Matthieu Diebolt

Rapport projet AP4A

Table des matières :

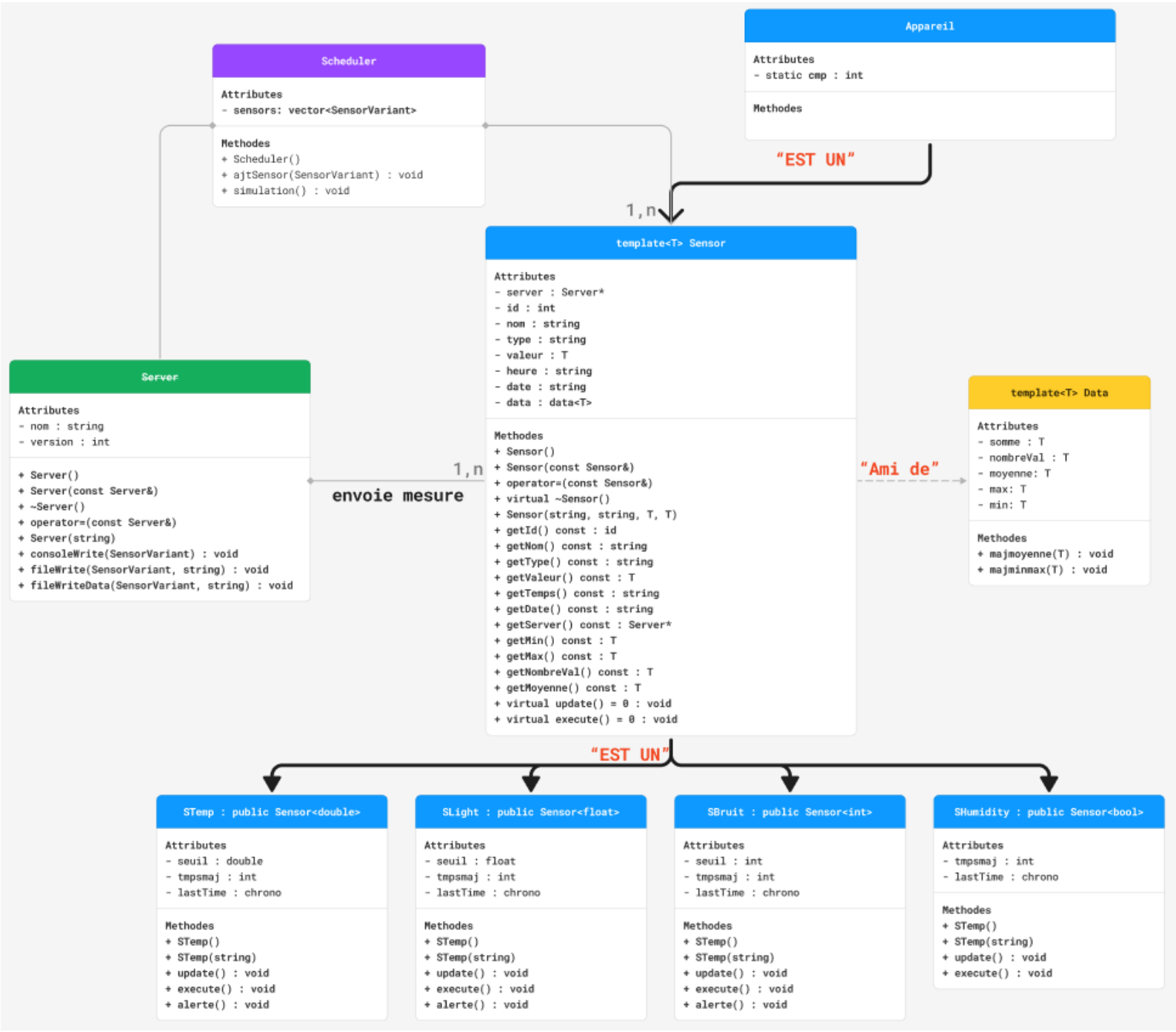
Introduction

Ce projet avait pour objectif de nous familiariser et de mettre en pratique les cours de C++ dans le cadre de l'UV AP4A. Pour cela, nous étions seuls et avions 3 séances de TP en plus du travail personnel pour le réaliser.

