Par exemple :

- l'interface de l'Université de Lorraine présente une multitude d'interfaces avec des informations complexes en temps réel qui ne sont pas compréhensible au moindre coup d'œil.
- le site éco$_2$mix ne permet d'afficher des données sur une seul page, il n'est pas adapté à l'affichage sur une télévision.

B. Stockage des données

L'un des problèmes qui se pose dans le cas des applications d'I.O.T. est le stockage des données. En effet, les systèmes I.O.T. génèrent un grand nombre de données, lié à la multitude de capteurs et leur fréquence d'émissions ainsi qu'à l'accumulation des données dans le temps. Pour stocker les données issues des capteurs, il faut donc utiliser une base de données. Il existe 3 grandes familles de bases de données[5] :

- les bases de données relationnelles
- les bases de données `NoSQL`
- les bases de données `Time Series`

Pour décider quel type de base de données choisir, la présentation suivante présente un comparatif.

Les bases de données relationnelles :

+ fiable ;
+ permet de réaliser des opérations complexes sur les données ;
– nécessite de modéliser les données en amont ;
– implémentation des données temporelles `Time Series` non native.

Les bases de données NoSql :

+ flexible, elle ne nécessite pas la définition strictes des tables ;
+ requête plus rapide ;
– mise à jour des données lente.

Les bases de données Time Series :

+ base de données optimisée pour les données horodatées ou les séries chronologiques ;
+ les bases de données `Time Series` sont spécifiquement conçues pour une gestion efficace du cycle de vie des données, de la compression et des analyses à grande échelle de nombreux enregistrements ;
– utilisation spécifique.

Notre choix s'est logiquement porté sur une base de données Time Series pour sa gestion optimisée des données horodatées. Nous avons décidé d'utiliser ***InfluxDB***[1], qui est un système de base de données de type Time Series, car c'est un outil répandu dans le monde de l'I.O.T.[6] et il présente des avantages considérables, tels que l'automatisation de l'ajout en base de données issue d'un *broker-MQTT* à l'aide de l'agent *Telegraf*.

C. Affichage du système de supervision

Pour afficher l'outil de supervision, la solution la plus appropriée est de créer une interface web sous la forme d'un tableau de bord. Cette interface web doit être conçue pour récupérer les données de la base de données et afficher des graphiques pertinents et de complexité variable.

L'utilisation d'une interface web est logique car elle permet d'accéder facilement au tableau de bord depuis n'importe quel appareil disposant d'un navigateur web. Cela évite également la nécessité d'installer des logiciels supplémentaires sur les appareils clients.

L'application web sera développée avec `NodeJS`, c'est un environnement d'exécution `JavaScript` qui permet de développer des applications web performantes et évolutives.

Backend Le *backend* de l'application est écrit en `Go`. Ce choix s'explique par la légèreté, la rapidité et la fiabilité de ce langage moderne. Le *backend* de l'application permet de faire le lien entre la base de données et le *frontend* en interrogeant cette dernière sur sollicitation de l'utilisateur via le *frontend*. Par exemple, renvoyant des données au format `JSON` pour afficher les graphiques ou pour se connecter. De plus, l'utilisation du *framework* `Go Echo` [8] est tout indiquée pour transformer notre programme `Go` en un véritable *backend* d'application web. Ce *framework* offre une variété de fonctionnalités pour faciliter le développement d'applications web,

Frontend Le *frontend* utilise le *framework* `JavaScript` *SvelteKit* [9], qui permet de gérer directement les routes avec l'arborescence de fichiers du *frontend*. En plus de sa syntaxe légère, ce *framework* offre l'occasion de découvrir une nouvelle technologie `JavaScript`.

