Auteurs : Mathéo JOSEPH, Antonin MARXER, Ambre MYCHALSKI, Florian PAUTRAT

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

AirWatcher - Document de Design

Introduction

Ce document détaille le design de l'application. Nous vous présenterons notre choix d'architecture argumenté ainsi que quelques pseudo-codes d'algorithmes majeurs de notre application. Dans un dernier temps, nous détaillerons les différents tests qui permettront lors de l'implémentation de vérifier le bon fonctionnement de ces algorithmes.

Architecture

Architecture	Avantages	Désavantages
Layered	Évite les services redondants (la couche authentification est commune à tous les services).	Les requêtes sont interprétées par chaque couche.
Model View Controller	Les données peuvent être mises à jour de manière indépendante de l'affichage.	Lourd à mettre en place
Client-server	Réseaux qui permettent aux clients d'accéder aux serveurs. Il n'est pas nécessaire d'implémenter tous les services	Si un service ne fonctionne plus, le serveur ne gère plus les requêtes
Repository	Les composants peuvent être indépendants. Les données sont centralisées et peuvent donc être gérées de manière consistantes	Si le répertoire ne fonctionne plus, tout le système est affecté.
Pipe and filter	La structure suit le workflow des processus. L'ajout de transformation se fait de manière directe. Le système peut être implémenté de manière séquentielle ou concurrente	Chaque transformation spécifie un format d'entrée et de sortie. Chaque transformation doit donc interpréter ses entrées, et retourner des sorties non interprétées. Il risque donc d'y avoir des incompatibilités de structures de données

Tableau 1 : Etude comparative des différentes architectures possibles

Au terme de cette analyse, l'**architecture en couches** apparaît comme la solution la plus intéressante. En effet, notre système a peu de couches différentes et entraîne donc peu de pertes de performance. L'**architecture Client-server** est peu pertinente car l'accès au réseau n'est pas géré par l'application. L'**architecture Pipe and filter** entraîne beaucoup de redondance (au niveau de l'authentification notamment). **L'architecture MVC** nous a paru trop lourde à mettre en place dans le cadre de notre projet.

Notre choix s'est donc porté sur l'architecture en couche, ou Layered architecture.

Décomposition modulaire

Les différents modules de notre architecture sont :

1. **Interface utilisateur :** Console.
2. **Services de logique de l'application :** authentification (simulée), calculer la similarité, calculer la moyenne ATMO dans un cercle, calculer à un point précis, analyser la fiabilité, calculer l'amélioration de l'air, calcul de l'indice ATMO.
3. **Supports systèmes :** Exploitation des fichiers de données, OS.

