# Une image contenant texte, capture d’écran, Police, Graphique Description générée automatiquement

Table des matières

[Compte Rendu de Projet AP4A 0](#_Toc182072641)

[1. Introduction 2](#_Toc182072642)

[1.1 Objectif du Projet 2](#_Toc182072643)

[1.2 Contexte 2](#_Toc182072644)

[1.3 Technologies Utilisées 2](#_Toc182072645)

[2. Analyse et Conception 2](#_Toc182072646)

[2.1 Structure Générale du Projet 2](#_Toc182072647)

[2.2 Description des Classes 3](#_Toc182072648)

[3. Implémentation 5](#_Toc182072649)

[3.1 Méthodes Clés 5](#_Toc182072650)

[3.2 Gestion des Fichiers CSV 6](#_Toc182072651)

[4. Tests et Résultats 6](#_Toc182072652)

[4.1 Exécution des Capteurs 6](#_Toc182072653)

[4.2 Vérification des Fichiers CSV 6](#_Toc182072654)

[4.3 Affichage en Console 6](#_Toc182072655)

[5. Conclusion et Perspectives 7](#_Toc182072656)

[5.1 Bilan du Projet 7](#_Toc182072657)

[5.2 Difficultés Rencontrées 7](#_Toc182072658)

[5.3 Améliorations Futures 7](#_Toc182072659)

# 1. Introduction

## 1.1 Objectif du Projet

Ce projet consiste à développer un simulateur d'écosystème IoT permettant de surveiller la qualité de l'air dans un espace de travail. Il repose sur un serveur central qui reçoit et analyse des données provenant de plusieurs types de capteurs : température, humidité, son, et lumière.

## 1.2 Contexte

Le simulateur IoT est conçu pour permettre le suivi de paramètres environnementaux clés. Chaque capteur simule des lectures de données et les envoie au serveur pour être enregistrées et analysées. Le projet utilise les concepts de la programmation orientée objet en C++ et exploite les fonctionnalités d’héritage, de polymorphisme, et d'abstraction pour une architecture flexible et extensible.

## 1.3 Technologies Utilisées

Le projet est réalisé en C++ avec le compilateur g++. La gestion de fichiers se fait via la bibliothèque <filesystem>, et le projet utilise une structure multi-fichier pour une meilleure organisation et maintenabilité.

# 2. Analyse et Conception

## 2.1 Structure Générale du Projet

L'architecture du projet est organisée autour de quatre classes principales :

* Server : Représente le serveur central recevant les données des capteurs.
* Scheduler : Responsable de l'activation cyclique des capteurs pour simuler les lectures de données en temps réel.
* Sensor : Classe abstraite qui définit le comportement générique des capteurs.
* Classes spécifiques de capteurs : TemperatureSensor, HumiditySensor, SoundSensor, LightSensor.

**Schéma UML des Classes**

**Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement**

## 2.2 Description des Classes

**Server**

La classe Server sert de point central pour la réception et l'analyse des données envoyées par les capteurs. Elle enregistre les données dans des fichiers .CSV dédiés à chaque capteur. Un en-tête (Time | Value) est écrit dans le fichier si celui-ci est vide ou inexistant. Cette classe peut également afficher des données dans la console.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

**Scheduler**

La classe Scheduler gère l’activation des capteurs selon un intervalle de temps fixe. Elle appelle la méthode update de chaque capteur, ce qui permet de simuler des lectures périodiques de données. Scheduler utilise le polymorphisme pour activer tous les capteurs sans connaître leur type spécifique.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**Sensor (classe abstraite)**

Sensor est une classe abstraite qui définit une interface pour les capteurs. Elle contient une méthode virtuelle pure update qui est redéfinie par chaque capteur spécifique.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

**Capteurs Spécifiques**

* **TemperatureSensor** : Génère des valeurs de température et les envoie au serveur.
* **HumiditySensor** : Génère des valeurs d'humidité.
* **SoundSensor** : Simule le niveau sonore en décibels.
* **LightSensor** : Simule la détection de lumière (allumé/éteint).

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

# 3. Implémentation

## 3.1 Méthodes Clés

**Méthode Server::logToFile**

Cette méthode enregistre les données dans un fichier CSV. Si le fichier n'existe pas ou est vide, elle ajoute un en-tête au début. Ensuite, elle ajoute chaque nouvelle donnée sous forme Time,Value.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**Méthode Scheduler::simulate**

La méthode simulate de la classe Scheduler active chaque capteur dans un cycle de 10 itérations, en utilisant une pause d'une seconde entre chaque cycle pour simuler des lectures en temps réel. Après chaque update, les données de capteurs sont transmises au serveur.

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

**Méthode Sensor::update pour les Capteurs Spécifiques**

Chaque capteur spécifique redéfinit la méthode update pour générer et enregistrer ses données de manière polymorphique. Par exemple, TemperatureSensor enregistre des valeurs de température en utilisant un format prédéfini.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

## 3.2 Gestion des Fichiers .CSV

Le serveur génère un fichier .CSV pour chaque capteur, avec un en-tête unique en première ligne (Time | Value). Chaque nouvelle donnée de capteur est horodatée et ajoutée à la fin du fichier.

# 4. Tests et Résultats

## 4.1 Exécution des Capteurs

Chaque capteur est testé pour s’assurer qu’il génère des données et les envoie correctement au serveur. L'affichage en console confirme que les données sont générées de manière réaliste.

## 4.2 Vérification des Fichiers .CSV

Les fichiers CSV générés pour chaque capteur sont vérifiés pour garantir qu'ils contiennent l'en-tête unique en première ligne et des valeurs horodatées pour chaque mesure. Ci-dessous un exemple de fichier généré pour le capteur de température.

## 4.3 Affichage en Console

L'affichage en console pendant la simulation montre que les capteurs envoient régulièrement des données au serveur.

# 5. Conclusion et Perspectives

## 5.1 Bilan du Projet

Ce projet a permis de concevoir un simulateur IoT pour surveiller la qualité de l'air dans un environnement de travail, en utilisant des principes de la programmation orientée objet tels que l'héritage, le polymorphisme et l'abstraction.

## 5.2 Difficultés Rencontrées

Certaines difficultés incluent l'implémentation du polymorphisme, la gestion des fichiers CSV avec un en-tête unique, et l'utilisation des bibliothèques de gestion de fichiers.

## 5.3 Améliorations Futures

Plusieurs améliorations sont possibles :

* Ajout de nouveaux types de capteurs.
* Utilisation de threads pour simuler les lectures de données en temps réel.
* Création d'une interface graphique pour visualiser les données en temps réel.
* Ajout de fonctions avancées d’analyse de données dans la classe Server.