Pour l'affichage des graphiques, notre choix c'est porté sur la librairie `JavaScript` *ChartJS*. Elle permet de réaliser des graphiques de tous types (Figure 4) : donuts, camemberts, histogrammes, courbes…

Fig. 4. Exemple d'utilisation de *ChartJS*

D. Résumé des choix techniques pour l'outil de supervision

La structure de l'outil de supervision que nous allons mettre en place est visible sur la Figure 5

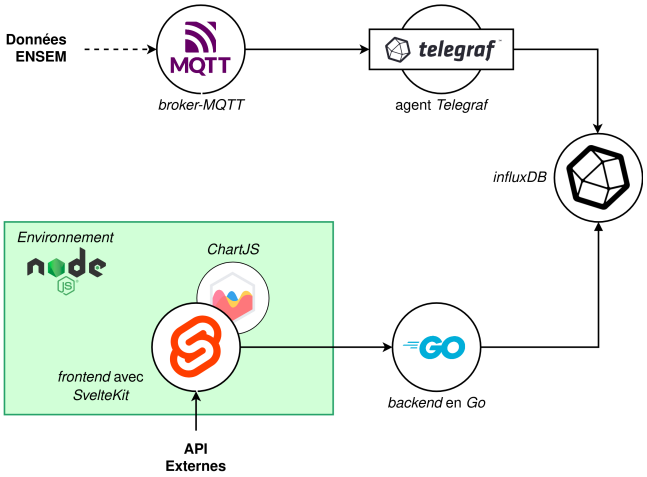

Fig. 5. Structure général de l'outil de supervision

IV. Conception et Mise en place

A. Mise en place de l'environnement

Le livrable de ce projet étant l'application toute entière, elle est amenée à être déployée mais également à être enrichie en données. Par conséquent, nous avons décidé d'utiliser *Docker* pour éviter tout problème de compatibilité pour les personnes qui souhaiteraient développer ou déployer le projet. Ainsi, la première étape de la conception a été de définir quelles images utiliser, quels ports exposer, quelles variables d'environnement considérer et quels fichiers/dossiers monter dans le conteneur.

BDD image officielle DockerHub[2], port 8086, montage des fichiers de configuration, fichier d'environnement unique.

Frontend image basée sur Node[3], port 3000, montage des fichiers sources et des fichiers de configuration, fichier d'environnement global.

Backend image officielle DockerHu[4], port 8000, montage des fichiers sources, fichier d'environnement global.

[2]https://hub.docker.com/_/influxdb
[3]https://hub.docker.com/_/node
[4]https://hub.docker.com/_/golang

Telegraf image oficielle DockerHub[5], montage du fichier de configuration, fichier d'environnement global.

B. Structure de l'application

Ce qui donne l'arborescence suivante avec une profondeur limitée à 2 :

```
/
├── .env
├── Makefile
├── backend/ ..... API
│   ├── Dockerfile.dev
│   ├── Dockerfile.prod
│   ├── api/
│   ├── bin/
│   ├── cmd/
│   ├── go.mod
│   ├── go.sum
│   └── interval/
├── db/ ..... BDD
│   ├── EAT.csv
│   └── pat.csv
├── docker/ ..... Configuration
│   └── telegraf/
├── docker-compose.dev.yml
├── docker-compose.prod.yml
└── frontend/ ..... Page web
    ├── Dockerfile.dev
    ├── Dockerfile.prod
    ├── node_modules/
    ├── package-lock.json
    ├── package.json
    ├── postcss.config.js
    ├── src/
    ├── static/
    ├── svelte.config.js
    ├── tailwind.config.js
    └── vite.config.js
```

C. Fonctionnement en composants

Pour faciliter l'intégration de nouvelles sources de données dans le tableau de bord, chaque type de graphique est conçu comme un composant réutilisable. Cette approche modulaire permet de gagner du temps et d'assurer une cohérence visuelle entre les différents graphiques. On retrouve alors :

- un composant pour les courbes : classe `Line` dans *ChartJS*

[5]https://hub.docker.com/_/telegraf

- un composant pour les mix énergétiques : classe `Line` dans *ChartJS* avec plusieurs jeux de données.
- un composant station météo.

On retrouve ces composants dans :

```
frontend/
└── src/
    └── lib/
        └── components/ ..... tous les composants
            └── graph/
                ├── eco2mix.svelte
                └── ..
```

D. Récupération, stockage et affichage des données de l'ENSEM

Pour afficher les données issues des capteurs, il nous faut dans un premier temps être capable de stocker les données publiées sur le serveur MQTT. Pour cela, on nous fournit 3 *topics* :
- `salle206/temperature` : qui envoie la température de la salle 206
- `salle206/puissance_ordonateur02` : qui envoie la puissance d'un ordinateur.
- `salle206/puissance_ecranordinateur02` : qui envoie la puissance d'un écran d'ordinateur.

