INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Software Engineering & UML

AirWatcher Application

Team Project : B3120+ B3123

HOUDOUX Adrien
FAKRONI Mohammed
CHAOUKI Youssef
ZHANG Lizhi

Introduction

AirWatcher is an application commissioned by a government environmental protection agency to facilitate the monitoring and analysis of air quality across a large territory.

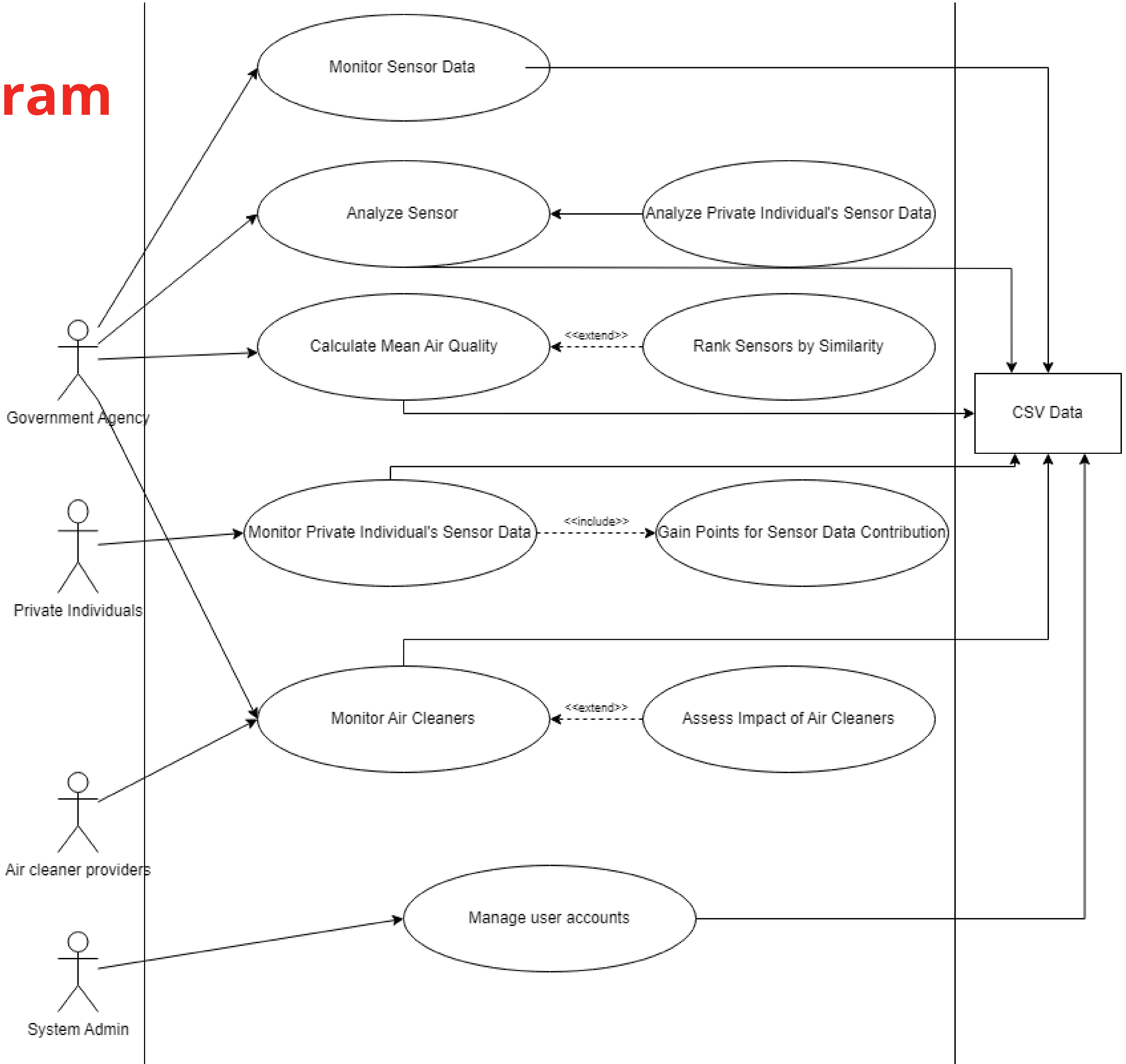
AirWatcher is designed to process and analyze environmental data collected through an array of sensors dispersed throughout the region. It will also enable the agency to maintain sensor integrity and manage air cleaners

INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
LYON

Requirements specification

AirWatcher
Application

1) Use Case Diagram

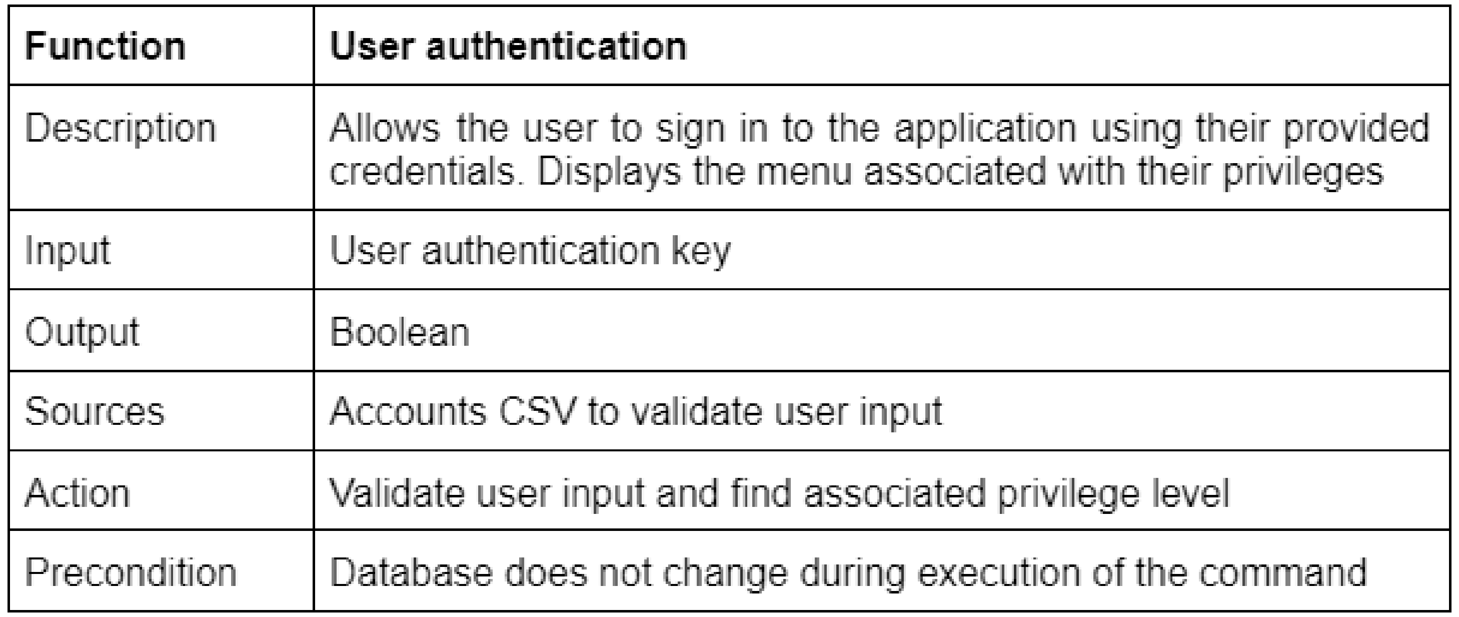

Monitor Sensor Data
Analyze Sensor
Analyze Private Individual's Sensor Data
Calculate Mean Air Quality
<<extend>>
Rank Sensors by Similarity
CSV Data
Government Agency
Monitor Private Individual's Sensor Data
<<include>>
Gain Points for Sensor Data Contribution
Private Individuals
Monitor Air Cleaners
<<extend>>
Assess Impact of Air Cleaners
Air cleaner providers
Manage user accounts
System Admin

