Authors: DRUTEL – GALLE – GUILLEVIC – RASCOUSSIER

AirWatcher – Design Document

This is the Design Document for the team composed of Maxence DRUTEL, Romain GALLÉ, Marie GUILLEVIC and Florian RASCOUSSIER.

Summary

Authors: DRUTEL – GALLE – GUILLEVIC – RASCOUSSIER

I. Application Architecture

I.1. Architecture Design

We plan to design our application using the Model View Controller model (MVC). Storage classes (sensor data, user data…) will be stored in the Model package. A Controller class will oversee the execution our algorithms and our core commands. Finally, a Terminal class is responsible for recognizing user's command and displaying the application's outputs.

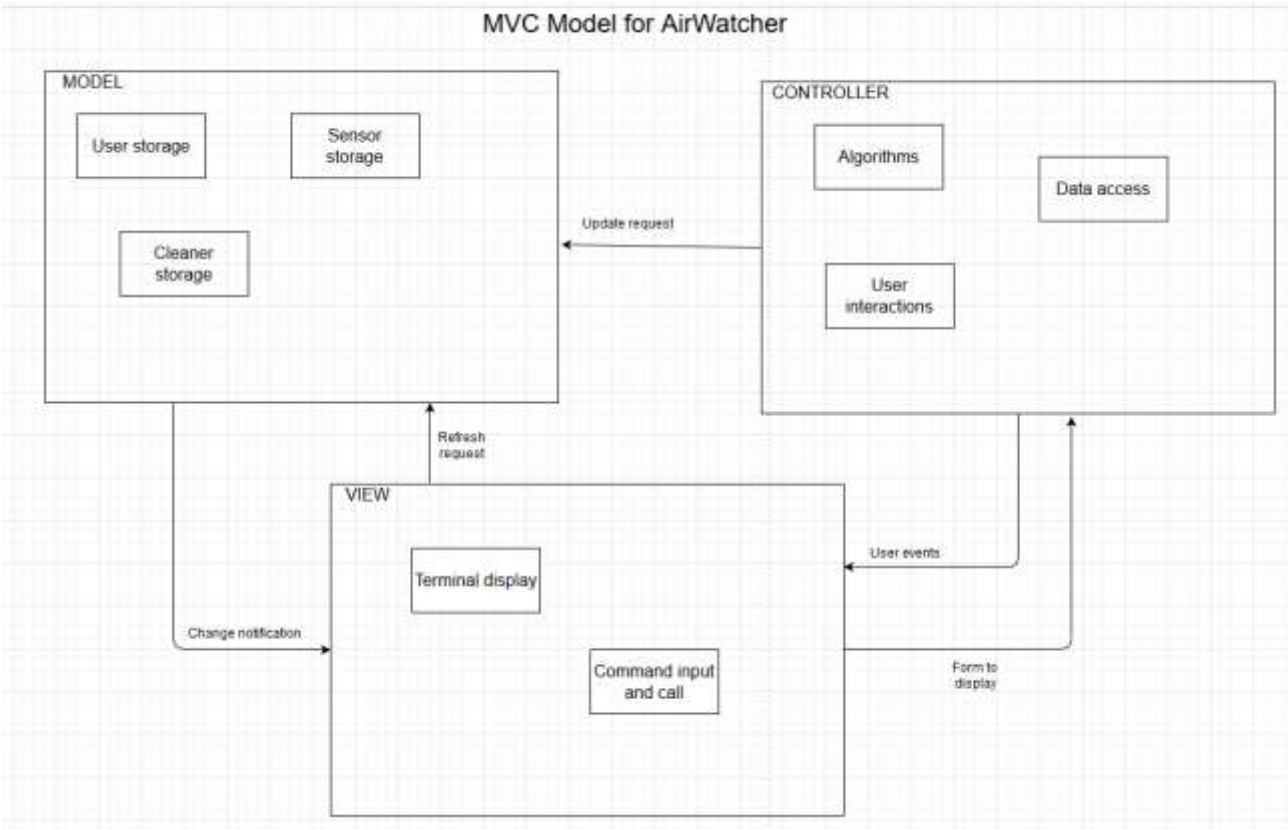

Figure 1 - MVC Architecture of AirWatcher

I.2. Class Diagram

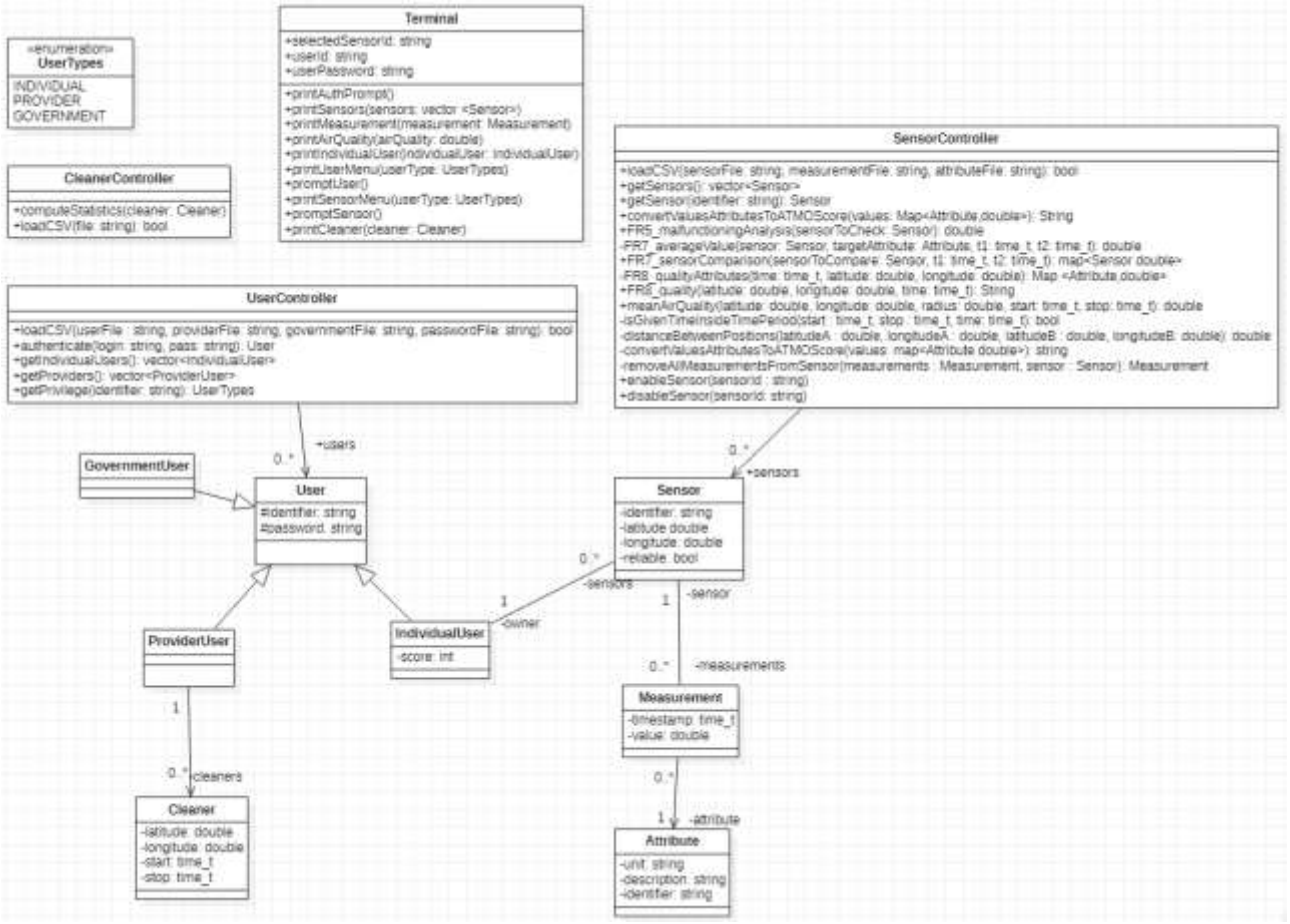

Figure 2 - Class Diagram of AirWatcher

II. Example scenarios and sequence diagrams

II.1. Scenario 1 – A private user needs to compare sensors

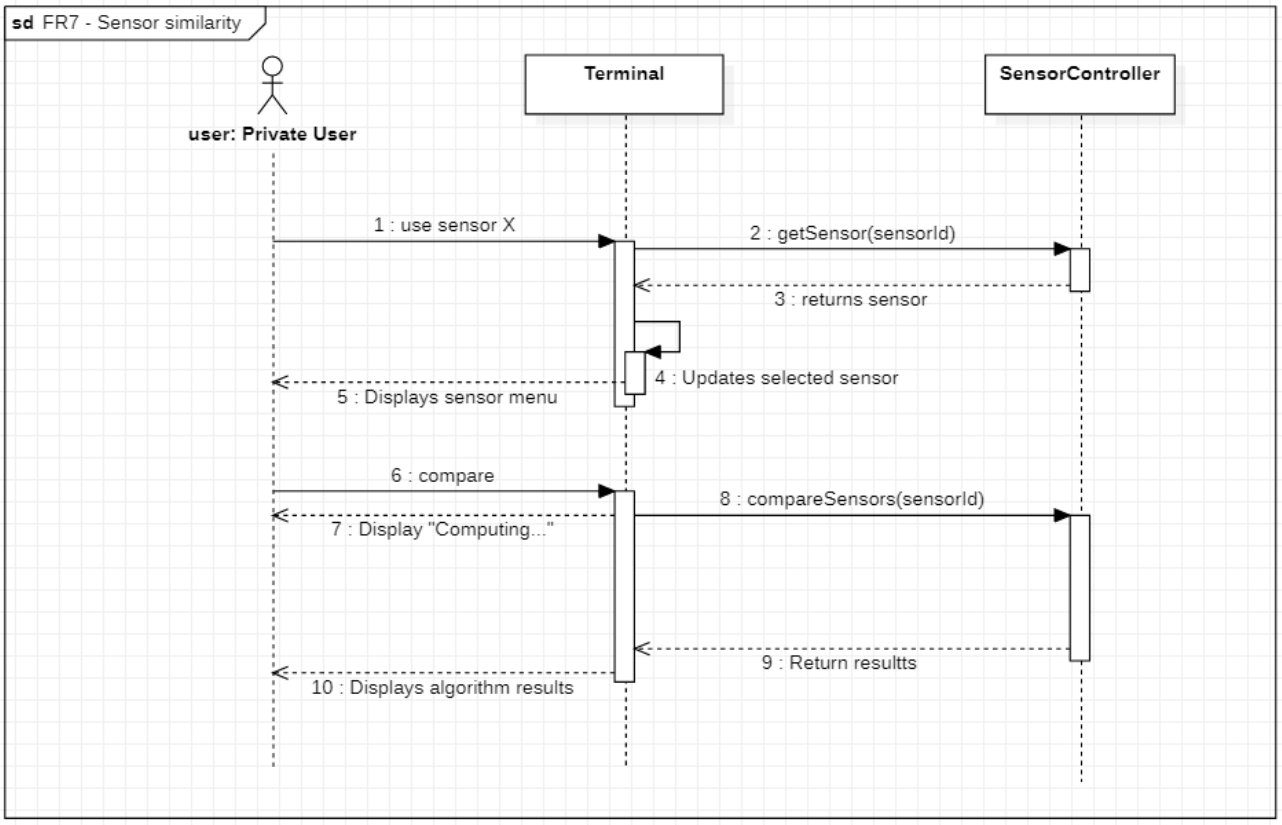

Figure 3 - A private user wants to compare sensors

II.2. Scenario 2 – Various users need to access the list of air cleaners

A user wants to get the list of air cleaners. According to his privileges, the user will be denied if he is private, access the list of owned cleaners if he is a provider, or access the list of all air cleaners if he is from government.