A l'aide d'un agent *Telegraf*, on va pouvoir configurer un agent MQTT qui s'abonne aux différents *topics* et ajoute les données dans la base de données automatiquement.

```
[[inputs.mqtt_consumer]]

servers = ["mqtt://broker.hivemq.com
   :1883"]

## Topics that will be subscribed to.
topics = [
    "ensem/salle206/temperature",
]

data_format = "value"
data_type = "float"
```

Listing 1. Configuration de l'agent *Telegraf* pour `MQTT Consumer`

Avec cette méthode, les données sont automatiquement ajoutées à la base de données. Cependant, nous avons rapidement constaté un problème : la fréquence d'ajout est élevée, avec trois nouvelles valeurs toutes les 5 secondes. Pour réduire le nombre de valeurs et optimiser l'espace de stockage, nous avons choisi d'agréger les données en utilisant le *plugin* `aggregators.basicstats`. Ce dernier permet de calculer et d'ajouter à intervalles réguliers la valeur moyenne des données reçues sur cet intervalle.

Comme les valeurs reçues ont une faible fluctuation, nous avons décidé d'utiliser un intervalle de 1 heure.

```
[[aggregators.basicstats]]

period = "1h"         # send & clear
   the aggregate every 1h.

drop_original = true  # drop the
   original metrics.

stats = ["mean"]
```

Listing 2. Configuration de l'agent *Telegraf* pour `aggregators.basicstats`

Les données sont enregistrées sous le format suivant :

_time	_value	_field	_measurement
2024-05-22T09:36:10Z	20	value_mean	temperature

TABLE I
FORMAT DES DONNÉES EN BASE

Une fois les données stockées dans la base de données, le processus d'affichage est simple. Le *frontend* envoie une requête au *backend* pour récupérer les données d'une mesure spécifique. Le *backend* traite la requête en interrogeant la base de données pour obtenir toutes les données correspondant à cette mesure. Ensuite, le *backend* transmet le résultat au *frontend*, qui se charge de l'afficher à l'aide d'un composant.

Pour l'affichage de la température de la salle 206, on utilise le composant `LineChart`, on obtient le rendu sur la figure 6.

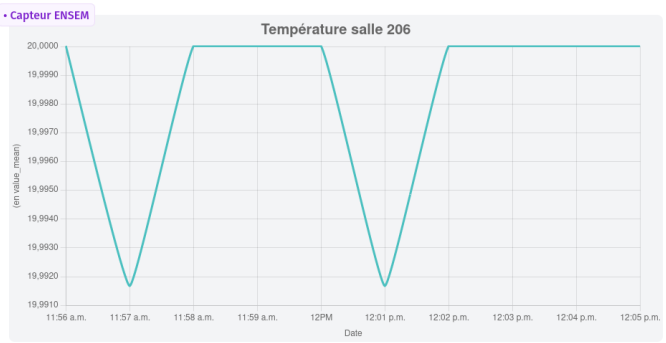

Fig. 6. Rendu de la température de la salle 206 sur le tableau de bord

Pour estimer la consommation électrique de la salle 206, nous utilisons les capteurs installés sur l'ordinateur et l'écran n°2. Ces capteurs nous permettent de mesurer la puissance consommée par l'ordinateur et l'écran en question. Nous multiplions ensuite ces valeurs par le nombre total de postes informatiques présents dans la salle, soit 18, afin d'obtenir une approximation globale de la consommation électrique de la salle. On obtient le graphique présenté sur la figure 7.