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

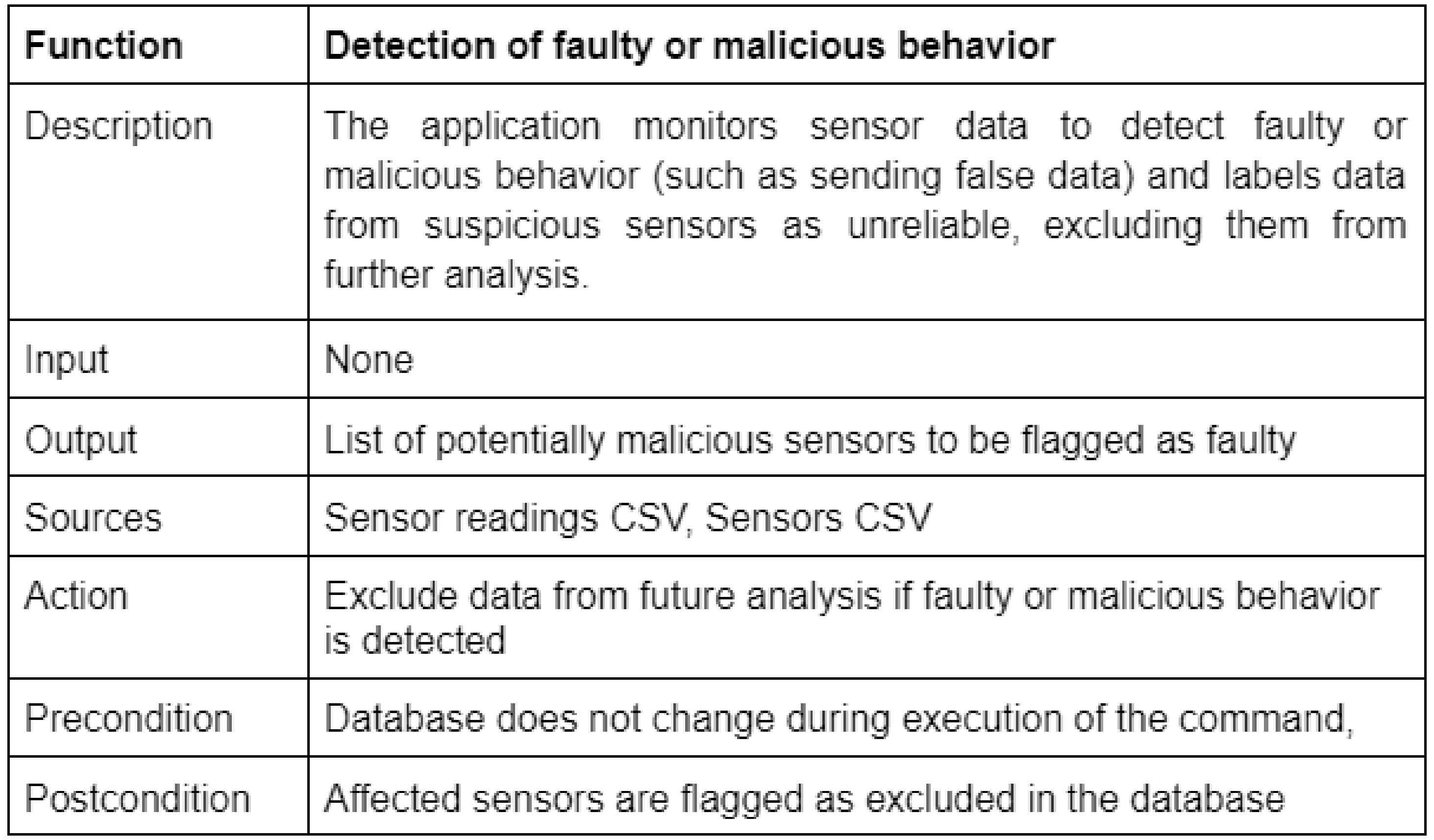
Requirements specification

AirWatcher Application

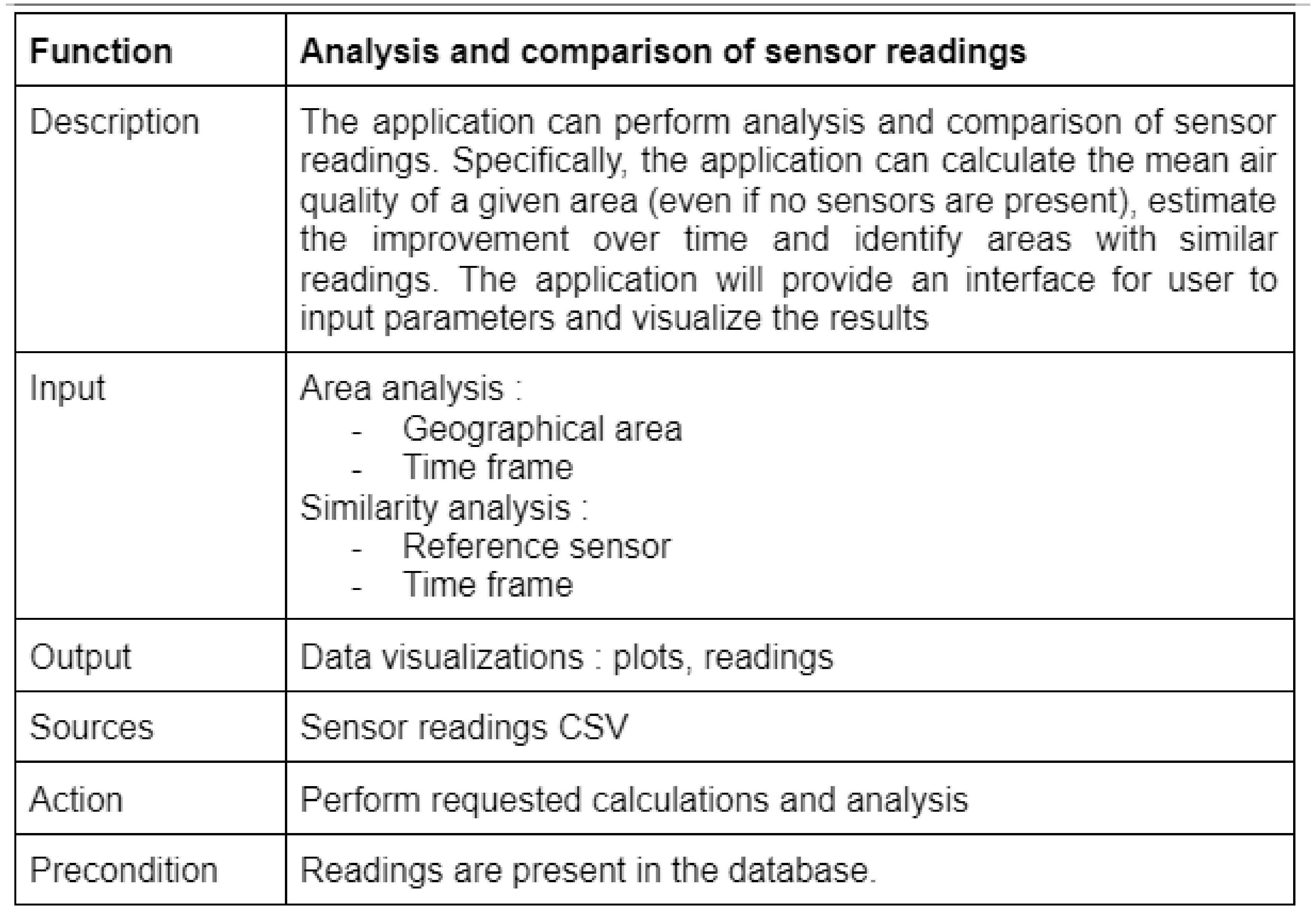
2) Functional requirements of the system



Function	User authentication
Description	Allows the user to sign in to the application using their provided credentials. Displays the menu associated with their privileges
Input	User authentication key
Output	Boolean
Sources	Accounts CSV to validate user input
Action	Validate user input and find associated privilege level
Precondition	Database does not change during execution of the command