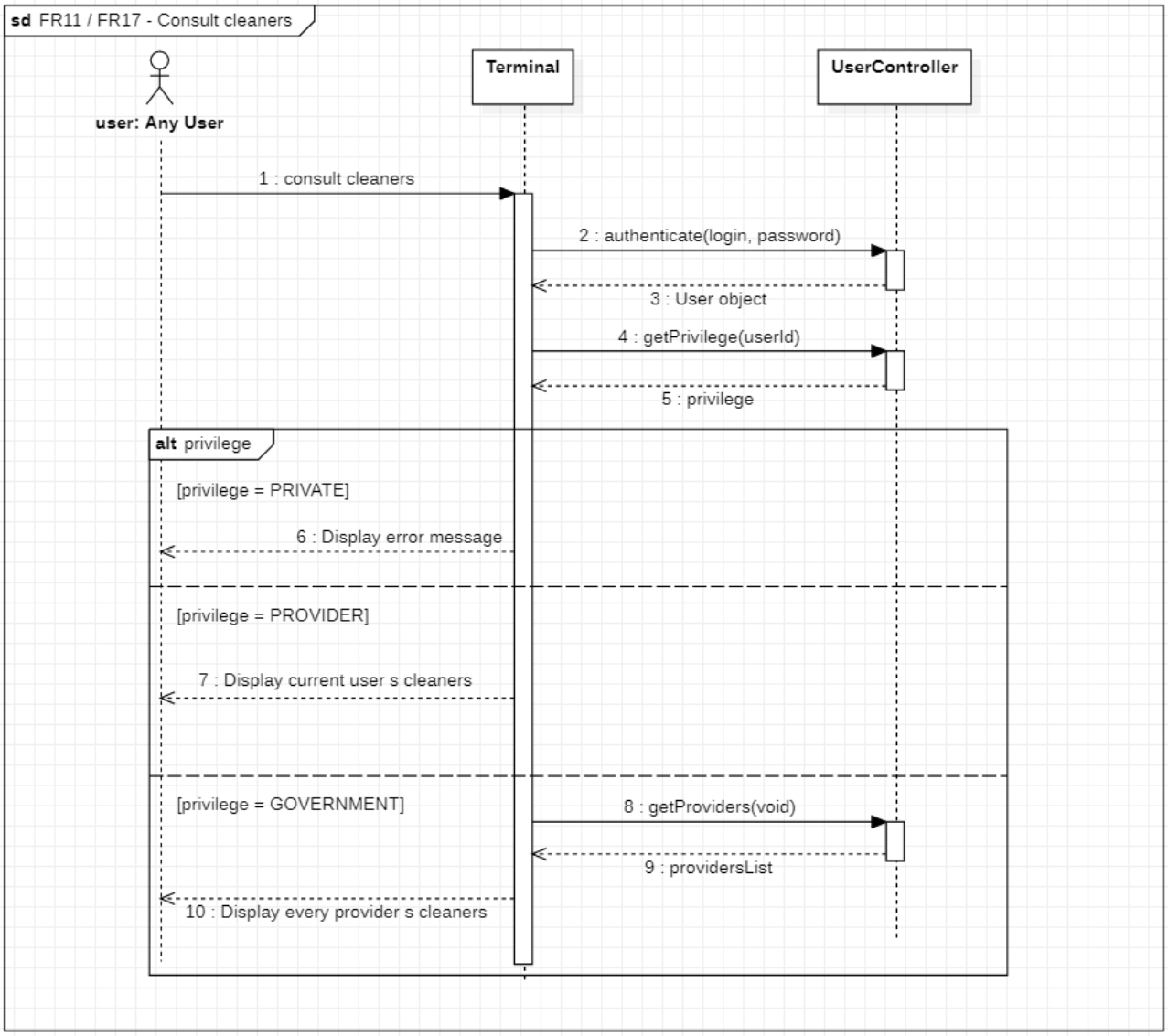

Figure 4 - Various users need to access air cleaners

II.3. Scenario 3 – A government user needs to mark a sensor unreliable

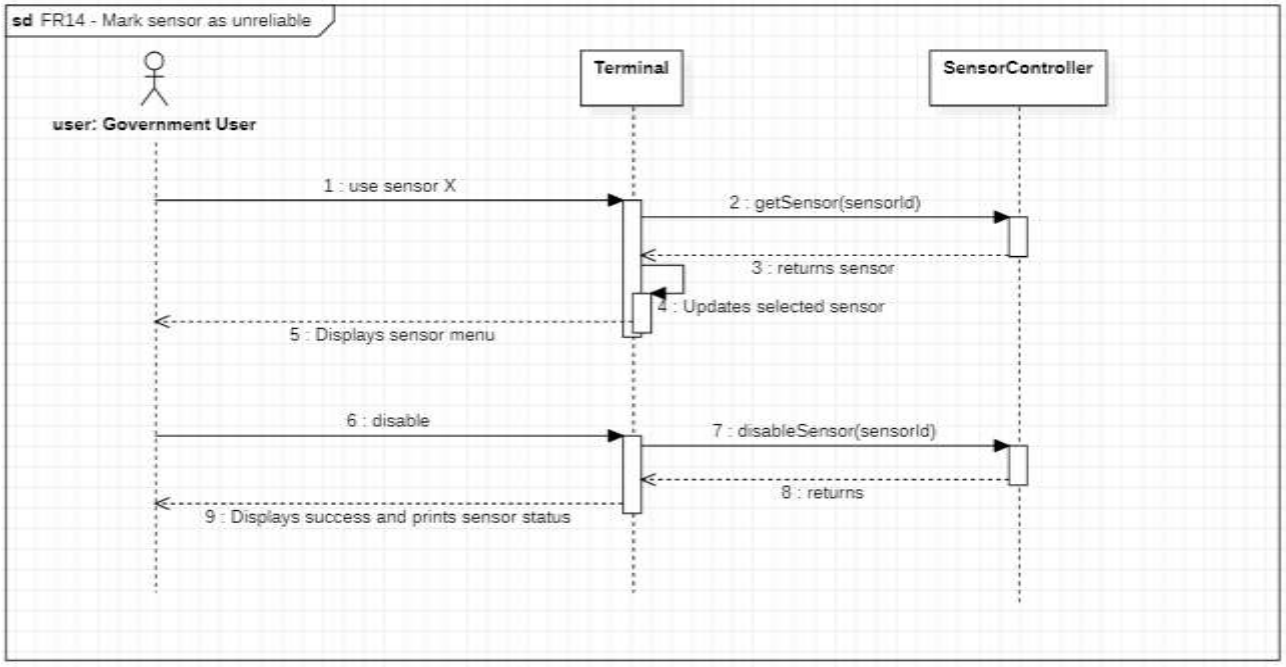

Figure 5 - A government user needs to mark a sensor unreliable

III. Algorithms' description

Our code will work based on 3 core algorithms complying to the functional requirements of the specification document.

III.1. FR8 – Determine the quality of air at a given time and location

The goal is to determine the ATMO quality of air at a given time and location, based on the neighbouring's sensors records. We will proceed by estimating the concentration of the 4 attributes (O3, SO2, NO2 and PM10) at the target location from all measurements made up to 24 hours before of after the target time.

We will then weight these measurements by their relative distance: the closer the sensor is, the more it will count. To proceed, we will take a random sensor and calculate its distance for the target location: it will be our referential and have a weight of 1. The weight of all other sensors will then be inversely proportionate to the distance from the target location: $weight_{sensor} = \frac{distance_{referencial}}{distance_{sensor}}$. If the referential is at 100m of the target location, a sensor located at 50m will have a weight of 2, and a sensor at 300m a weight of 1/3.

Finally, we determine the weighted average of measurements for each attribute from our determined weights. We can then apply the ATMO quality of air formula (https://fr.wikipedia.org/wiki/Indice_de_qualit%C3%A9_de_l%27air) to return the quality of air of the location at the specified time.