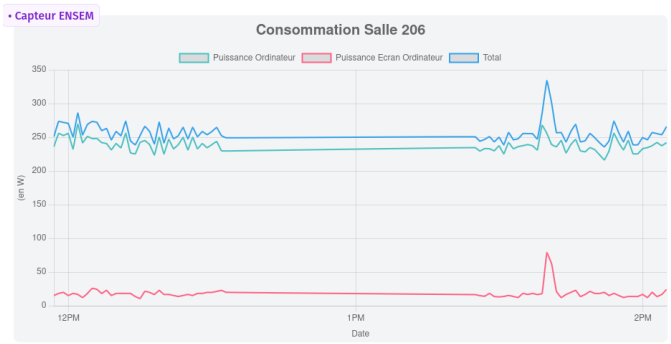

Fig. 7. Rendu de la consommation des postes informatiques de la salle 206 sur le tableau de bord

Remarque : Dans la figure 7, l'écran (courbe rouge) et en veille et l'ordinateur (courbe verte) aussi.

E. Récupération et affichage des données des API externes

1) Données de mix énergétique issue de l'API du Réseau de Transport d'Électricité: Dans la documentation fournie par l'ENSEM, le calcul du mix énergétique est fourni par une API externe associée à la structure RTE. Le mix énergétique est accessible en temps réel sur le site de RTE via un outil appelé *éco2mix* (Figure 8).

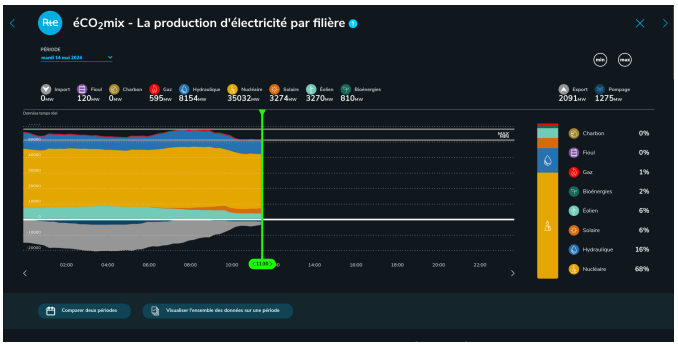

Fig. 8. éCO2mix - La production d'électricité par filière

L'API d'OpenDataSoft [7], permet de récupérer les données du mix énergétique depuis 2023. Cela veut donc dire que nous n'avons pas besoin de stocker les données dans la base de données. Le *frontend* de l'application appellera directement l'API pour afficher le mix énergétique.

Pour récupérer les données du jour, il faut adapter la requête pour que cette dernière renvoie le mix énergétique du jour.

L'appel à l'API nous renvoie une liste d'objets. Chaque objet correspond au mix énergétique pour une heure donnée. On retrouve la production en MW pour tous types de filières : nucléaire, hydraulique, solaire, éolien...

```
{"total_count": 88,
"results": [
    {
        "perimetre": "France",
        "nature": "Donnees temps reel"
    ,
        "date": "2024-05-21",
        "heure": "02:00",
        "date_heure": "2024-05-21T00:0
    0:00+00:00",
        "consommation": 36020,
        "nucleaire": 40270,
        "eolien": 2612,
        ...
    },
        ...
    {
        ...
    }
```

Listing 3. Structure du résultat de l'API *OpenDataSoft*

A l'aide du composant `eco2mix.svelte`, on peut alors afficher le mix énergétique sur notre tableau de bord (Figure 9).

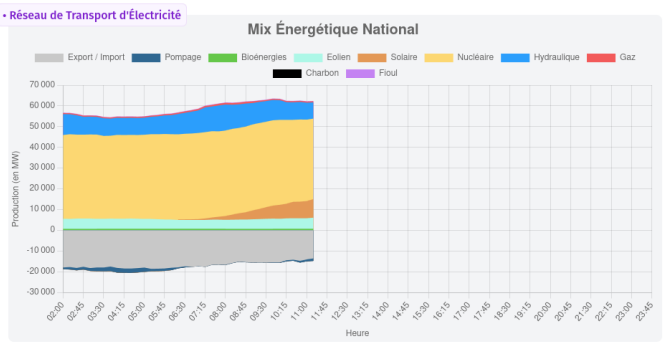

Fig. 9. Rendu du mix énergétique sur le tableau de bord

2) Composant Météo à l'aide de l'API de la station de l'ENSEM & l'API OpenWheater: Afin d'ajouter un composant pour la météo, nous avons pu nous baser sur les résultats de deux API :

- une API qui permet d'avoir les relevés météorologiques de l'ENSEM. Cette API est issue de la station météo installée à l'ENSEM.
- une API (*OpenWeather*) qui permet d'avoir des relevés et des prévisions pour la ville de Nancy.