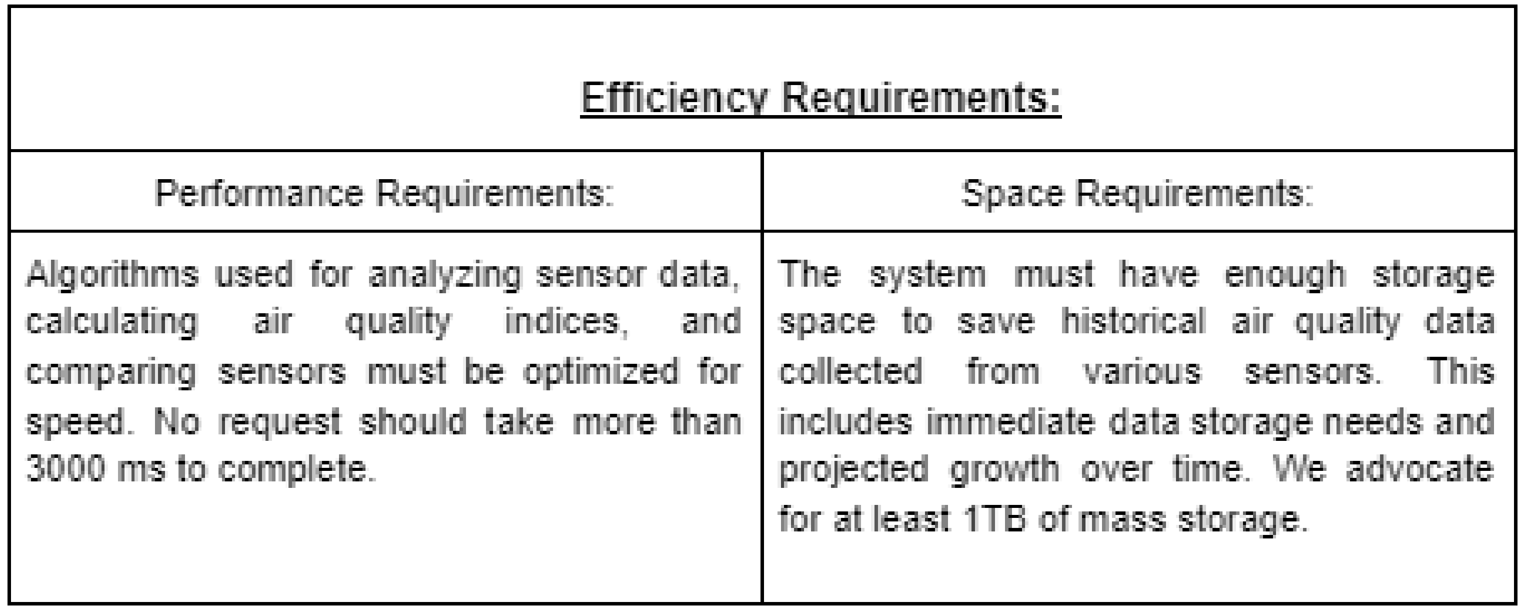
Function	Detection of faulty or malicious behavior
Description	The application monitors sensor data to detect faulty or malicious behavior (such as sending false data) and labels data from suspicious sensors as unreliable, excluding them from further analysis.
Input	None
Output	List of potentially malicious sensors to be flagged as faulty
Sources	Sensor readings CSV, Sensors CSV
Action	Exclude data from future analysis if faulty or malicious behavior is detected
Precondition	Database does not change during execution of the command,
Postcondition	Affected sensors are flagged as excluded in the database