Diagrammes de classe

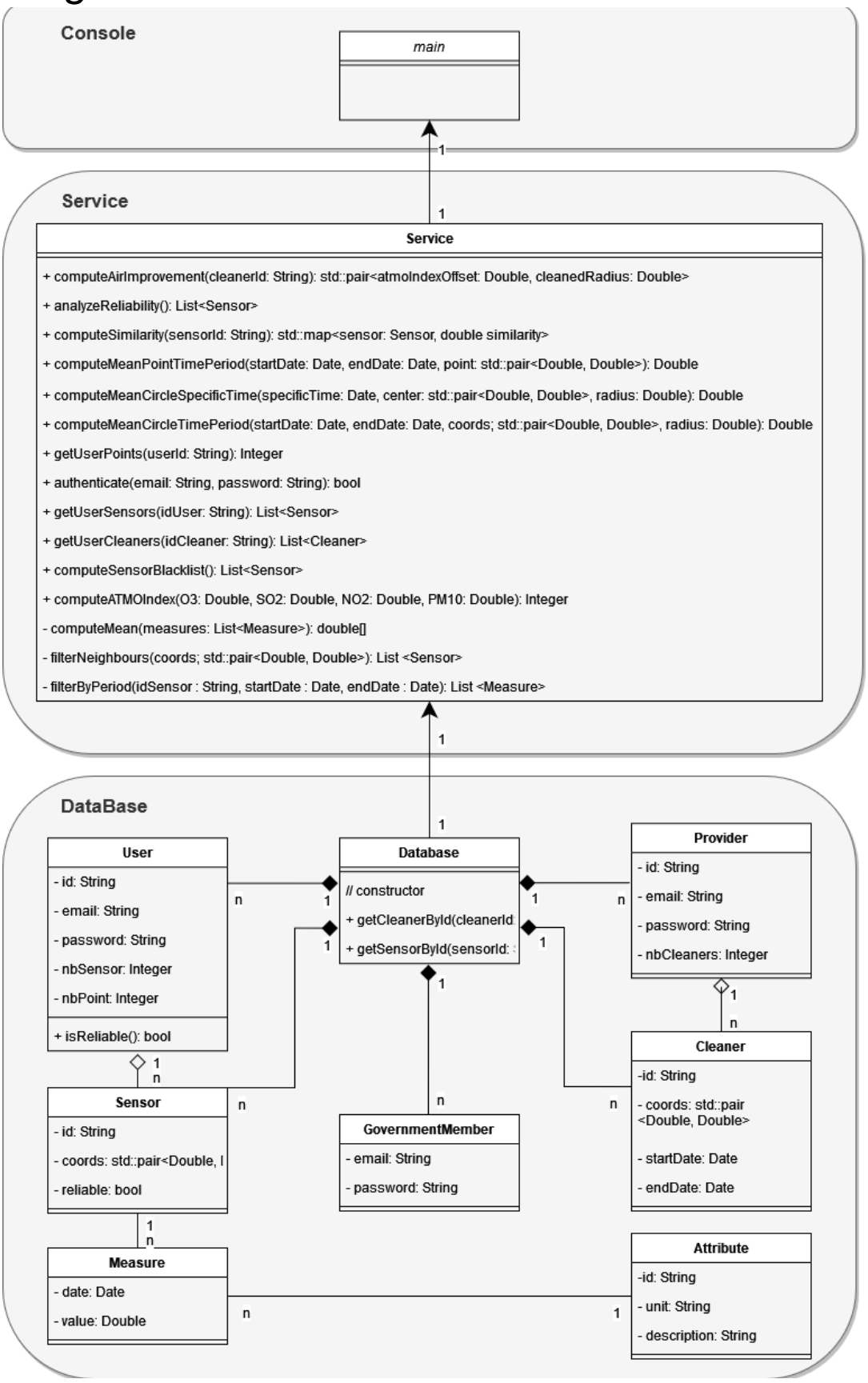

Figure 1 : Diagramme de classe de l'application AirWatcher

Diagrammes de séquence des trois principaux scénarios

A. Diagramme de séquence d'analyzeReliability

Cette fonctionnalité utilise la fonction computeSimilarity dont on détaillera le pseudo-code par la suite.