L'API de la station météo renvoie une liste de données qui correspond aux relevés météorologiques heure par heure des sept derniers jours. Elle permet de consulter : la température, l'humidité, la vitesse du vent, l'indice UV…

On remarque que les valeurs renvoyées par l'API sont en unité impériale, il faut donc les convertir :

$$Temp_C = \frac{(Temp_F - 32) \times 5}{9}$$

On réalise ce traitement directement dans le *frontend*, le résultat de l'API présente un résultat limité et donc facilement traitable.

Ensuite, afin d'obtenir les prévisions, on utilise l'API *OpenWeather*. Elle est moins précise car ce n'est pas une station prévue spécialement pour l'école, mais elle permet d'avoir une tendance car c'est une prévision pour la localité. Le composant est visible sur la figure 10.

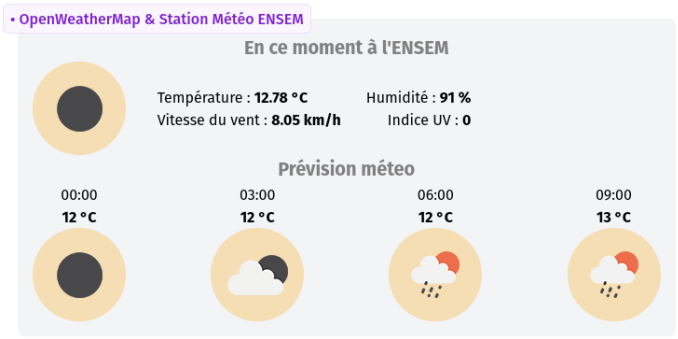

Fig. 10. Rendu du composant météo sur le tableau de bord

F. Intégration des données de l'Université de Lorraine

L'Université de Lorraine fournit des données de consommation énergétique des bâtiments, grâce à un outil de supervision en temps réel. Cependant, une problématique a rapidement émergé : il n'est pas possible d'extraire des données directement depuis cet outil. Ainsi, les données utilisées pour cette étude ont été collectées manuellement par les étudiants de l'ENSEM, en relevant les valeurs des capteurs de l'Université de Lorraine pendant le mois de février. Ces données statiques ont une portée purement pédagogique, car elles permettent d'illustrer des courbes types de consommation énergétique.

Afin d'afficher les données collectées, nous les avons ajoutées manuellement à la base de données ***InfluxDB***. Cette dernière permet l'importation de données à partir de fichiers `CSV`, à condition qu'ils soient préalablement traités pour respecter le format requis. Les données ainsi importées sont ensuite accessibles pour le *frontend* de la même manière que les données issues du *broker MQTT*.

G. Affichages du tableau de bord

La portée de ce projet, telle qu'énoncée dans l'introduction, est de sensibiliser quant aux effets de la surconsommation énergétique. C'est pourquoi l'ENSEM a fait l'acquisition d'une télévision sur laquelle seront affichées les différentes courbes et autres données. Ainsi, l'interface graphique de notre application se présente sous 2 formes :

- un **tableau de bord** sur lequel se trouvent tous les composants sous forme de grille pour une vue d'ensemble (Figure 11) ;
- un **carrousel** qui fait défiler les composants périodiquement en pleine page, spécifiquement pour la télévision (Figure 12).