Le projet consistait à réaliser un simulateur permettant de modéliser un écosystème IoT spécialisé dans le monitoring de la qualité de l'air d'espaces de travail. Pour cela, nous devions implémenter un écosystème composé d'un serveur qui peut communiquer avec des capteurs de types différents : lumière, température, son, humidité... Le serveur devait également pouvoir enregistrer les relevés des capteurs dans des fichiers de logs spécialisés pour chaque capteur. De plus, un scheduler capable de simuler le relevé des informations des capteurs à intervalles réguliers devait être implémenté.

Diagramme UML et choix d'implémentation

Voici un diagramme UML présentant l'implémentation du projet :



J'ai fait le choix de rajouter une classe Appareil comme classe mère de Sensor afin de gérer les IDs uniques des capteurs de façon centralisé et de laisser la possibilité de rajouter de nouvelles fonctionnalités pour le projet. Par exemple, on pourrait imaginer un appareil qui ajoute de l'humidité dans l'air...

Pour la classe Sensor, j'ai fait le choix de l'implémenter en tant que classe template afin de pouvoir l'utiliser avec différents types de valeurs. De plus, j'ai décidé qu'elle serait abstraite, car dans mon projet, un capteur doit obligatoirement être spécialisé pour un type de relevé. J'ai également choisi de mettre les classes filles de Sensor (les capteurs avec une spécialisation) dans le même fichier que leur classe parente. Grâce à la classe template mère, chaque capteur spécialisé peut être de types différents, par exemple float, int, ou encore booléen.

En plus des attributs et méthodes que je définis dans la classe mère, j'ajoute pour la spécialisation des capteurs des attributs comme le seuil (par exemple, une température trop élevée), tmpsmaj (intervalle de temps entre chaque nouveau relevé du capteur), et l'heure lastTime du dernier relevé. Chaque capteur spécialisé possède également des méthodes supplémentaires. J'ai choisi d'ajouter ces méthodes et attributs dans les classes filles car je souhaitais que chaque capteur puisse avoir ou non cette fonctionnalité ; par exemple, la méthode alerte() ou des fonctionnalités de data ne sont pas utilisées dans SHumidity, car elles sont inutiles selon moi pour des capteurs avec valeurs booléennes.

La méthode alerte() permet simplement d'afficher un message sur la console si un certain seuil est dépassé, indiquant qu'il y a un problème dans la pièce où est installé le capteur.

Pour le fonctionnement de la mise à jour des nouveaux relevés des capteurs, dans mon projet, execute() sert à simuler un nouveau relevé, et donc à changer l'heure, la date, et la valeur relevée. La valeur est générée de façon aléatoire selon une plage de valeurs et le type de valeur par exemple la fonction random(5.0f,10.0f) renverra un valeur flottant entre 5.0 et 10.0, les différents types sont gérés grâce à la une fonction Template. Le temps (l'heure) et la date sont récupérés par les fonction indépendantes au class, recupdate() et recuptemps().Quant à update(), grâce aux attributs tmpsmaj (qui indique tous les combien de secondes minimum le capteur doit se mettre à jour) et lastTime, elle vérifie si le temps de tmpsmaj est écoulé, puis lance execute() pour effectuer un nouveau relevé. De plus, update() met à jour les données de la classe data.

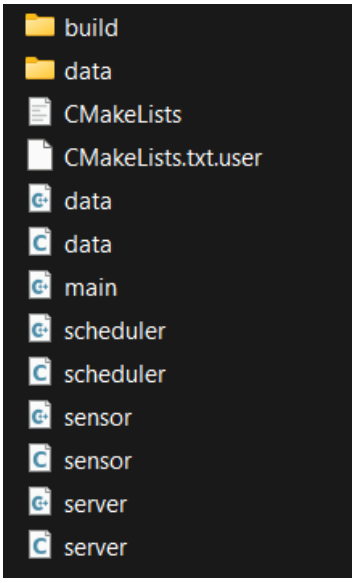
En ce qui concerne la classe data, j'ai choisi de l'ajouter au projet afin d'obtenir des informations supplémentaires sur nos capteurs. Actuellement, elle enregistre la moyenne de toutes les valeurs relevées par les capteurs, le relevé le plus faible enregistré, et le relevé le plus élevé. C'est le rôle du serveur d'enregistrer les données de data dans des fichiers de logs spécialisés.