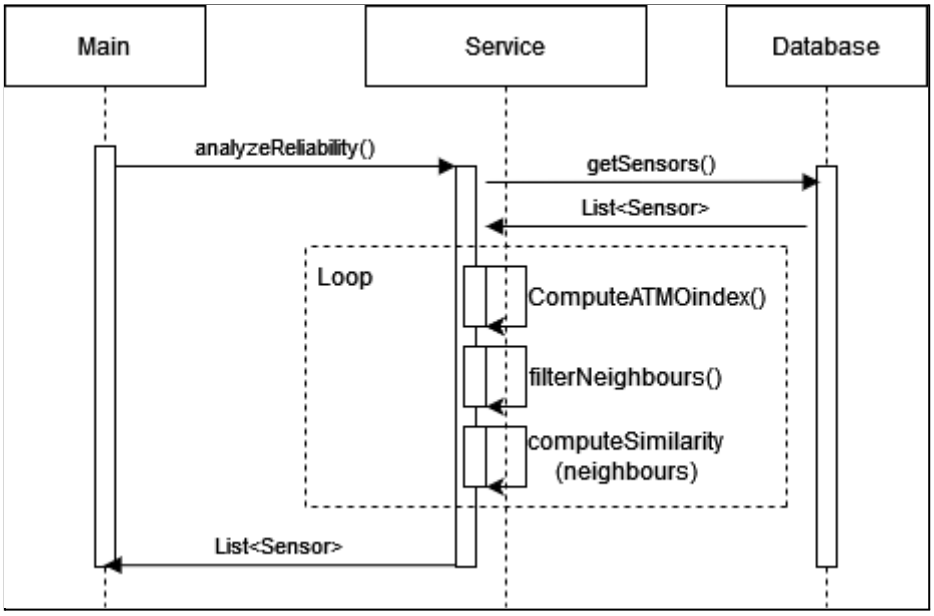

Figure 2 : Diagramme de séquence de la fonction analyzeReliability

B. Diagramme de séquence de meanPointTimePeriod

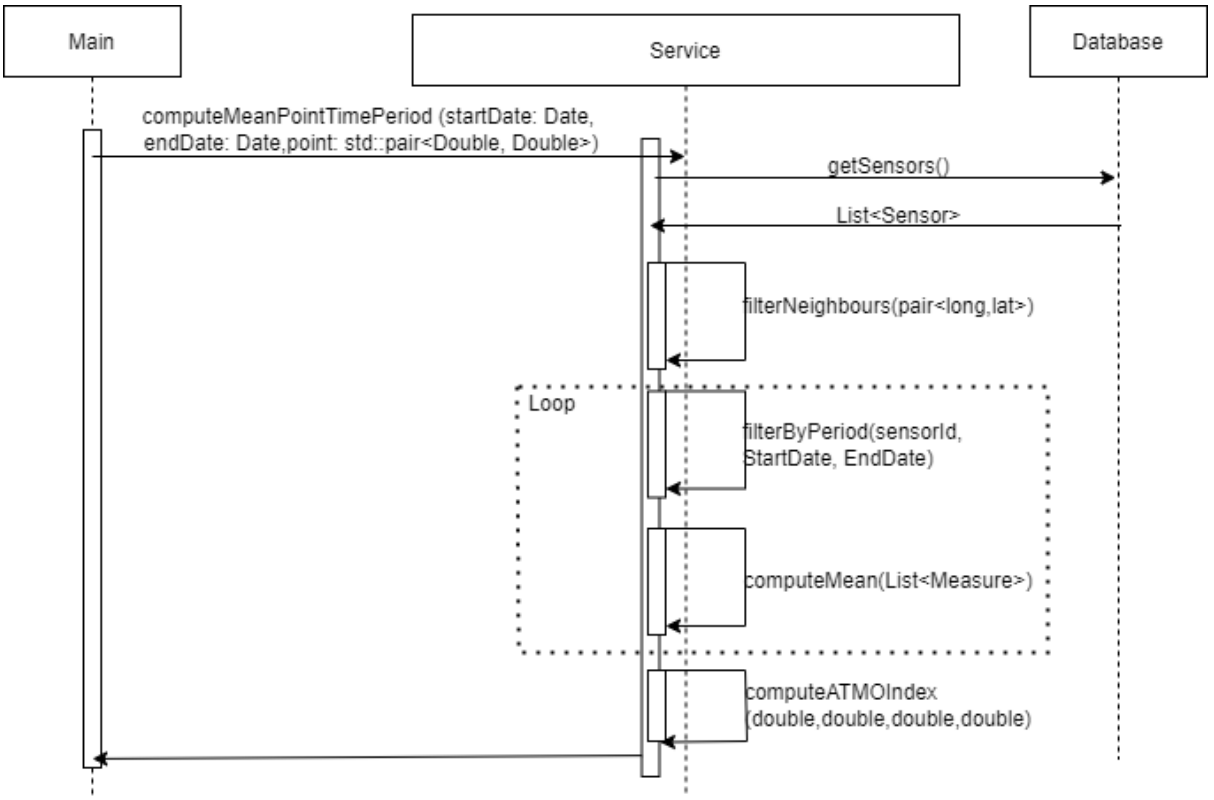

Figure 3 : Diagramme de séquence de la fonction computeMeanPointTimePeriod

C. Diagramme de séquence de computeAirImprovement

computeAirImprovement est une fonctionnalité qui permet de calculer l'amélioration de l'air autour d'un cleaner donné en paramètre. Elle renvoie la valeur du rayon d'air purifié grâce au cleaner et la différence d'indice ATMO maximale que cette purification a engendrée.

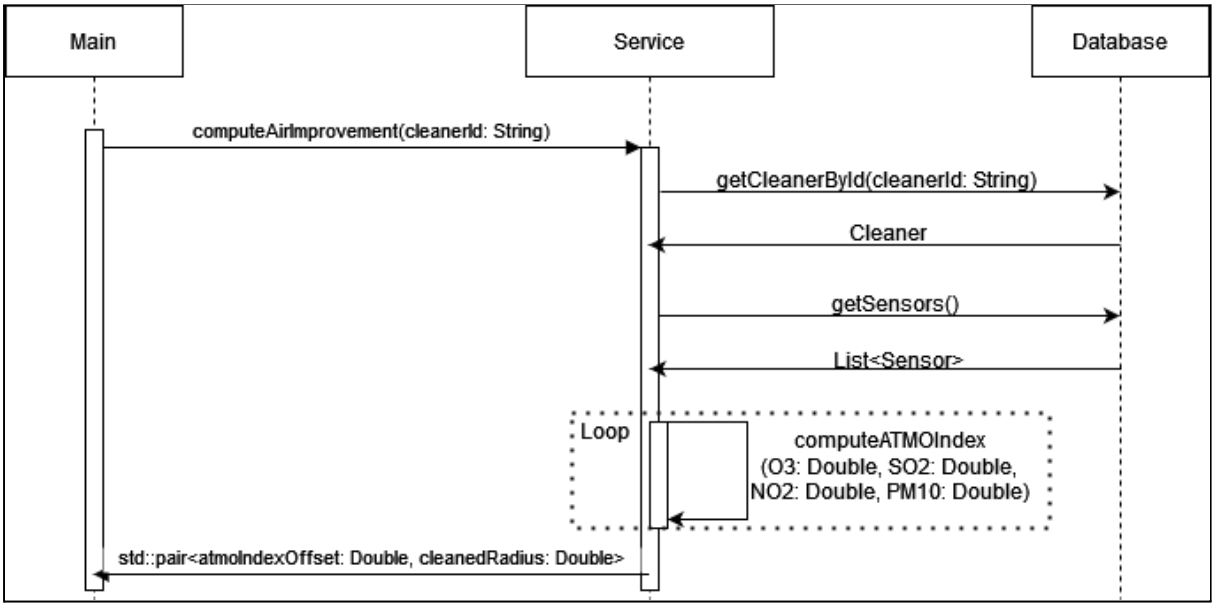

Figure 4 : Diagramme de séquence de la fonction computeAirImprovement

Pour calculer l'impact engendré par le cleaner nous accèderons aux valeurs des différents capteurs juste avant la mise en place du cleaner et également à la fin de sa présence effective. Nous calculerons l'indice ATMO lié à ces différentes mesures et effectuerons la différence entre ces valeurs pré-cleaner et post-cleaner. Plus la différence entre ces indices ATMO est grande, plus le cleaner a eu un impact. Il suffit donc de regarder jusqu'où cette différence est supérieure à un offset donné (jugé offset significatif de purification).

