DOCUMENTATION PROJET IOT

Alexandra ROBIN, Sébastien SAEZ

INSTALLATION DE L'ENVIRONNEMENT

Installer MongoDB : https://docs.mongodb.com/manual/administration/install-community/
Installer nodeJS : https://nodejs.org/en/download/

Importer les collections Mongo :

```
> mongoimport --db iotbdd --collection temperatures --drop --file primer-dataset.json
> mongoimport --db iotbdd --collection leds --drop --file primer-dataset.json
> mongoimport --db iotbdd --collection photosensors --drop --file primer-dataset.json
> mongoimport --db iotbdd --collection batiments --drop --file primer-dataset.json
```

Le lancement du projet nécessite 3 terminaux.

- ➔ Dans le **1er terminal** :

  ```
  > mongod
  ```

Vérifier que mongod démarre bien sur le port **21017**.
Si ce n'est pas le cas, modifier le port dans les fichiers (ligne 4) **./app/routes.js** et **./app/mongo.js** du dossier **serveur** :

```
const url = 'mongodb://localhost:27017';
```

- ➔ Dans le **2nd terminal** : se placer dans le dossier serveur

  ```
  > npm install
  > node server.js
  ```

- ➔ Configurer l'Arduino :
  - Led : 19 (const int ledPin = 19;)
  - Temperature : 23 (const int temperaturePin = 23;)
  - Photoresistor : A0 (const int sensorPin = A0;)

- ➔ Connecter l'Arduino au PC et lancer le script.

- ➔ Configurer le client par rapport au serveur :
  Lors du démarrage, le serveur affiche une adresse IP et un port dans le terminal.

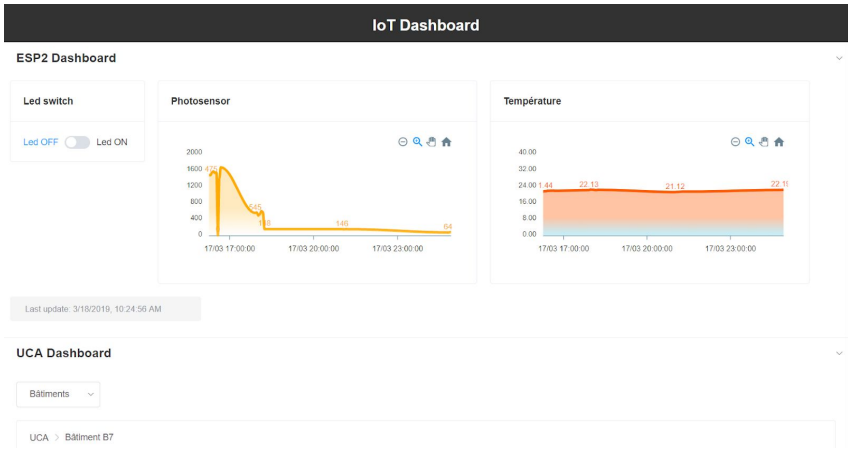
Dans le fichier **client_IHM/src/App.vue (env ligne 180)**, modifier la variable `serverIP` avec l'IP du serveur et la variable `port` avec le port du serveur.

```
serverIP: "192.168.43.63",
port: 8081
```

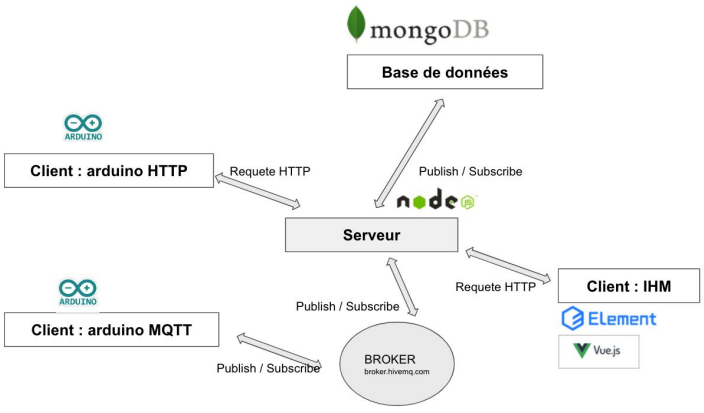
➔ Dans le **3eme terminal** : se placer dans le dossier client_IHM

```
> npm install
> npm run dev
```

Le client se lance automatiquement dans un navigateur :



ARCHITECTURE



- **Persistence des données** :
  Une base de données MongoDB est utilisée pour stocker les données des boards

Arduino.

- **Économie d'énergie** (partie HTTP) :
  La board Arduino effectue des deepsleeps de 2 minutes.

- Le **serveur Node.js** gère une flotte d'**Arduino clients** qui lui envoient leurs données pour qu'il les enregistre dans la base Mongo.

- L'**IHM est un client** qui effectue des requêtes HTTP au serveur. Celui-ci récupère les données de la base MongoDB pour les transmettre à l'IHM.