La classe server possède trois méthodes : consoleWrite(), qui écrit de façon formaté les données des capteurs ; fileWrite(), qui enregistre dans des fichiers de logs spécialisés pour chaque type de capteur les données relevées ; et enfin fileWriteData(), qui enregistre les données de data dans des fichiers également spécialisés. Pour fileWriteData(), contrairement à fileWrite() qui ajoute une ligne à chaque nouveau relevé, j'ai choisi que seule la première ligne soit utilisée et réécrasée pour la mettre à jour à chaque nouveau relevé.

Enfin, pour la classe Scheduler, à partir d'une liste de pointeurs vers des capteurs Sensor de types différents, elle simule, grâce à Simulation(), des relevés à intervalles réguliers. J'ai décidé qu'à chaque intervalle, les informations des données de chaque capteur seraient affichées. Par exemple, il faudra attendre trois relevés pour voir une mise à jour d'un capteur, alors qu'un seul relevé suffira pour un autre, puisque chaque capteur se met à jour à un intervalle de temps différent.

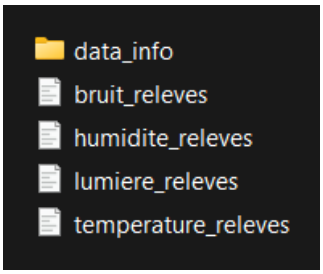
Exécution

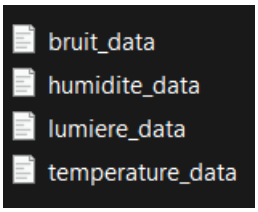
L'ensemble de ces fichiers doit être présent :



Pour l'implémentation du projet sur un nouvel appareil, si les fichiers de logs contenant les diverses données des capteurs et de data ne s'ouvrent pas correctement ou ne se créent pas au bon endroit, il faudra alors modifier dans 'scheduler.cpp' la destination des fichiers : Par default les fichiers devraient se créer dans data pour les relevés des capteurs et dans data info pour les data supplémentaires.

```
server->fileWrite(sensor, "../../data/"); //on ecrit le r
server->fileWriteData(sensor, "../../data/data_info/"); /
```





De plus il est possible de changer le temps entre chaque relevé de la simulation en modifiant cette ligne dans la méthode simulation du fichier ‘scheduler.cpp’ : (par default le temps est de 4 sec + 1sec) :

```
24          std::this_thread::sleep_for(std::chrono::seconds(1))
```

```
39          std::this_thread::sleep_for(std::chrono::seconds(4));
```

Enfin, il est possible de modifier les intervalles de mise à jour des données (qui correspond à un nouvel relever du capteur physique) en modifiant l’attribut « tempsmaj » des class SBruit, STemp, SHumidty, SLight… dans le fichier ‘sensor.h’ : (Par exemple ici le sensor se mettra à jour au min toutes les 20 sec) :

```
148         int tempsmaj = 20;
```