Cependant, nous avons remarqué que deux cleaner pouvaient avoir un impact sur une même zone (intersection entre les zones purifiées de deux cleaner différents). Plutôt que de regarder la plus grande distance supérieure à une différence d'indice ATMO donné, ce qui pourrait conduire à donner un rayon bien supérieur (qui irait jusqu'au bout de la zone purifié d'un autre cleaner cf. figure 7), nous regardons le point le plus proche du cleaner où la différence d'indice ATMO est inférieure à l'offset, donc point jugé non purifié par le cleaner. La distance à ce point (rayon maximum) sera strictement supérieure au rayon de la zone purifiée.

Nous renverrons donc parmi les capteurs compris dans ce rayon maximum et dont la différence d'indice ATMO est supérieure à l'offset la distance maximum au cleaner ainsi que la différence maximum d'indice ATMO qu'on y trouve (plus grande amélioration).

Description et pseudo-code de trois algorithmes majeurs

A. Pseudo-code computeSimilarity

Service computeSimilarity(sensorId, sensorList, startDate, endDate)
Entrées : String sensorId, List <Sensor> sensorList, Date startDate, Date endDate
Sortie : List <Sensor, Double>
Description : Liste des capteurs de sensorList associés à leur similarité par rapport au capteur de référence, sur une période donnée
Déclaration :

Variable	***Usage***
distance	Nombre flottant correspondant à la “distance” entre les valeurs du capteur de référence et d’un autre capteur de la liste
similarity	Nombre flottant normalisé entre 0 et 1. 0 signifie que le capteur de référence n’a aucune similarité avec les capteurs renseignés. 1 signifie qu’ils sont similaires
measureListRef	Liste des mesures du capteur de référence sur la période donnée
meanRefTab	Liste des moyennes de chaque type de mesures du capteur de référence
measureList	Liste des mesures d’un des capteurs de sensorList sur la période donnée
meanTab	Liste des moyennes de chaque type de mesures d’un des capteurs de sensorList
i	Nombre entier

```
DEBUT
measureListRef ← filterByPeriod(sensorId, startDate, endDate)
meanRefTab ← computeMean(measuresListRef)

distanceList ← liste vide
similarityList ← liste vide
distanceMax ← 0
DEBUT Pour chaque sensor dans sensorList faire:
    // Si le capteur est bien distinct du capteur de référence
    DEBUT si sensorId != sensor.getId() faire:
        measureList  ← filterByPeriod(sensor.getId(), startDate, endDate)
        distance ← 0
        DEBUT si measureList n’est pas vide faire:
            meanTab  ← computeMean(measureList)
            DEBUT Pour (i ← 0; i < 4; i ← i + 1) faire:
                // Incrémenter la distance totale par la distance (au carré)
```

```
            // entre les valeurs moyennes mesurées par ce capteur et le capteur de
            // référence, à condition que ces valeurs existent bien
            DEBUT si meanRefTab[i] >= 0 et meanTab[i] >= 0 faire:
                distance ← distance + (meanRefTab[i]-meanTab[i])^2
            FIN si
        FIN pour
    FIN si
    distanceList ← ajouter Pair <sensor, distance^0.5> à la liste
    distanceMax ← max (distance, distanceMax)
  FIN si
FIN pour
// Si tous les capteurs ne sont pas exactement identiques
DEBUT si distanceMax ≠ 0 faire:
  // normaliser les distances entre 0 et 1
  DEBUT Pour chaque elemDistance dans distanceList faire:
    normalizedDistance ← elemDistance.second / distanceMax
    elemDistance.second ← normalizedDistance
  FIN pour
FIN si
DEBUT Pour chaque elemNormalizedDistance dans distanceList faire:
  similarity ← 1 - elemNormalizedDistance.second
  similarityList ← ajouter Pair<elemNormalizedDistance.first, similarity>
FIN Pour
similarityList ← trier par ordre décroissant de similarité
retourner ← similarityList
FIN
```

B. Pseudo-code meanPointTimePeriodè

Service meanPointTimePeriod(startDate, endDate, pair<long,lat>)
Entrées : date startDate, date endDate, pair<double long, double lat>
Sortie : entier indiceATMOPoint
Description : Moyenne de la qualité de l'air à un endroit donné sur une période donnée (pondération sur les mesures puis renvoie de l'indice ATMO)
Déclaration :

Variable	***Usage***
listSensorNeighbours	Liste des capteurs voisins
sommeDistance	somme des distance des capteurs par rapport au point choisi
distance	distance entre un capteur et le point choisi
listMeasure	liste des mesures des capteurs dans une période donnée
measuresMean	Moyenne des mesures d'un capteur par type de mesures