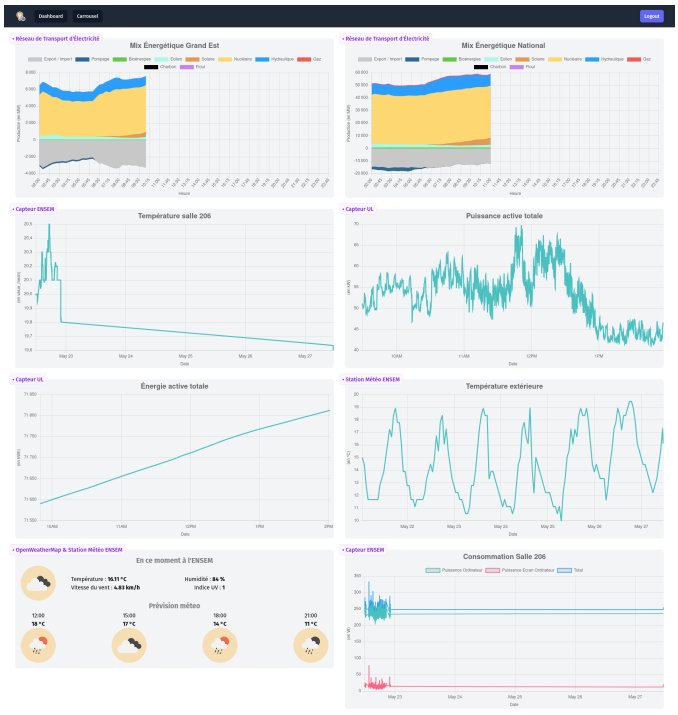

Fig. 11. Rendu du la page tableau de bord

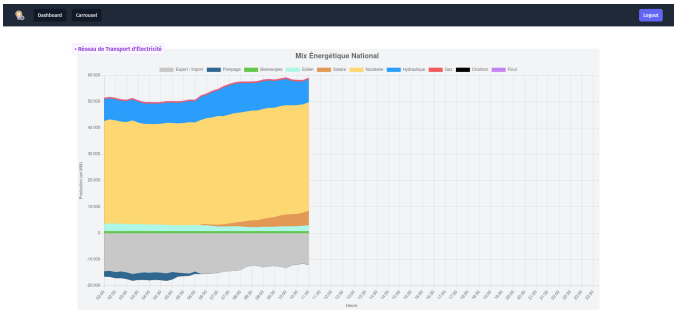

Fig. 12. Rendu de la page carrousel

H. Intégration et déploiement en continu

Télécom Nancy nous a mis à disposition une machine virtuelle pour pouvoir héberger les différentes composantes de notre application. Nous avons fait le choix d'utiliser les outils mis à disposition par GitHub pour faciliter la mise en production. En ce qui concerne le déploiement de l'interface graphique, nous avions deux possibilités :

- Héberger le serveur web (*frontend* et *backend*) sur la machine virtuelle qui contient aussi la base de données, les outils de communication et des algorithmes de traitements. Cette option a l'avantage de pouvoir permettre à quiconque dispose d'une connexion internet de visualiser le tableau de bord. Cependant, elle a l'inconvénient de surcharger le serveur sur lequel seront déjà présents d'autres services plus ou moins gourmands en termes de puissance de calcul mais également de stockage.
- Installer un raspberry pi au dos de l'écran côté ENSEM et héberger le code du *frontend* sur le mini ordinateur. Inconvénient principal : plus coûteux pour un gain de performance pas significatif.

Après nous être concertés avec notre encadrant et les principaux intéressés du côté de l'ENSEM, nous avons opté pour la première option. Une fois cette décision prise, nous avons profité de la conteneurisation de notre application pour simplifier la mise en production. Plus particulièrement, le fichier `Makefile` nous permet de compiler et de lancer l'application avec les commandes `make build` et `make up`. Ainsi, nous avons utilisé la fonctionnalité de *runners*[6] de GitHub. Cette fonctionnalité nous permet de lancer les actions de CI/CD sur un serveur distant, dans notre cas la machine virtuelle mise à disposition. Concrètement, à chaque *push* sur la branche principale, les conteneurs sont recompilés sur la machine virtuelle et si tout s'est bien passé, l'application est lancée.

Une fois cet environnement mis en place, chaque nouvelle fonctionnalité est codée et testée sur une branche séparée puis fusionnée pour être mise en production.