The worst-case complexity of this algorithm is **O(m)** where 'm' is the number of measurements.

```
algorithm FR8_quality is
    inputs: Period timePeriod, Position askedPosition
    output: String predictedATMOScore
    call: FR8_quality(Period timePeriod, Position askedPosition)
{
    // returns True if the timestamp is inside the period, False otherwise.
    function isGivenTimeInsideTimePeriod(Pediod period, Time timestamp) -> Boolean isInside
        if period.start <= timestamp and period.end >= timestamp
            return true
        else
            return false
```

```
    // returns the distance between 2 positions
    function distanceBetweenPositions(Position a, Position b) -> double distance

    // get a random Sensor, the fastest to get actually
    function getASensor() -> Sensor sensor

    // convert
    function convertValuesAttributesToATMOScore(Map<Attribute, double> values) -> String predictedATMOScore

    // get a list of all measurements
    function getAllMeasurements() -> Measurement[] allMeasurements

    // returns predicted values for ATMO attributes for a given position and a considered period of time for the data
    function FR8_qualityAttributes(Period timePeriod, Position askedPosition) -> Map<Attribute, double> attributesPredictedValues
        var Attribute[] attributes := getAllAttributes() // an array of all data types (Attibutes)
        var Measurement[] allMeasurements := getAllMeasurements()

        var Map<Attribute, double> numeratorSums := {
            attributes.O3: 0.0,
            attributes.NO3: 0.0,
            attributes.SO2: 0.0,
            attributes.PM10: 0.0
        }
        var Map<Attribute, double> denominatorSums := {
            attributes.O3: 0.0,
            attributes.NO3: 0.0,
            attributes.SO2: 0.0,
            attributes.PM10: 0.0
        }
```

```
        var Sensor referentiel := getASensor()

        for each measurement in allMeasurements
            if measurement.getSensor().reliable and isGivenTimeInsideTimePeriod(timePeriod, measurement.timestamp)
                var double coefficient := distanceBetweenPositions(referentiel.position, measurement.getSensor().position)
                numeratorSums[measurement.attribute] := numeratorSums[measurement.attribute] + (coefficient * measurement.value)
                denominatorSums[measurement.attribute] := denominatorSums[measurement.attribute] + coefficient

        var Map<Attribute, double> attributesPredictedValues := {
            attributes.O3: 0.0,
            attributes.NO3: 0.0,
            attributes.SO2: 0.0,
            attributes.PM10: 0.0
        }
        attributesPredictedValues := numeratorSum/denominatorSums

        return attributesPredictedValues

    // returns a ATMO coefficient for a given position and a considered period of time for the data
    function FR8_quality(Period timePeriod, Position askedPosition) -> String predictedATMOScore
        var Map<Attribute, double> results := FR8_qualityAttributes
        return convertValuesAttributesToATMOScore(results)

}
```

III.2. FR5 – Help determine if a sensor is reliable

The goal is to help determine if a private sensor is reliable or not. This algorithm will use our method designed for FR8 to predict values at a desired location and time based on neighbouring sensors' data. We will thus create a list of all sensors excepting the one we want to analyse, and for each measurement of the analysed sensor, call the FR8 algorithm for the position of the sensor and the moment of each measurement.

We will then calculate the relative gap between the predicted value and the effective value measured by the sensor, which formula is:

$$relative\ gap = \frac{|measuruedValue - predictedValue|}{predictedValue}$$

Finally, we will sum these relative gaps for each measurement of the tested sensor and calculate an average relative gap and return it. This average relative gap represents the overall gap of the tested sensor's measurements compared to the expected value from all other sensors. The government agency can then judge if the sensor will be marked at unreliable or not.

The worst-case complexity of this algorithm is **O(m²)**.

```
Algorithm FR5_malfunctioningAnalysis is
    input: Sensor sensorToCheck
    output: Double averageRelativeGap
    call: FR5_malfunctioningAnalysis(Sensor sensorToCheck)
{
    // get a list of all measurements
    function getAllMeasurements() -> Measurement[] allMeasurements

    // removes all the measurements from a measurement list of a given sensor
    function removeAllMeasurementsFromSensor(Measurement[] measurements, Sensor sensor) -> Measurement[] remainingMeasurements

    function FR5_malfunctioningAnalysis(Sensor sensorToCheck) -> Boolean isReliable
        var Measurement[] measurements := getAllMeasurements()
        measurements := removeAllMeasurementsFromSensor(measurements, sensorToCheck)

        var Double relativeSum := 0.0
```

```
        var Integer nbOfMeasurementsForSensorToCheck := 0

        // for every measurement of the sensor, check if it is close to the expected one or not by adding to relative sum
        for each measurement in sensorToCheck.getMeasurements()
            var Map<Attribute, Double> expectedValues := FR8_qualityAttributes(ALWAYS, measurement.getSensor().position)
            var Double expectedValue := expectedValues[measurement.attribute]
            var Double relativeDiff := abs(expectedValue - measurement.value) / expectedValue

            relativeSum := relativeSum + relativeDiff
            nbOfMeasurementsForSensorToCheck := nbOfMeasurementsForSensorToCheck + 1

        var Double averageRelativeGap := relativeSum / nbOfMeasurementsForSensorToCheck

        return averageRelativeGap
}
```

III.3. FR7 – Find the level of similarity between one specified sensor and all the others for a specific period

The goal is to estimate a level of similarity between one specified sensor and all the others for a specific period. A level of similarity of 1 means the two sensors found the same values for the period, while a similarity tending towards 0 means the two sensors had their measurements with a gap close to infinity.

To proceed, we will reuse our principle of relative gap used for the FR5 algorithm: for each Attribute, we will calculate their average value for all sensors and find the relative gap between the target sensor's average and all other sensors using the previous formula. We will then finally average the relative gaps of all attributes to get a global relative gap between the specified sensor and the others.