pondération	coefficient de pondération selon la distance entre chaque capteur et le point
O3	Nombre flottant correspondant à la moyenne des mesures d'O3 des capteurs pondérés par leur distance
SO2	Nombre flottant correspondant à la moyenne des mesures de SO2 des capteurs pondérés par leur distance
NO2	Nombre flottant correspondant à la moyenne des mesures de NO2 des capteurs pondérés par leur distance
PM10	Nombre flottant correspondant à la moyenne des mesures de PM10 des capteurs pondérés par leur distance
résultat	Index ATMO moyen

```
DEBUT
    listSensorNeighbours  ← filtrerNeighbours()
    sommeDistance ←  0
    // On trouve la distance totale séparant les sensors voisins du point
    // passé en paramètres
    DEBUT Pour chaque SensorNeighbour faire:
        distance  ← calculer la distance entre le sensor et le point
        sommeDistance ← sommeDistance  + distance
    FIN Pour
    (O3, SO2, NO2, PM10) <- (0, 0, 0, 0)

    // Pour chaque de type de mesure de chaque sensors, on fait leur moyenne sur
    // la période de temps, puis on pondère cette moyenne
    // par leur distance au point passé en paramètre
    DEBUT Pour chaque SensorNeighbour faire:
        listMeasure ← filterByPeriod(sensor.getId(), startDate, endDate)
        measuresMean[] ← computeMean(listMeasure)
        pondération ← (1-( calculer la distance entre le sensor et le point / sommeDistance))/ (nombre de
        voisins-1)
        O3 ← O3 +measuresMean.O3*pondération
        SO2 ← SO2 + measuresMean.S02*pondération
        NO2 ← NO2 + measuresMean.NO2*pondération
        PM10 ←PM10 + measuresMean.PM10*pondération
    FIN Pour
    résultat  ← computeATMOIndex(O3, SO2, NO2, PM10)
   retourner ← résultat
FIN
```

C. Pseudo-code computeAirImprovement

```
Service computeAirImprovement( cleanerId )
```

Entrées : String cleanerId
Sortie : std::pair <atmoIndexOffset : int, cleanedRadius : double>
Description : Calcul de l'amélioration de l'air due aux cleaners et renvoie un périmètre d'air purifié (on trouve le point le plus proche qui n'est pas nettoyé)
Déclaration :

Variable	***Usage***
sensorData	Structure de donnée stockant la différence d'index ATMO et distance au cleaner d'un sensor
oldMeasures	dernières mesures avant la mise en place du cleaner
lastATMOIndex	dernière mesures avant le retrait du cleaner
sensorDataList	Liste contenant les objets sensorData
cleanedRadius	rayon de l'espace purifié par le cleaner
atmoIndexOffset	différence max de deux index ATMO dans la zone purifiée
OFFSET	différence de qualité d'air à partir de laquelle on considère que le cleaner a eu une action significative
maxCleanedRadius	

```
struct SensorData {
        Sensor sensor,
        int diffIndexATMO,
        double distance
};
```

DEBUT

```
    List<SensorData > sensorDataList
    SensorData sensorData
    listSensorInCleanedArea List<SensorData >
    cleaner ← getCleanerById(cleanerId)
    listSensors ← getSensors()

    OFFSET = 2          //choix arbitraire actuellement
    maxCleanedRadius ← distance entre le premier sensor de listSensors et cleaner

    // Instanciation des différents paramètres de listSensor
    DEBUT Pour chaque sensor de listSensors faire :
        // Récupérer les 4 mesures dans un tableau
        oldMeasures ← dernières mesures de sensor datant d'avant cleaner.getStartDate()
        lastMeasures ← dernières mesures de sensor récoltées avant cleaner.getEndDate()
                oldATMOIndex ← computeATMOIndex(oldMeasures)
        lastATMOIndex ← computeATMOIndex(lastMeasures)
        sensorData ← nouvel objet SensorData
        sensorData.sensor ← sensor
```

```
        sensorData.diffIndexATMO ←lastATMOIndex-oldATMOIndex
        sensorData.distance  ← calcul de la distance au carré entre le cleaner et le sensor
        sensorDataList  ← Ajouter sensorData
        // Trouver le rayon maximum d'air purifié (même si le rayon peut être très étendu si il
        // y a une liaison avec la zone d'un autre cleaner)
        DEBUT si sensorData.diffIndexATMO >= OFFSET faire :
          si sensorData.distance>maxCleanedRadius faire :
                  maxCleanedRadius  ← sensorData.distance
          FIN si
        FIN si
    FIN Pour

    // Réduire le rayon max d'air purifié à l'endroit le plus proche d'air non purifié
    DEBUT Pour chaque sensorData de sensorDataList faire :
          DEBUT si OFFSET > sensorData.diffIndexATMO  et
                    si maxCleanedRadius > sensorData.distance faire :
             maxCleanedRadius ← sensorData.distance
          FIN si
    FIN pour

    // Instancier la liste des sensors appartenant à la zone d'air réellement purifié
    DEBUT Pour chaque sensorData de sensorDataList faire :
        DEBUT si OFFSET < sensorData.diffIndexATMO  et
                  si maxCleanedRadius >= sensorData.distance faire :
                  listSensorInCleanedArea ← Ajouter  sensorData
        FIN Si
    FIN pour

    // Valeurs à renvoyer
    cleanedRadius   ← max(listSensorInCleanedArea.distance)
    atmoIndexOffset  ← max(listSensorInCleanedArea.diffIndexATMO)
    return pair<cleanedRadius,atmoIndexOffset>
FIN
```

Tests

A. Test computeSimilarity

Service computeSimlarity(sensorId, sensorList, startDate, endDate)

Données :