[6]https://docs.github.com/en/actions/using-github-hosted-runners

V. Bilan

A. Problèmes rencontrés

1) Manque de données: Dans l'analyse préliminaire, nous avions cerné trois sources différentes à partir desquelles afficher les données :

1) Les capteurs installés dans le cadre du projet étudiant de l'ENSEM;
2) Des *API* externes telles que celle de *RTE*, d'*OpenWeather* ou de la station météo de l'ENSEM;
3) Les capteurs de l'Université de Lorraine, entre autres pour le chauffage et les consommations électriques des différentes parties du bâtiment.

Nous nous sommes cependant heurtés à un problème, les données fournies par l'Université de Lorraine le sont à travers une page Web et non pas une *API*. De plus, une analyse du code source (côté client) et du trafic réseau ne nous ont pas permis de *scrapper* et donc d'intégrer ces données automatiquement dans notre application. Dans l'optique de tout de même montrer ce qu'il est possible de faire, nous avons injecté à la main des données récupérées sous la forme d'un fichier `.csv` par une personne de l'Université dans la BDD.

2) Synchronisation avec l'ENSEM: Pour rappel, notre partie du projet dépend en grande partie des informations fournies par l'ENSEM (données énergétiques, pertinence des données à afficher et maquettes du tableau d'affichage) comme le montre la figure 5. Cependant, tout comme nous avons rencontré des difficultés, les étudiants de l'ENSEM en charge de la collecte et de la transmission des données aussi. Ce qui a parfois retardé la livraison de jalons intermédiaires tels que la première version du tableau de bord ou l'adresse du serveur MQTT.

B. Extension du projet initial

Dès le début du projet, la possibilité d'adapter le système de supervision à d'autres bâtiments/écoles a été évoquée. Ainsi, nous avons développé notre application de manière "générique", en utilisant des conteneurs, des composants React réutilisables, des technologies *open-source*.

Cependant, le projet repose en grande partie sur les données fournies par le serveur NodeRed de l'ENSEM,

qui est, sans aucun doute, l'élément le plus compliqué à rendre générique. En effet, il aurait fallu se concerter avec les élèves de l'ENSEM lors de la conception de leur projet (choix et mise en place des capteurs) pour se mettre d'accord sur une convention pour l'envoi des données sur le serveur MQTT afin de pouvoir créer un composant générique.

VI. Conclusion

L'outil de supervision a été conçu dans le cadre d'un projet commun entre les étudiants de l'ENSEM et de TELECOM Nancy. Il vise à sensibiliser les étudiants aux enjeux de la consommation énergétique en fournissant une interface visuelle et conviviale pour les données de consommation énergétique. L'outil est modulaire et peut facilement s'adapter à des sources de données de complexité variable et évoluer dans le temps en intégrant de nouveaux capteurs. Cependant, l'outil souffre de la non-normalisation des outils I.O.T., ce qui impose le respect de normes arbitraires pour que l'intégration des données dans la base se déroule sans encombre.

Pour que l'outil soit pleinement efficace en tant qu'élément de sensibilisation, une augmentation du nombre de capteurs et de données pertinentes sera nécessaire. Cependant, l'outil fournit les bases nécessaires à des extensions futures et a le potentiel de devenir un outil pour la sensibilisation à la consommation énergétique dans les établissements d'enseignement.

Remerciements

Les auteurs tiennent à remercier les élèves de l'ENSEM pour leur travail préliminaire d'installation des capteurs, qui nous a permis de rejoindre le projet ATI en cours de route. Nous remercions également M. SONG pour son suivi attentif des équipes de travail de l'ENSEM ainsi que pour le travail de coordination qu'il a effectué. Enfin, nous tenons à exprimer notre gratitude envers M. CHOLEZ, notre encadrant de projet interdisciplinaire de recherche, pour ses précieux conseils et son aide tout au long de la conception de cet outil.

References

[1] *InfluxDB* - Site officiel
[2] Paessler - IT Explained : MQTT
[3] *Mosquitto* - Site officiel
[4] éCO$_2$mix - Toutes les données de l'électricité en temps réel
[5] Digora - IoT, Big Data : quelle base de données pour vos applications ?
[6] DB-Engines Ranking of Time Series DBMS
[7] OpenDataSoft - API sur le mix énergétique national.
[8] *Go Echo* - Site officiel
[9] SvelteKit - Site officiel