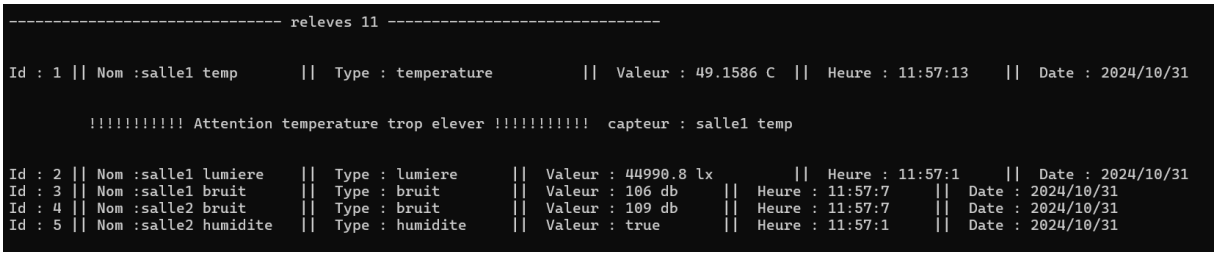
Présentation

Lorsqu’on lance le programme les relevés se lance à intervalles réguliers

```
------------------------------ releves 8 ------------------------------

Id : 1 || Nom :salle1 temp       ||  Type : temperature          ||  Valeur : 24.6723 C  ||  Heure : 11:56:55    ||  Date : 2024/10/31
Id : 2 || Nom :salle1 lumiere    ||  Type : lumiere       ||  Valeur : 65758.3 lx         ||  Heure : 11:56:43    ||  Date : 2024/10/31
Id : 3 || Nom :salle1 bruit      ||  Type : bruit         ||  Valeur : 111 db      ||  Heure : 11:56:37    ||  Date : 2024/10/31
Id : 4 || Nom :salle2 bruit      ||  Type : bruit         ||  Valeur : 24 db       ||  Heure : 11:56:37    ||  Date : 2024/10/31
Id : 5 || Nom :salle2 humidite   ||  Type : humidite      ||  Valeur : true        ||  Heure : 11:56:43    ||  Date : 2024/10/31


------------------------------ releves 9 ------------------------------

Id : 1 || Nom :salle1 temp       ||  Type : temperature          ||  Valeur : 21.2798 C  ||  Heure : 11:57:1     ||  Date : 2024/10/31
Id : 2 || Nom :salle1 lumiere    ||  Type : lumiere       ||  Valeur : 44990.8 lx         ||  Heure : 11:57:1     ||  Date : 2024/10/31
Id : 3 || Nom :salle1 bruit      ||  Type : bruit         ||  Valeur : 111 db      ||  Heure : 11:56:37    ||  Date : 2024/10/31
Id : 4 || Nom :salle2 bruit      ||  Type : bruit         ||  Valeur : 24 db       ||  Heure : 11:56:37    ||  Date : 2024/10/31
Id : 5 || Nom :salle2 humidite   ||  Type : humidite      ||  Valeur : true        ||  Heure : 11:57:1     ||  Date : 2024/10/31


------------------------------ releves 10 ------------------------------

Id : 1 || Nom :salle1 temp       ||  Type : temperature          ||  Valeur : 23.2618 C  ||  Heure : 11:57:7     ||  Date : 2024/10/31
Id : 2 || Nom :salle1 lumiere    ||  Type : lumiere       ||  Valeur : 44990.8 lx         ||  Heure : 11:57:1     ||  Date : 2024/10/31
Id : 3 || Nom :salle1 bruit      ||  Type : bruit         ||  Valeur : 106 db      ||  Heure : 11:57:7     ||  Date : 2024/10/31
Id : 4 || Nom :salle2 bruit      ||  Type : bruit         ||  Valeur : 109 db      ||  Heure : 11:57:7     ||  Date : 2024/10/31
Id : 5 || Nom :salle2 humidite   ||  Type : humidite      ||  Valeur : true        ||  Heure : 11:57:1     ||  Date : 2024/10/31
```