Capteurs	***Mesures***
S1 id : sensor1	M1_0 date : 2019-01-01 12:00:00 | attributeId : O3 | value : 50
	M1_1 date : 2019-01-01 12:00:00 | attributeId : SO2 | value : 40
	M1_2 date : 2019-01-01 12:00:00 | attributeId : NO2 | value : 70
	M1_3 date : 2019-01-01 12:45:00 | attributeId : O3 | value : 60
	M1_4 date : 2019-01-01 12:45:00 | attributeId : NO2 | value : 70
S2 id : sensor2	M2_0 date : 2019-01-01 11:30:00 | attributeId : NO2 | value : 60
	M2_1 date : 2019-01-01 11:30:00 | attributeId : PM10 | value : 40
	M2_2 date : 2019-01-01 12:00:00 | attributeId : NO2 | value : 100
S3 id : sensor3	∅
S4 id : sensor4	M4_0 date : 2019-01-01 12:00:00 | attributeId : O3 | value : 100
	M4_1 date : 2019-01-01 12:45:00 | attributeId : O3 | value : 60
	M4_2 date : 2019-01-01 12:45:00 | attributeId : NO2 | value : 70

Tableau 2 : Mesures du test computeSimilarity

Tests :

sensorId	*sensorList*	*startdate*	*endDate*	***résultat attendu***
sensor1	*emptyList*	2019-01-01 11:00:00	2019-01-01 15:00:00	*emptyList*

sensor1	List(S1)	2019-01-01 11:00:00	2019-01-01 15:00:00	*emptyList*
sensor1	List(S2)	2019-01-01 11:00:00	2019-01-01 15:00:00	List(<S2,0>)
sensor1	List(S3)	2019-01-01 11:00:00	2019-01-01 15:00:00	List(<S3,1>)
sensor1	List(S2,S3)	2019-01-01 11:00:00	2019-01-01 15:00:00	List(<S3,1>,<S2,0>)
sensor1	List(S4)	2019-01-01 11:00:00	2019-01-01 15:00:00	List(<S4,0>)
sensor1	List(S4)	2019-01-01 12:45:00	2019-01-01 12:45:00	List(<S4,1>)
sensor1	List(S2,S4)	2019-01-01 11:30:00	2019-01-01 12:00:00	List(<S2,0.8>,<S4,0>)
sensor1	List(S2,S3,S4)	2019-02-01 11:00:00	2019-02-01 15:00:00	List(<S2,1>,<S3,1>,<S4,1>)
sensor1	List(S2,S3,S4)	2019-02-01 15:00:00	2019-02-01 11:00:00	*emptyList*

Tableau 3

Service computeMean(measureList)

Données : *précédentes*

Tests :

measureList	***résultat attendu***
emptyList	[-1,-1,-1,-1]
List(M1_0)	[50,-1,-1,-1]
List(M1_1)	[-1,40,-1,-1]
List(M1_2)	[-1,-1,70,-1]
List(M2_1)	[-1,-1,-1,40]
List(M1_0,M1_1,M1_2,M2_1)	[50,40,70,40]
List(M1_0,M1_3,M4_0,M1_2,M1_4,M4_2)	[70,-1,70,-1]

