Rapport de projet

AP4A

Sommaire

Introduction

Ce projet avait pour objectif de réaliser un simulateur de réseau IoT spécialisé dans la surveillance de la qualité de l'air d'un espace de travail. Il est composé de différents capteurs : température, lumière, son et humidité ainsi qu'un serveur. Le serveur doit être capable d'écrire les informations renvoyées par les capteurs en console et dans des fichiers logs triés par capteur. L'ensemble des éléments est contrôlé par un planificateur, faisant le relais entre les données des capteurs et le serveur. Il permet également de régler l'intervalle d'actualisation indépendamment pour chaque capteur.

Implémentations

Mon projet se divise en plusieurs fichiers en-têtes et sources comme affichés ci-dessous :

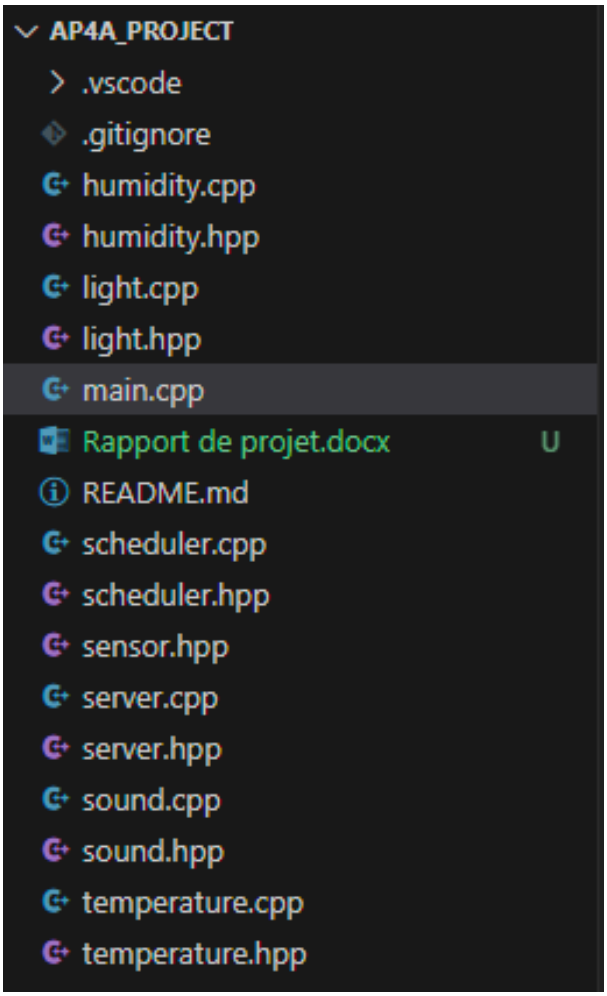
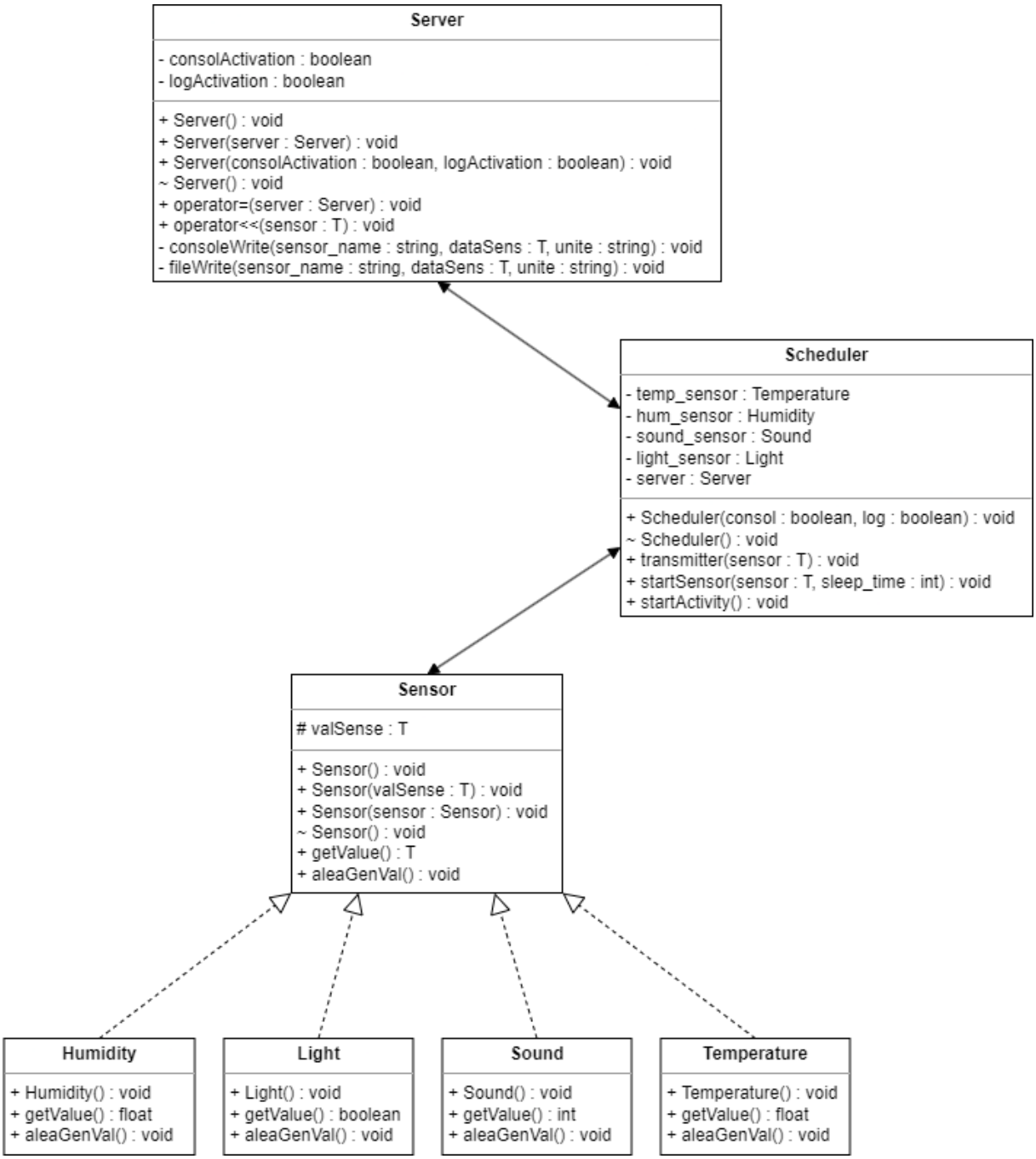