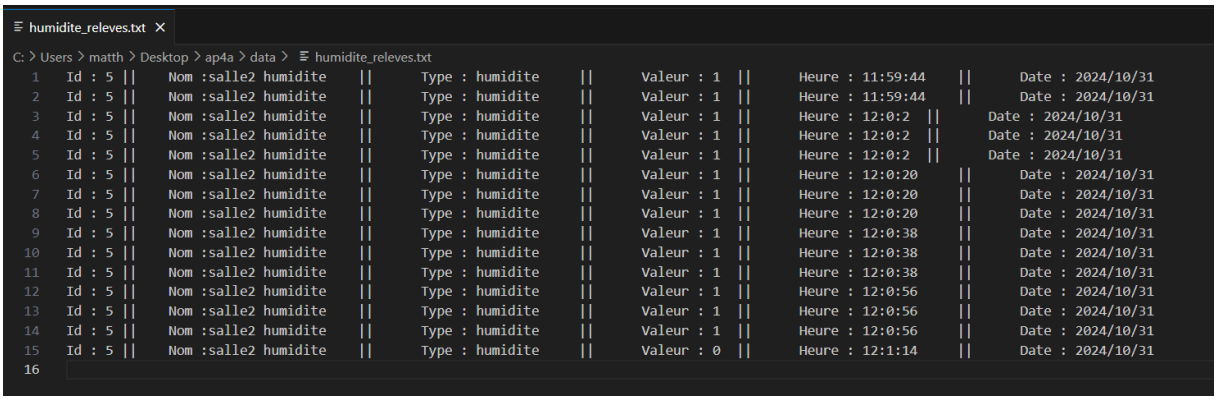
On peut voir ici que nous avons 5 capteurs. Chaque capteur à un id unique, le type de valeur ainsi que l’unité de la valeur varie selon le type de capteur. De plus on peut voir l’heure (et la date) à laquelle le dernier relevé à été fait par le capteur, ainsi nous avons des heures qui varient entre le type de capteur car chaque type s’active à un intervalle différent par exemple la température toutes les 5 secondes, le bruit toutes les 30 secondes…

Ici nous pouvons voir le fonctionnement de l'alerte :



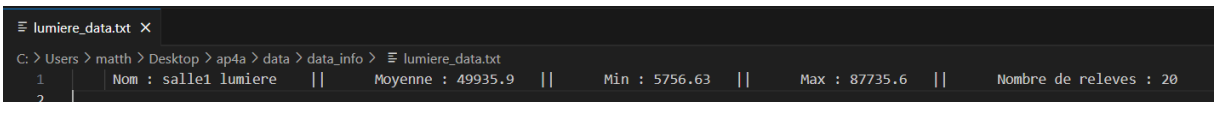
On peut voir que la température a atteint 49.1°C ainsi vu qu'il s'agit d'une température bien trop élevée, une alerte est affichée sur la console.

Enfin, si l'on va dans le dossier data et que l'on ouvre par exemple humiditite_releves.txt on pourra observer :



On peut voir que les données du capteur sont correctement enregistrées dans le fichier de logs créer spécialement pour les capteurs d'humidité. Les nouveaux relevés se rajoutent l'un après l'autre.

Et si l'on va dans le dossier data info et que l'on ouvre un fichier par exemple lumière_data.txt, on pourra observer :



On peut voir qu'il n'y a qu'une seule ligne qui se met à jour et l'on peut voir le nombre de relevés effectués par le capteur ainsi qu'une moyenne faite sur les relevés ainsi que la valeur la plus basse et la plus haute enregistrées.

Piste d’améliorations

Pour améliorer mon projet, j’aurais pu rendre d'autres classes ou méthodes Template, par exemple consoleWrite(), ce qui m’aurait permis de personnaliser encore plus le relevé selon le type de capteur. J’aurais également pu ajouter davantage d’attributs uniques ou de méthodes à chaque type de capteur.

De plus, j’aurais pu gérer les valeurs des relevés de façon plus logique, en ne prenant pas un nombre totalement aléatoire, mais par exemple, si la température est de 20, je pourrais l'augmenter ou la diminuer d'un nombre compris entre 0 et 5 à la prochaine mise à jour. Cela m’éviterait de passer de 0 à 40 en un seul relevé.

J’aurais également pu mieux gérer la console en donnant la possibilité à l’utilisateur d’arrêter la simulation pour ajouter un capteur à partir de la console, puis de relancer la simulation. Enfin, j’aurais pu imaginer d'autres appareils, comme un « chauffage », par exemple, qui augmenterait la température de la pièce de 5 °C en une certaine période si la température de la pièce est trop froide.

Conclusion

En conclusion, la création de ce simulateur d’écosystème IoT m’a permis de mettre en pratique les notions de C++ abordées dans le cadre de l'UV AP4A. De plus, j'ai pu explorer de nouvelles notions non vues en cours en effectuant des recherches personnelles, comme la gestion du temps ou encore les nombres aléatoires. Enfin, j'ai découvert la notion de diagramme UML. Ce projet a donc été très enrichissant pour moi.