Tableau 4 : Résultat attendu des tests de computeMean

B. Test computeMeanPointTimePeriod

TEST 1 : Point dans la zone couverte par les capteurs

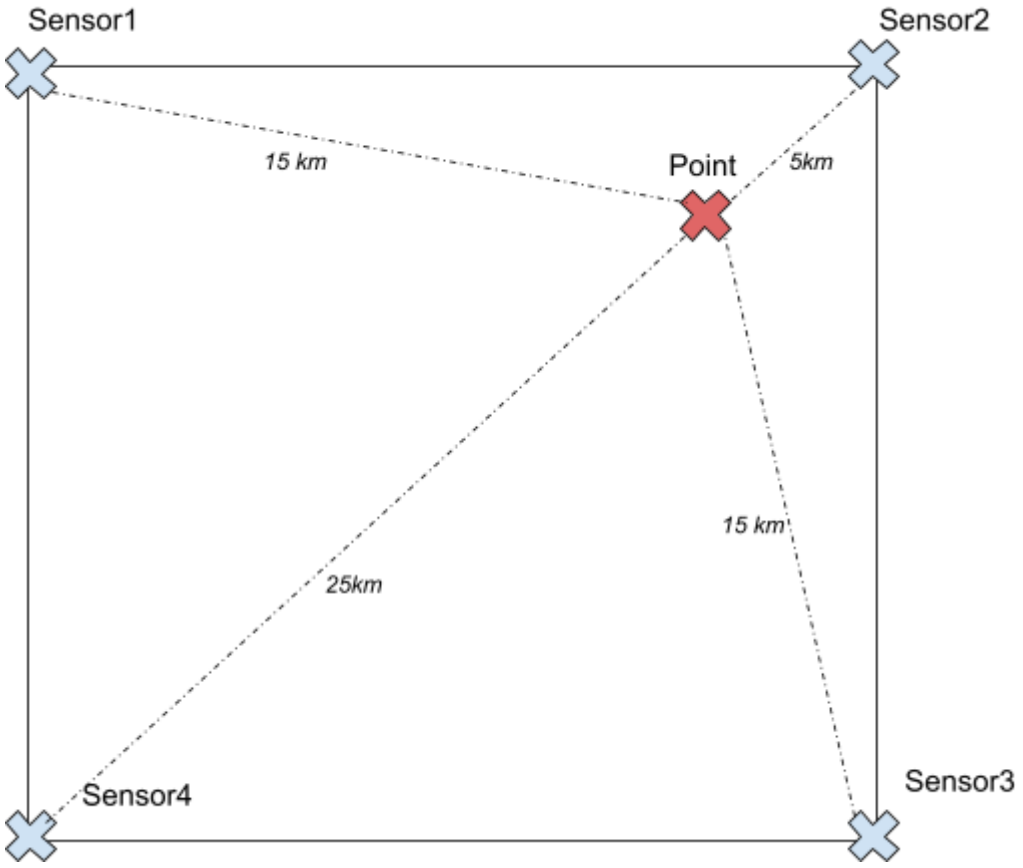

Figure 5 : Schéma du positionnement des capteurs autour du point choisi

SensorID	O3	SO2	NO2	PM10
Sensor1	0	0	0	0
Sensor1	50	50	50	0
Sensor2	100	100	100	40
Sensor2	100	100	100	40
Sensor3	200	200	200	0
Sensor3	150	150	150	0
Sensor4	300	300	300	60
Sensor4	300	300	300	60

Tableau 5 : Mesures des capteurs 1 à 4 sur la période donnée

Moyennes par type des mesures du capteur 1:
O3: 25 SO2: 25 NO2: 25 PM10: 0
Moyennes par type des mesures du capteur 2:
O3: 100 SO2: 100 NO2: 100 PM10: 40
Moyennes par type des mesures du capteur 3:
O3: 175 SO2: 175 NO2: 175 PM10: 0
Moyennes par type des mesures du capteur 4:
O3: 300 SO2: 300 NO2: 300 PM10: 60

Donc les moyennes par type des mesures une fois pondérées sont:
O3: 162,5 SO2: 162,5 NO2: 162,5 PM10: 23,81

Résultat attendu :
indice ATMO : 7

TEST 2 : Point en dehors de la zone couvert par les capteurs

Résultat attendu :
indice ATMO : -1

TEST 3 : Période de temps pour laquelle il n'y a pas de mesures

Résultat attendu :
indice ATMO : -1

C. Test computeAirImprovement

TEST 1 : Cleaner fonctionnel

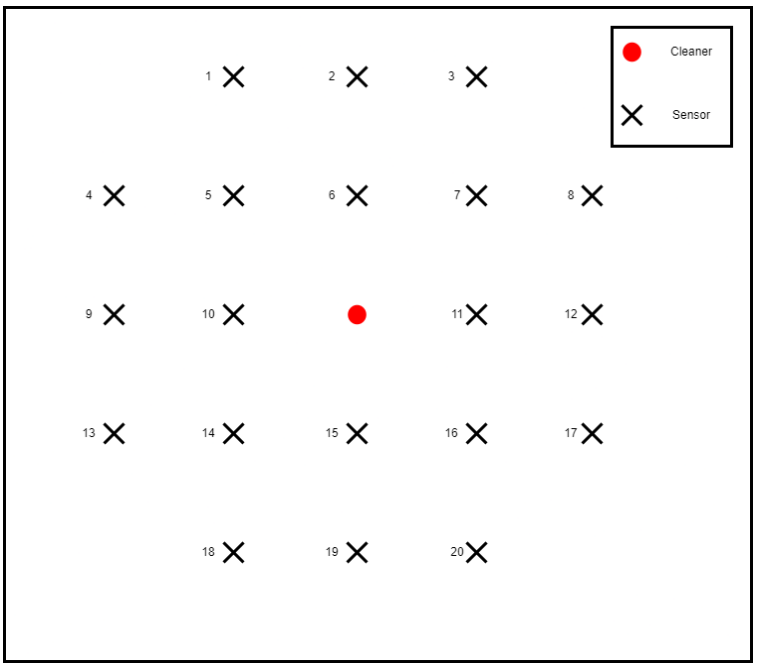

Figure 6 : Répartition des sensors et du cleaner étudié

- **Informations sur le cleaner :**
  StartDate (ou date de mise en fonctionnement) : 01/01/2019
  EndDate (ou date de fin du fonctionnement) : 01/03/2019
  coordonnées : (0.0 , 0.0)

- **Informations sur les sensors :**
  chaque sensor est disposé selon un quadrillage et distants de 10 km

Mesures prises le 31/12/2018 à 12:00
(mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index
Sensor 1 à 20	141	230	154	39	6

Tableau 6 : Mesures des capteurs 1 à 20 avant la mise en route du cleaner (test 1)

Mesures prises le 28/02/2019 à 12:00
(mesures prises à la fin de la période de fonctionnement du cleaner, et calcul de la différence d'indice ATMO avec les mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index	Différence d'indice ATMO
Sensor 1 à 3	90	137	92	25	4	2
Sensor 4	110	177	125	31	5	1
Sensor 5 à 7	60	90	64	17	3	3
Sensor 8 à 9	110	177	125	31	5	1
Sensor 10 à 11	60	90	64	17	3	3
Sensor 12 à 13	110	177	125	31	5	1
Sensor 14 à 16	60	90	64	17	3	3
Sensor 17 à 20	110	177	125	31	5	1