Diagramme UML représentant les classes du projet :



J'ai fait le choix de diviser les types de capteurs en fichiers indépendants. Les classes des capteurs (Temperature, Sound, Light, Humidity) sont des classes filles de la classe « Template » Sensor. Elles partagent donc les mêmes constructeurs et le même destructeur. J'ai redéfini les méthodes getValue et aleaGenVal dans chaque classe fille pour que ces deux méthodes renvoient des valeurs du type associé à la classe et pour que la génération de la valeur aléatoire renvoie une valeur dans un intervalle plus réaliste en fonction du capteur dans l'hypothèse qu'ils sont situés dans un logement. Les valeurs générées par la bibliothèque cstdlib avec l'utilisation de srand et rand n'étant pas satisfaisantes, j'ai fait le choix d'utiliser la bibliothèque random pour générer des valeurs semblant bien plus aléatoire. Quant

au serveur, j'ai décidé de garder 2 attributs booléens : consolActivation et logActivation qui indiquent respectivement lorsque l'écriture dans la console ou dans les fichiers logs est activée. Ces deux attributs sont paramétrables à l'exécution du programme en passant des arguments (voir Exécution). Pour l'écriture en console et dans les fichiers logs, l'identification du type de capteur se fait par correspondance avec le type de la classe du capteur en paramètre. Un formatage des informations est ensuite effectué pour ajouter des informations comme l'unité de la valeur du capteur et son nom pour l'inscrire dans le fichier correspondant. La transmission des informations des capteurs s'effectue grâce à la méthode transmitter du Scheduler qui prend le capteur en paramètre et envoie l'information du capteur au serveur. Les valeurs générées par les capteurs sont gérées par des threads qui génèrent et envoient au serveur, à intervalle prédéfini par macro. Pour un projet à but commercial, mon choix aurait été de définir à l'exécution par argument les intervalles mais pour un projet comme celui-ci, la modification de macro m'a semblé suffisant pour la démonstration du simulateur. Les macros sont modifiables dans le fichier scheduler.cpp. Mes dernières modifications par rapport à l'UML de référence se portent sur le fichier main.cpp : J'ai ajouté quelques arguments pour donner le contrôle à l'utilisateur (voir Exécution) ainsi que la gestion du signal SIGINT pour arrêter le programme correctement à l'appui de Ctrl + C.

Le développement du simulateur s'est effectué à l'aide de Git et Valgrind.

Exécution

Prérequis :

- Installer GNU Compiler Collection (GCC)

Pour compiler le programme, il suffit de taper la commande suivante dans le répertoire racine du projet :

g++ *.cpp -o main

Pour exécuter le programme, il faut taper la commande suivante toujours dans le répertoire racine du projet :

- Sur Windows : .\main <arguments>
- Sur Linux : ./main <arguments>

Plusieurs arguments sont disponibles :

- Activer l'écriture dans les logs : -l OU --log
- Activer l'écriture en console : -c OU –console
- Afficher la version et la date de compilation : -v OU – version
- Afficher l'aide : -h OU –help

Par défaut, le programme s'exécute avec consolActivation et logActivation sur false. Il est possible d'exécuter le programme avec plusieurs arguments à la fois (comme -l et -c).

L'arrêt du programme s'effectue par l'appuie de Ctrl + C.

Quelques images de l'exécution de programme :

```
PS C:\                          \Programmes\AP4A_Project> .\main -v
AP4A Project - v1.0
Compilation date : Nov  3 2023 12:39:07
```

```
PS C:\                          \Programmes\AP4A_Project> .\main -h
Usage : C:\Users\HugoPlayer\Documents\Programmes\AP4A_Project\main.exe [options]

Options : -h, --help      Print this help manual.
          -v, --version   Print program's version and compilation date.
          -l, --log       Turn on writing on log file.
          -c, --console   Turn on console writing.
```

Exécution sans argument :

```
PS C:\                          \Programmes\AP4A_Project> .\main
Building temperature sensor...
Building humidity sensor...
Building sound sensor...
Building light sensor...
Building server...
Scheduler started.
```

Exécution avec l'argument -c :

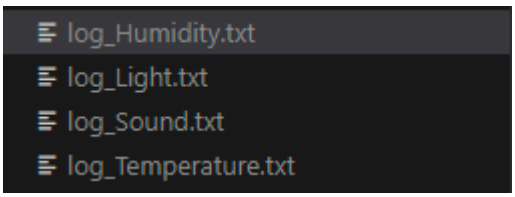
```
PS C:\                          \Programmes\AP4A_Project> .\main -c
Building temperature sensor...
Building humidity sensor...
Building sound sensor...
Building light sensor...
Building server...
Scheduler started.

The TemperatureThe  sensor returns : HumidityThe The SoundLight sensor returns :  sensor returns :  sensor returns : off
105 dB
33.0343 %20.6761
 °C
The Temperature sensor returns : 27.7305 °C
The LightThe  sensor returns : Temperatureoff sensor returns :
19.9312 °C
The Humidity sensor returns : 76.7273 %
The Temperature sensor returns : 6.25764 °C
The Sound sensor returns : The Light sensor returns : 117on dB

The Temperature sensor returns : 13.688 °C
The Temperature sensor returns : 33.4144 °C
The Humidity sensor returns : 71.1551 %
The Light sensor returns : on
The Temperature sensor returns : 27.454 °C
The Temperature sensor returns : 31.7965 °C
The Sound sensor returns : 64 dB
The Light sensor returns : on
The Temperature sensor returns : 26.0974 °C
```

