**Design Document**

# Diagramme de classe

Une image contenant texte, diagramme, Parallèle, Plan

Description générée automatiquement

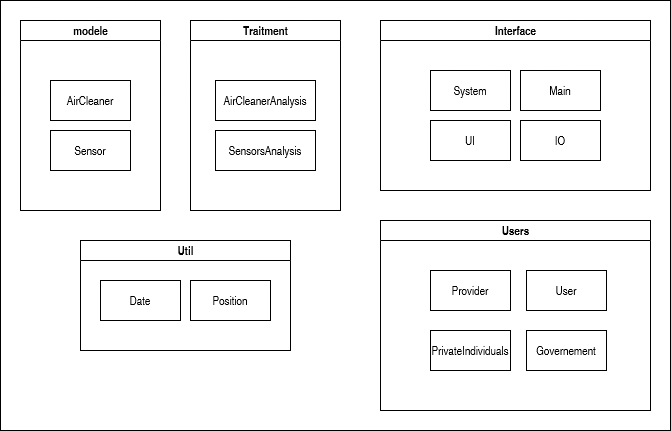
*Figure 1 : diagramme de classe de AirWatcher*

La classe **Main** contient uniquement le menu.

Il commencera par créer un **System**. Le **System** créé la **Fabrique** qui construit tous les éléments à partir des fichiers (**Sensor**, **AirWatcher** et **User**). Ils sont ensuite récupérés par le **System**.

Les **SensorAnalysis** et **CleanerAnalysis** sont créés lors de l’appel d’une méthode et détruits à la fin de la méthode. Cela permettrait de faire du multithreading lors du lancement d’une fonctionnalité.

# Décomposition modulaire



*Figure 2 : Diagramme de décomposition des classes*

# Diagrammes de séquences

Dans le premier cas on considère [A REMPLIR]

[IMAGE]

X3

# Principaux algorithmes

REFAIRE LES ALGO POUR LA DETECTION DE CAPTEUR DYSFONCTIONNEL & SIMILARITE

detectProblem() {

Si la mesure n et n+1 a un écart supérieur à 15% sur une des quatres données

mettre la variable true à doute

Si la mesure n et n+1 a de nouveau un écart supérieur à 15% sur une des quatres données et doute = true

Return true

Return false

}

findSimilarity(capteurId, dateDebut, dateFin) {

Récupérer la liste de tous les capteurs

parcourir la liste des capteurs

Si un capteur possède la même moyenne et écart type sur la période de temps avec un écart de 5% et distance > 200m

alors ajouter le capteur à la liste similar.

Return similar

}

checkEfficiency(AirCleaner, vector<Sensor>) {

Trier les capteurs par ordre de distance avec le airCleaner.

Pour le capteur (non défectueux) le plus proche et pour chaque particule

Calculer le pourcentage d’efficacité entre la veille du lancement du airCleaner et le lendemain

Définir la borne de non efficacité à 25% de l'efficacité

Regarder jusqu’à quelle date après la fin de fonctionnement du cleaner le relevé est inférieur à la borne

Si l’efficacité est négligeable (inférieur à 5%) pour tous les polluants afficher un message est partir.

Pour les capteurs suivants

Si les valeur du premier jour de fonctionnement sont inférieur à la borne

Passer au capteur suivant

Sinon :

Sortir de la boucle

Calculer la distance entre l’airCleaner et le dernier capteur parcouru

Afficher l’efficacité de fonctionnement pour chaque polluant, le rayon d’action et le temps d’action après l'arrêt.

}

# Fichiers de sauvegarde

Nous devons créer des fichiers de sauvegarde afin de se remémorer les login et mot de passe des utilisateurs. Ces derniers ne seront pas cryptés.

Il s’agit du fichiers *data.csv*. Il contient les données dans le format suivant:

LOGIN ; MDP ; TYPE ; [NB de point] ; [NB de capteurs/airCleaners] ; [ID du capteur/airCleaner 1] ; [ID du capteur/airCleaner 2]…

Le type est 1 si c’est un utilisateur disposant d’un capteur ou non, 2 si c’est un fournisseur d’airCleaner ou 3 si c’est un membre du gouvernement.

Les valeurs suivantes sont présentes pour les utilisateurs et les fournisseurs uniquement.

Pour un utilisateur, il s’agit du nombre de point qu’il a accumulé, puis du nombre de capteurs qu’il possède et enfin liste des id de ces capteurs.

Pour les fournisseurs, il s’agit uniquement du nombre de airCleaner déployé ainsi que la liste des id de ces derniers.

# Tests unitaires

User checkpasswd(String login, String password)

Doit renvoyer user si l’utilisateur est dans le fichier *data.csv*

nullptr sinon

données : checkpasswd(user1, mdp) ⇒ user

checkpasswd(user1, mdp2) ⇒ NULLPTR

String getBonReduction()

Si le user.getPoints() est suffisant : Renvoyer vrai

Sinon : Renvoyer faux

données : if user.getPoints() > 150 ⇒ True

user.getPoints() < 150 ⇒ False

bool detectProblem()

Si le capteur possède une anomalie (par exemple un écart de valeur > 10% par heure) on renvoie True

et on appelle aSensor.setIsValid(false)

Sinon on renvoie False

données : pour O2 : 12h 21.1, 13h 22.0, 14h 21.5 ⇒ False

: 12h 21.1, 13h 35.0, 14h 15.5 ⇒ True

String generateStatistics(pos, startTime, endTime)

Renvoie les statistiques(voir Partie 1) sous forme d'une chaîne de caractères

Si pas de données viables, renvoyer "Pas de données"

generateStatistics(0.5:1, 01/01/2023:13h, 01/01/2023:13h)

if unCapteur.pos == 0.5:1 && unCapteur. ⇒ mesure

if tousLesCapteur.pos != 0.5:1 ⇒ pas de mesure

String generateStatistics(pos, r, startTime, endTime)

Renvoie les statistiques(voir Partie 1, pas de capteur à moins de 50km) d'une zone sous forme d'une chaîne de caractères

Si pas de données viables, renvoyer "Pas de données"

vector<Sensor> detectSimilarZones(Sensor aSensor, startTime, endTime)

Renvoie un vecteur de capteurs étant similaire (même moyenne et écart-type) triés par ordre décroissant de similarité.

Renvoie un vecteur vide s'il n'existe pas de capteurs similaires.

(moyenne des données sur le temps passé en paramètre)

Sensor 1 : O3 = 50 NO2 = 45

Sensor 2 : O3 = 49 NO2 = 46 ⇒ les deux capteurs sont similaire

Sensor 1 : O3 = 50 NO2 = 65

Sensor 2 : O3 = 49 NO2 = 70 ⇒ aucun capteurs n’est similaire

double checkEfficiency(AirCleaner)

Renvoie la zone d'éfficacité en mètre du airCleaner

données : Nous avons 5 capteurs disposés chacun à une distance différente du aircleaner. Le cinquième capteur est le plus loin. Pour les quatres premiers capteurs, les mesures sont rédite de 50 %. Or pour le cinquième, elle ne sont pas réduites. Le rayon d’efficacité doit donc se baser sur le capteur 4 avec une fficacité de 50 %

Bool In(position, r)

Renvoie vrai si la position est dans le cercle donné en paramètre

Sinon renvoie faux

données : this.lat : 45 this.long : 0.5

lat = 45 long = 0.7 r = 2000km ⇒ true

lat = 45 long = 0.7 r = 50m ⇒ false