Finally, we will have to determine a level of similarity, not a relative gap. It means finding a function for which $f(0) = 1$ (a relative gap of zero means the sensors are 100% identical) and $\lim_{x \to +\infty} f(x) = 0$. A fitting candidate would be an inverse function matching $f(0) = 1$, thus:

$$similarity = \frac{1}{relative\ gap + 1}$$

We then store the level of similarity in a map along with its linked sensor and return it to the user.

The worst-case complexity of this algorithm is **O(max(s,m))** where 's' is the number of sensors and 'm' the number of measurements.

```
Algorithm FR7_sensorComparison is
    inputs: Sensor sensorToCompare, timestamp t1, timestamp t2
    output: Map<Sensor,double> proximity
    Pre-condition: sensorToCompare has measurements during the specified period
    call: FR7_sensorComparison(Sensor sensorToCompare, timestamp t1, timestamp t2)
{
    function getAllSensors() -> Sensor[] allSensors

    // Calculates the average value of a targetted attribute of a sensor between t1 and t2
    function FR7_averageValue(Sensor sensor, Attibute targetAttribute, timestamp t1, timestamp t2)
        var Double sum := 0
```

```
        var Integer checkedMeasurement :=0
        for each measurement in sensor.measurements
            if measurement.attribute.identifier = targetAttribute.identifier and measurement.timestamp > t1 and measurement.timestam
p < t2
                sum += measurement.value
                checkedMeasurement+=1

        return sum/checedMeasurement



    //Calculates the proximity of all sensors compared to the targetted sensor: Calculates the relative gap between the average targ
et value and the other sensors. Returns a map of all sensors with their target's proximity
    function FR7_sensorComparison(Sensor sensorToCompare, timestamp t1, timestamp t2) {

        var Map<Sensor, double> proximity

        var Double refValues[4]; //Stores the average value of the target sensor
        var Integer i:=0
        for each attribute //NO2, O3, PM10...
            refValues[i] := FR7_averageValue(sensorToCompare,attribute,t1,t2)
            i+=1

        for each sensor in getAllSensors()
            if sensor!=sensorToCompare
                var Double relative_gap := 0
                var Integer i := 0
                for each attribute
                    var Double average := FR7_averageValue(sensor,attribute,t1,t2)
                    relative_gap += absolute(average-refValues[i])/refValues[i]
                    i+=1
```

```
            relative_gap := relative_gap/i

            proximity[sensor] = 1/(relative_gap+1)

    return proximity
}
```

IV. Test plan

To execute the different tests, we created new csv files dedicated to the tests. We choose to take only 2 different sensors 0 and 1 and took one measurement for each of them. We also created a file for the passwords and for the government's logins.

IV.1. Unitary Tests

IV.1.a. LoadService functions

a- LoadUsers

Tested Function	Test description / input	Expected result
loadService.loadUser(string userFile, string providerFile, string governmentFile, string passwordFile)	Paths to the following CSV files: users.csv providers.csv government.csv passwords.csv	true
Additional Test: Verification of the obtained user list after loading the different files	Size of userController.users should be equal to the sum of the entries in users.csv, providers.csv and government.csv	6
Additional Test: entry integrity check	Check that list entries are not empty. Check that first and last entries contains valid users, and all have password differing from null or string.empty()	N/A

Tested Function	Test description / input	Expected result
loadService.loadUsers(string userFile , string providerFile, string governmentFile, string passwordFile)	Non-exisiting files and empty / not valid files	false

b- LoadSensors

Tested Function	Test description / input	Expected result
loadService.loadSensor(string sensorFile, string measurementFile, string attributeFile)	Path to the following CSV files: sensors_test.csv, measurements_test.csv, attributes.csv	true
Additional Test: Verification of the obtained sensors list after loading the different files	sensorController.sensors.size()	2
Additional Test: Verification of the obtained measurements list after loading the different files	Check the total number of measurements (sum of the number of measurements for each sensor)	8

Tested Function	Test description / input	Expected result
loadService.loadSensor (string sensorFile, string measurementFile, string attributeFile)	Non-existing or invalid files	false
Additional Test: check sensors list integrity	Empty list before test. Check that sensorController.sensors list is empty	N/A

c- LoadCleaners

Tested Function	Test description / input	Expected result
loadCleaners.loadCleaners(string file)	Path to the cleaners.csv file	true
Additional Test: Verification of the obtained cleaners list after loading the different files	Check the total number of cleaners (sum of the number of cleaners for each provider)	2

Tested Function	Test description / input	Expected result
loadCleaners.loadCleaners(string file)	Non-existing of invalid file	false
Additional Test: check cleaners list integrity	Empty lists before previous test. Check that all the providers have no cleaners registered.	N/A

IV.1.b. UserService functions

a- Authentication

Tested Function	Test description / input	Expected result
userController.authenticate(string login, string pass)	Provider0 provider0	User object with identifier provider0 and cleaner object identifier Cleaner0