Tableau 7 : Mesures des capteurs 1 à 20 à la fin du fonctionnement du cleaner (test 1)

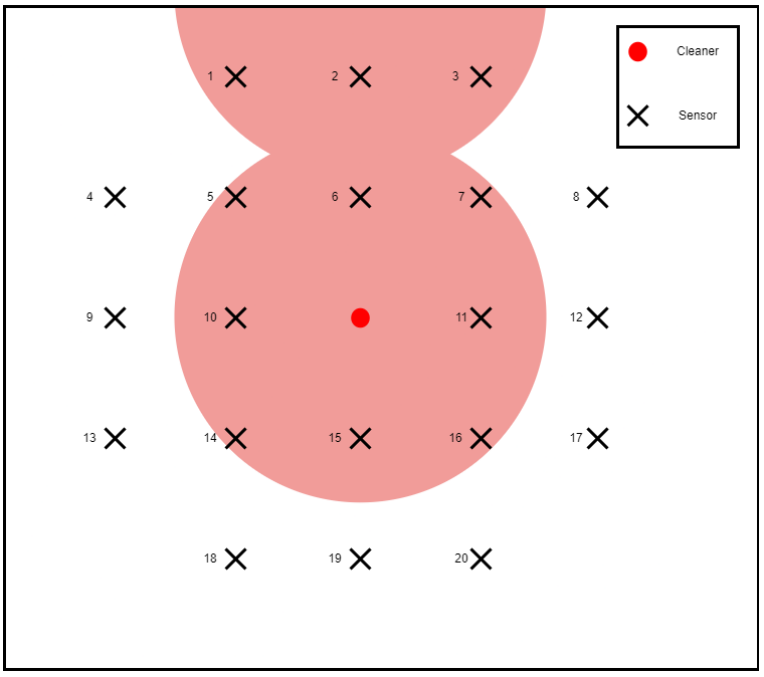

Figure 7 : En rouge clair : zone correspondant à une différence d'indice ATMO de plus de 2

Nous remarquons que le rayon de la zone purifiée par le cleaner sélectionné ne s'étend pas au delà des sensors 5, 7, 16 et 14 et qu'une nouvelle zone purifiée par un autre cleaner intersecte la zone que nous cherchons à établir. D'où l'utilité du maxCleanedRadius.

RESULTATS DU TEST :

cleanedRadius ← 14.1 // en km
atmoIndexOffset ← 3

TEST 2 : Cleaner non fonctionnel

- **Informations sur le cleaner :**
  StartDate (ou date de mise en fonctionnement) : 01/01/2019
  EndDate (ou date de fin du fonctionnement) : 01/03/2019
  coordonnées : (0.0 , 0.0)

- **Informations sur les sensors :**
  chaque sensor est disposé selon un quadrillage et distants de 10 km

Mesures prises le 31/12/2018 à 12:00

(mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index
Sensor 1 à 20	141	230	154	39	6

Tableau 8 : Mesures des capteurs 1 à 20 avant la mise en route du cleaner (test 2)

Mesures prises le 31/12/2018 à 12:00

(mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index	Différence d'indice ATMO
Sensor 1 à 20	141	230	154	39	6	0

Tableau 9 : Mesures des capteurs 1 à 20 à la fin du fonctionnement du cleaner (test 2)

Nous remarquons que le cleaner ne fonctionne pas car il n'y a aucune amélioration de l'indice ATMO.

RESULTATS DU TEST :

cleanedRadius ← 0
atmoIndexOffset ← 0

TEST 3 : Cleaner non fonctionnel, et la qualité de l'air se dégrade

- **Informations sur le cleaner :**
  StartDate (ou date de mise en fonctionnement) : 01/01/2019

EndDate (ou date de fin du fonctionnement) : 01/03/2019
coordonnées : (0.0 , 0.0)

- **Informations sur les sensors :**
  chaque sensor est disposé selon un quadrillage et distants de 10 km

Mesures prises le 31/12/2018 à 12:00

(mesures prises à la fin de la période de fonctionnement du cleaner, et calcul de la différence d'indice ATMO avec les mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index	Différence d'indice ATMO
Sensor 1 à 3	90	137	92	25	4	2
Sensor 4	110	177	125	31	5	1
Sensor 5 à 7	60	90	64	17	3	3
Sensor 8 à 9	110	177	125	31	5	1
Sensor 10 à 11	60	90	64	17	3	3
Sensor 12 à 13	110	177	125	31	5	1
Sensor 14 à 16	60	90	64	17	3	3
Sensor 17 à 20	110	177	125	31	5	1

Tableau 10 : Mesures des capteurs 1 à 20 à la fin du fonctionnement du cleaner (cf image du test 1)

Mesures prises le 28/02/2019 à 12:00

(mesures prises avant la mise en fonctionnement du cleaner)

ID Sensors	O_3	SO_2	NO_2	PM_{10}	ATMO Index
Sensor 1 à 20	141	230	154	39	6

Tableau 11 : Mesures des capteurs 1 à 20 avant la mise en route du cleaner (cf image du test 1)

RESULTATS DU TEST :

cleanedRadius ← 0 // en km

atmoIndexOffset ← **-3**