EXPLICATION DE CODE

ARDUINO HTTP

Le script fonctionne de la façon suivante :

1) Une minute d'action libre qui permet d'allumer/éteindre la led grâce à des requêtes GET toutes les ½ secondes.
2) Une fois la minute écoulée, on récupère la température et la luminosité.
3) On configure la requete PUT HTTP.
4) On envoie les données au serveur grâce aux requêtes PUT de température et de lumière.
5) L'Arduino s'endort pendant 2 minutes (deep sleep).
6) On recommence.

Nos décisions :

- 1 minutes d'action libre pour pouvoir tester la led.
- 2 minutes de sleep pour avoir des données assez rapidement (toutes les minutes) dans le cas du TP.

ARDUINO MQTT

La fonctionnalité deep sleep de l'Arduino n'a pas été implémentée dans ce script.

Le script fonctionne de la façon suivante :

1) On définit la localisation du capteur : Batiment et Salle.
2) On définit les differents topics.
3) On définit un seuil de température pour le jour et la nuit.
4) On définit un compteur pour faire un publish de la température toutes les 2 minutes.
5) On se connecte au Wifi.
6) On définit l'adresse et le port du broker MQTT : broker.hivemq.com / 1883
7) On crée une fonction de callback qui sera appelée lorsqu'un message arrive dans les topics abonnés.

8) Cette fonction de callback va permettre de récupérer l'heure actuelle et l'état du radiateur pour la salle.
9) On s'abonne donc au topic "Time" et au topic "Radiateur".
10) Le script fera des publish seulement si une heure courante est définie.
11) Il calculera ensuite le seuil à utiliser en fonction de cette heure courante.
12) Si la température est supérieure au seuil alors il publiera une alerte incendie en boucle pendant 1h.
13) En fonction du compteur, la température et son seuil sera publiée.

Nos décisions :

- Nous avons décidé de gérer les seuils directement dans le script Arduino. Il nous fallait donc la date et heure actuelle. Pour une question de simplicité, ces données sont récupérées sur un topic.

SERVEUR

Le serveur Node est composé de 4 fichiers :

➔ **server.js** : Démarre le serveur avec tous les composants.

➔ **app/mongo.js** : Exécute les requêtes MongoDB pour insérer/modifier/récupérer dans la base de données.

➔ **app/routes.js** : Permet au serveur d'échange des requêtes GET/POST/PUT avec tout le monde et défini les chemins pour les requêtes HTTP et les traitements associés.

➔ **app/mqtt.js** : S'abonne et publie sur les topics souhaités et réalise le traitement lors de la récupération des données.

Nos décisions :

- Nous avons décidé de mettre le serveur sur un PC et pas un Arduino pour une utilisation plus facile avec plusieurs Arduino clients.

CLIENT IHM

L'IHM est composée de 2 parties.

1) La première partie permet de gérer l'Arduino avec HTTP (TP2) :

    - Un switch/interrupteur permet d'allumer/éteindre la led positionnée sur le client Arduino.
    - On peut consulter les dernières données de température et de luminosité récupérées par l'Arduino.

- Les graphiques sont mis à jour automatiquement toutes les 2 minutes.
- On peut zoomer/dézoomer sur les graphiques et se déplacer sur l'axe des abscisses.

2) La seconde partie permet de gérer la température des salles des bâtiments de l'UCA grâce au protocole MQTT :

    - Un tableau affiche les données propres à un bâtiment choisi. Chaque ligne correspond a une salle du bâtiment.
    - Les données affichées sont les suivantes :
        - Les dernières températures sur un graphique.
        - La date et la valeur de la dernière température récupérée.
        - L'état du capteur : operationnel/defaillant
          → Un capteur est considéré comme défaillant si la dernière donnée envoyée par celui ci date de 1h avant la date courante.
        - L'état du radiateur : on/off.
    - La mise à jour automatique des données s'effectue toutes les 2 minutes.
    - Une alerte incendie est déclenchée si un incendie est détecté par l'Arduino.
        - La température est supérieure à 15/25 degrés en fonction de l'heure (nuit/jour).
        - Pour tester l'alerte incendie : réduire le compteur du publish de la température à 0 et débrancher l'Arduino quand la température dans le Serial dépasse le seuil.

Au niveau du code, le client IHM se compose essentiellement des éléments suivants (dossier **client_IHM/src**) :

➔ **main.js** :
- ◆ Importation de différents éléments externes : éléments de Vue.js et des librairies Element UI (composants graphiques), Apexcharts (charts), et Vue Notification (notification).
- ◆ Initialisation de la vue principale App.vue

➔ **App.vue** : Coeur de l'application. Contient toute la structure de l'IHM sous forme HTML, les données et les méthodes en Javascript, et le style en CSS.

➔ **components/PhotosensorChart.component.vue** :
Composant graphe adapté aux données du photosensor fournies par la board ESP32 (partie HTTP).

➔ **components/TemperatureChart.component.vue** :
Composant graphe adapté aux données de température fournies par la board ESP32

(partie HTTP).

➔ **components/TemperatureSalleMiniChart.component.vue** :
Composant graphe adapté aux données de température spécifique à chaque salle de l'UCA (partie MQTT).