Tested Function	Test description / input	Expected result
userController.authenticate(string login, string pass)	Provider5 provider0	Null

b- GetIndividualUsers

Tested Function	Test description / input	Expected result
userController.getIndividualUsers()	void	Vector of IndividualUsers objects with identifiers : User0 and User1

c- GetProviders

Tested Function	Test description / input	Expected result
userController.getProviders()	void	Vector of Providers objects with identifiers : Provider0 and Provider1

d- GetPrivilege

Tested Function	Test description / input	Expected result
userController.getPrivilege(string identifier)	User0	INDIVIDUAL

Tested Function	Test description / input	Expected result
userController.getPrivilege(string identifier)	Provider0	PROVIDER

Tested Function	Test description / input	Expected result
userController.getPrivilege(string identifier)	Government0	GOVERNMENT

Tested Function	Test description / input	Expected result
userController.getPrivilege(string identifier)	User8	NONE

IV.1.c. SensorController functions

a- GetSensors

Tested Function	Test description / input	Expected result
sensorController.getSensors( )	void	Vector of Sensor objects with identifiers: Sensor0 and Sensor1

b- GetSensor

Tested Function	Test description / input	Expected result
sensorController.getSensors(string identifier)	Sensor0	Sensor object with identifiers: Sensor0, latitude: 44 and longitude : -1

Tested Function	Test description / input	Expected result
sensorController.getSensors(string identifier)	Sensor101	null

c- Malfunctionning analysis

1- FR5_malfunctioningAnalysis(in sensorToCheck:Sensor): double

Tested Function	Test description / input	Expected result
sensorController. FR5_malfunctioningAnalysis(Sensor sensorToCheck)	sensor0	0.184

d- Mean Air Quality

Tested Function	Test description / input	Expected result
sensorController. meanAirQuality(double latitude, double longitude, double radius, time_t start, time_t stop)	45 -2 5 01/01/2019 12:00:00 01/01/2019 12:00:00	mediocre

Tested Function	Test description / input	Expected result
sensorController. meanAirQuality(double latitude, double longitude, double radius, time_t start, time_t stop)	45 -2 5 01/01/2019 12:00:00 15/15/2025 12:00:00	null

e- Compare Sensors

This method uses several other functions to produce the wanted result:

1- FR7_averageValue(Sensor sensor, Attribute targetAttribute, time_t t1, time_t t2)

Compute the average value of all its measurements for the given attribute

Tested Function	Test description / input	Expected result
sensorController. FR7_averageValue(Sensor sensor, Attibute targetAttribute, time_t t1, time_t t2)	sensor0 03 01/01/2019 12:00:00 01/01/2019 12:20:00	50.25

Tested Function	Test description / input	Expected result
sensorController. FR7_averageValue(Sensor sensor, Attibute targetAttribute, time_t t1, time_t t2)	sensor0 o2 01/01/2019 12:00:00 01/01/2019 12:20:00	NULL

2- FR7_sensorComparison (Sensor sensorToCompare, timestamp t1, timestamp t2)

Tested Function	Test description / input	Expected result
sensorController. FR7_sensorComparison (Sensor sensorToCompare, timestamp t1, timestamp t2)	Sensor0 01/01/2019 12:00:00 01/01/2019 12:20:00	A map with Sensor object and a double value (similarity 0 to 1) : Sensor1 and 0.846

Tested Function	Test description / input	Expected result
sensorController. FR7_sensorComparison (Sensor sensorToCompare, timestamp t1, timestamp t2)	Sensor101 01/01/2019 12:00:00 01/01/2019 12:20:00	Res2.size()==0

f- Air Quality

This method uses several other functions to produce the wanted result:

1- isGivenTimeInsideTimePeriod(time_t t1, time_t t2): bool

Tested Function	Test description / input	Expected result
sensorController. isGivenTimeInsideTimePeriod(time_t start, time_t : stop,time_t: time): bool	01/01/2019 12:00:00 01/01/2019 12:20:00	true

Tested Function	Test description / input	Expected result
sensorController. isGivenTimeInsideTimePeriod(time_t start, time_t : stop, time_t: time): bool	01/01/2019 12:00:00 01/01/2025 12:00:00	false

2- distanceBetweenPositions (double latitudeA, double longitude, double latitudeB, double longitudeB)

Tested Function	Test description / input	Expected result
sensorController. distanceBetweenPositions (double latitudeA, double longitude, double latitudeB, double longitudeB)	44 -1 45 -2	1.41

3- FR8_qualityAttributes (double latitude, double longitude, time_t time)

Tested Function	Test description / input	Expected result
sensorController. FR8_qualityAttributes (double latitude, double longitude, time_t time)	45 -2 01/01/2019 12:00:00	Returns a Map<Attribute,double> with the Attribute object's identifiers o3, No2, So2 and PM10 and their corresponding measurement: O3 : 55.56 NO2 : 69.28 S02 : 38.56 PM10 : 47.39

Tested Function	Test description / input	Expected result
sensorController. FR8_qualityAttributes (double latitude, double longitude, time_t time)	45 -2 15/15/2025 12:00:00	null

4- FR8_quality (double latitude, double longitude, time_t time)

Tested Function	Test description / input	Expected result
sensorController. FR8_quality (double latitude, double longitude, time_t time)	45 -2 01/01/2019 12:00:00	String equal to “Mediocre”

Tested Function	Test description / input	Expected result
sensorController. FR8_quality (double latitude, double longitude, time_t time)	45 -2 15/15/2025 12:00:00	Not computable

IV.1.d. CleanerController functions

a- Compute Cleaner Statistics

Tested Function	Test description / input	Expected result
cleanerController.computeStatistics(Cleaner cleaner)	Cleaner with identifier Cleaner0	null

IV.2. Functional Tests

To test the different functionalities of our application we need to define what happens at each step of the scenario.