Exécution avec l'argument -l :

```
PS C:\                          \Programmes\AP4A_Project> .\main -l
Building temperature sensor...
Building humidity sensor...
Building sound sensor...
Building light sensor...
Building server...
Scheduler started.
```



log_Temperature.txt

```
1  The Temperature sensor returns : 34.3511 °C
2  The Temperature sensor returns : 11.1717 °C
3  The Temperature sensor returns : 8.00516 °C
4  The Temperature sensor returns : 25.0236 °C
5  The Temperature sensor returns : 27.4057 °C
6  The Temperature sensor returns : 11.7374 °C
7  The Temperature sensor returns : 5.83699 °C
8  The Temperature sensor returns : 27.8678 °C
9  The Temperature sensor returns : 11.0342 °C
10 The Temperature sensor returns : 10.4903 °C
11 The Temperature sensor returns : 14.9492 °C
12 The Temperature sensor returns : 11.2904 °C
13 The Temperature sensor returns : 26.6716 °C
14 The Temperature sensor returns : 7.2085 °C
15 The Temperature sensor returns : 11.5651 °C
16 The Temperature sensor returns : 25.7611 °C
17 The Temperature sensor returns : 28.5721 °C
18 The Temperature sensor returns : 16.2949 °C
19 The Temperature sensor returns : 33.2649 °C
20 The Temperature sensor returns : 11.4348 °C
21 The Temperature sensor returns : 12.8761 °C
22 The Temperature sensor returns : 5.55466 °C
23 The Temperature sensor returns : 34.4877 °C
24 The Temperature sensor returns : 5.28612 °C
25 The Temperature sensor returns : 13.7117 °C
```

log_Humidity.txt

```
1  The Humidity sensor returns : 15.3705 %
2  The Humidity sensor returns : 51.5734 %
3  The Humidity sensor returns : 72.6093 %
4  The Humidity sensor returns : 32.2682 %
5  The Humidity sensor returns : 62.9721 %
6  The Humidity sensor returns : 84.453 %
7  The Humidity sensor returns : 27.9741 %
8  The Humidity sensor returns : 73.8278 %
9  The Humidity sensor returns : 77.3019 %
10 The Humidity sensor returns : 24.2009 %
11 The Humidity sensor returns : 36.4698 %
12 The Humidity sensor returns : 63.2632 %
13 The Humidity sensor returns : 18.2552 %
14 The Humidity sensor returns : 77.8605 %
15 The Humidity sensor returns : 28.8247 %
16 The Humidity sensor returns : 71.1304 %
17 The Humidity sensor returns : 26.2264 %
18 The Humidity sensor returns : 21.6778 %
19 The Humidity sensor returns : 0.492403 %
20 The Humidity sensor returns : 50.1797 %
21 The Humidity sensor returns : 6.82956 %
22 The Humidity sensor returns : 72.6754 %
23 The Humidity sensor returns : 76.8541 %
24 The Humidity sensor returns : 74.2122 %
25
```

Piste d'amélioration

Quelques améliorations sont possibles :

- Empêcher l'écriture simultanée des threads en console pour améliorer la lisibilité des premières lignes à l'exécution du programme
- Modifier pendant l'exécution du programme les caractères utilisées par le terminal Windows pour afficher correctement les caractères spéciaux
- Donner le choix à l'utilisateur final à travers des arguments de modifier les intervalles d'actualisation des capteurs

Conclusion

A travers la création de ce simulateur d'écosystème IoT, j'ai pu mettre en application l'apprentissage du langage C++ ainsi que le développement orienté objet plus généralement. De plus, j'ai pu découvrir le fonctionnement du compilateur g++, très similaire à gcc, ainsi que plusieurs outils tel que gdb et Valgrind pour débugger le programme lors de son développement et vérifier la présence de fuite de mémoire. L'utilisation d'outil de suivi comme git m'a été très utile et a simplifié grandement le développement du projet. Enfin, la création d'un diagramme UML a facilité l'écriture du code et la structuration de ce dernier.