1 - Login fail

Step	Description	Input	Results
1	The user has to login	Id : User101 Password: banana	You don't have access to the application. Please enter a good id or password.

2 - Access sensors list

Step	Description	Actions	Results
1	The user has to login	Id: Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use_sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	get sensors	Name : Sensor0 Longitude: 44 Latitude: -1 Name : Sensor1 Longitude: 44 Latitude: -0.3 (The options are once again displayed)

3 - Retrieve sensor's measurements

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <0>	Name: Sensor0 Longitude: 44 Latitude: -1 Sensor Menu : 1- measurements 2- measurements<date> 3- evaluate 4- disable 5- enable 6- compare 7- menu Which functionality do you want to use?
3	He then has to choose an option in the new menu	measurements	Date : 01/01/2019 12:00 O3 : 50.25 NO2: 74.5 SO2: 41.5 PM10: 44.75 (The options are once again displayed)

4 - Retrieve sensor's measurements at a special date

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <0>	Name: Sensor0 Longitude: 44 Latitude: -1 Sensor Menu : 1- measurements 2- measurements<date> 3- evaluate 4- disable 5- enable 6- compare 7- menu Which functionality do you want to use?
3	He then has to choose an option in the new menu	Measurements<01/01/2019 12:00>	Date : 01/01/2019 12:00 O3 : 50.25 NO2: 74.5 SO2: 41.5 PM10: 44.75 (The options are once again displayed)

5 - Run malfunctioning sensor detection analysis

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <0>	Name: Sensor0 Longitude: 44 Latitude: -1 Sensor Menu : 1- measurements 2- measurements<date> 3- evaluate 4- disable 5- enable 6- compare 7- menu Which functionality do you want to use?
3	He then has to choose an option in the new menu	evaluate	0.08 *(The options are once again displayed)*

6 - Mark specific sensor as unreliable *(Only for the government agency)*

Step	Description	Actions	Results
1	A government agent has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <0>	Name: Sensor0 Longitude: 44 Latitude: -1 Sensor Menu : 1- measurements 2- measurements<date> 3- evaluate 4- disable 5- enable 6- compare 7- menu Which functionality do you want to use?
3	He then has to choose an option in the new menu	disable	Sensor0 has been disabled. *(The options are once again displayed)*

Similar test but for the functionality: Mark specific sensor as reliable. The command is now enabled, and the result should be displayed: Sensor0 has been enabled.

7 - Compare the similarity between the other sensors

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <0>	Name: Sensor0 Longitude: 44 Latitude: -1 Sensor Menu : 1- measurements 2- measurements<date> 3- evaluate 4- disable 5- enable 6- compare 7- menu Which functionality do you want to use?
3	He then has to choose an option in the new menu	compare	Sensor1 : 85% *(The options are once again displayed)*

8 - Consult a sensor: failed attempt

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	use sensor <102>	No sensor found 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?

9 - Retrieve mean quality of air on a specified area

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	airQ <45> <-2> <01/01/2019 12:00><01/01/2019 12:00>”	Mean air quality : mediocre (The options are once again displayed)

Similar test but with an error in the dates, the latitude, or the longitude.
Result: “Error in the input, please try again”

10 - Retrieve quality of air at a given position and time

Step	Description	Actions	Results
1	The user has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	Enter the command “airQ <45> <-2> <01/01/2019 12:00>”	Air Quality : mediocre (The options are once again displayed)

Similar test but with an error in the date, the latitude, or the longitude.
Result : “Error in the input, please try again”.

11 – Access the list of owned air cleaners (Only for providers)

Step	Description	Actions	Results
1	The Provider has to login	Id : Provider0 Password: provider0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	consult cleaner<0>	Name : Cleaner0 Latitude: 45.3333 Longitude: 1.33333 Installation date : 01/02/2019 12:00:00 (The options are once again displayed)

12 - Access individual users list *(Only for the government agency)*

Step	Description	Actions	Results
1	The Government agent has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	get users	Name: User0 Sensor: Sensor70 Name: User1 Sensor: Sensor36 (The options are once again displayed)

13 - Access all air cleaners providers list (Only for the government agency)

Step	Description	Actions	Results
1	The Government agent has to login	Id : Government0 Password: government0	You are connected. Menu 1- get_sensors 2- use sensor <num> 3- airQ <latitude> <longitude> <start_date><end_date> 4- airQ <latitude> <longitude> <date> 5- consult cleaner <num> 6- get users 7- get providers 8- get_cleaners Which functionality do you want to use?
2	He then has to choose a functionality	get providers	Name: Provider0 Sensor: Cleaner0 Name: Provider1 Sensor: Cleaner1 (The options are once again displayed)