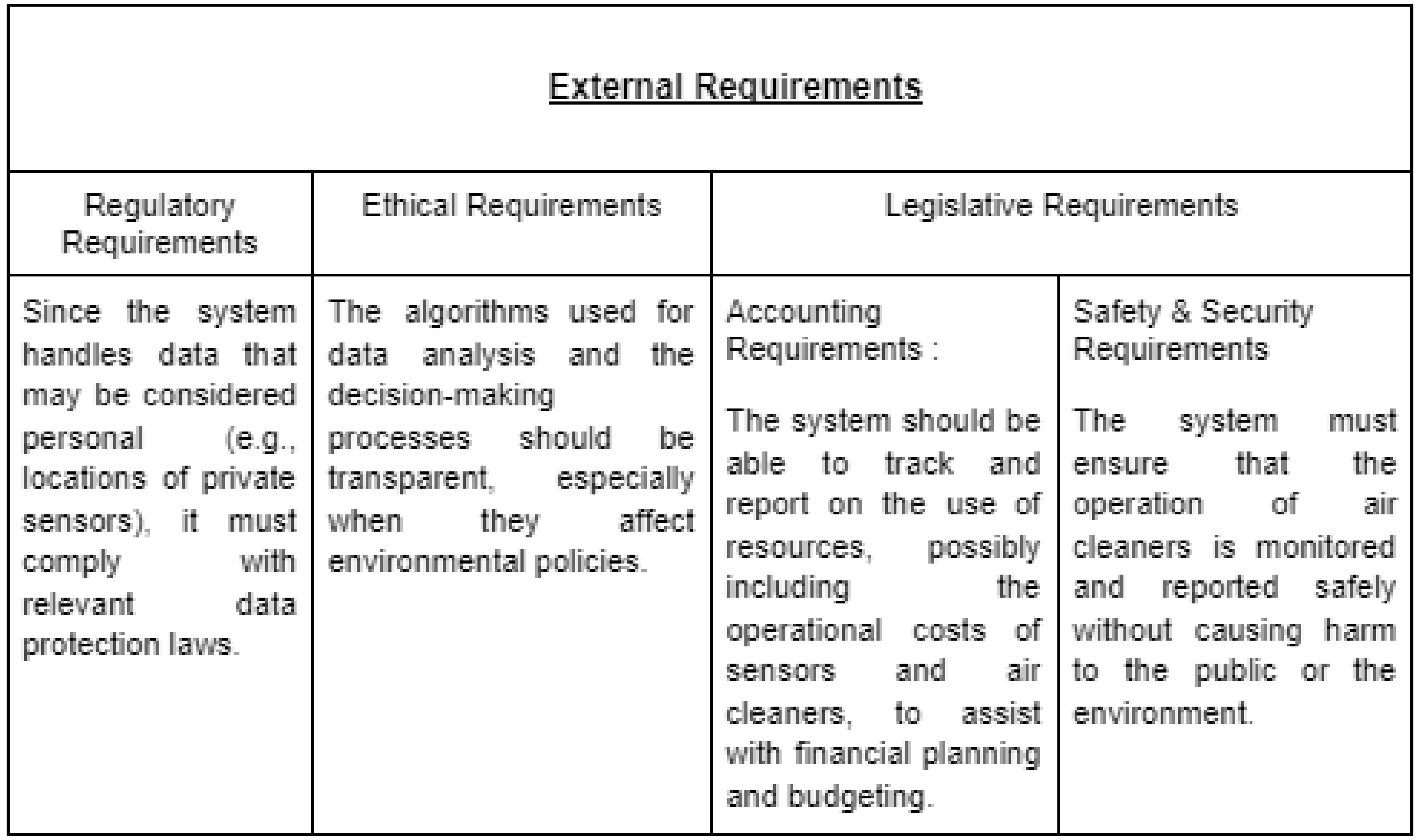
Function	Analysis and comparison of sensor readings
Description	The application can perform analysis and comparison of sensor readings. Specifically, the application can calculate the mean air quality of a given area (even if no sensors are present), estimate the improvement over time and identify areas with similar readings. The application will provide an interface for user to input parameters and visualize the results
Input	Area analysis : - Geographical area - Time frame Similarity analysis : - Reference sensor - Time frame
Output	Data visualizations : plots, readings
Sources	Sensor readings CSV
Action	Perform requested calculations and analysis
Precondition	Readings are present in the database.

Requirements specification

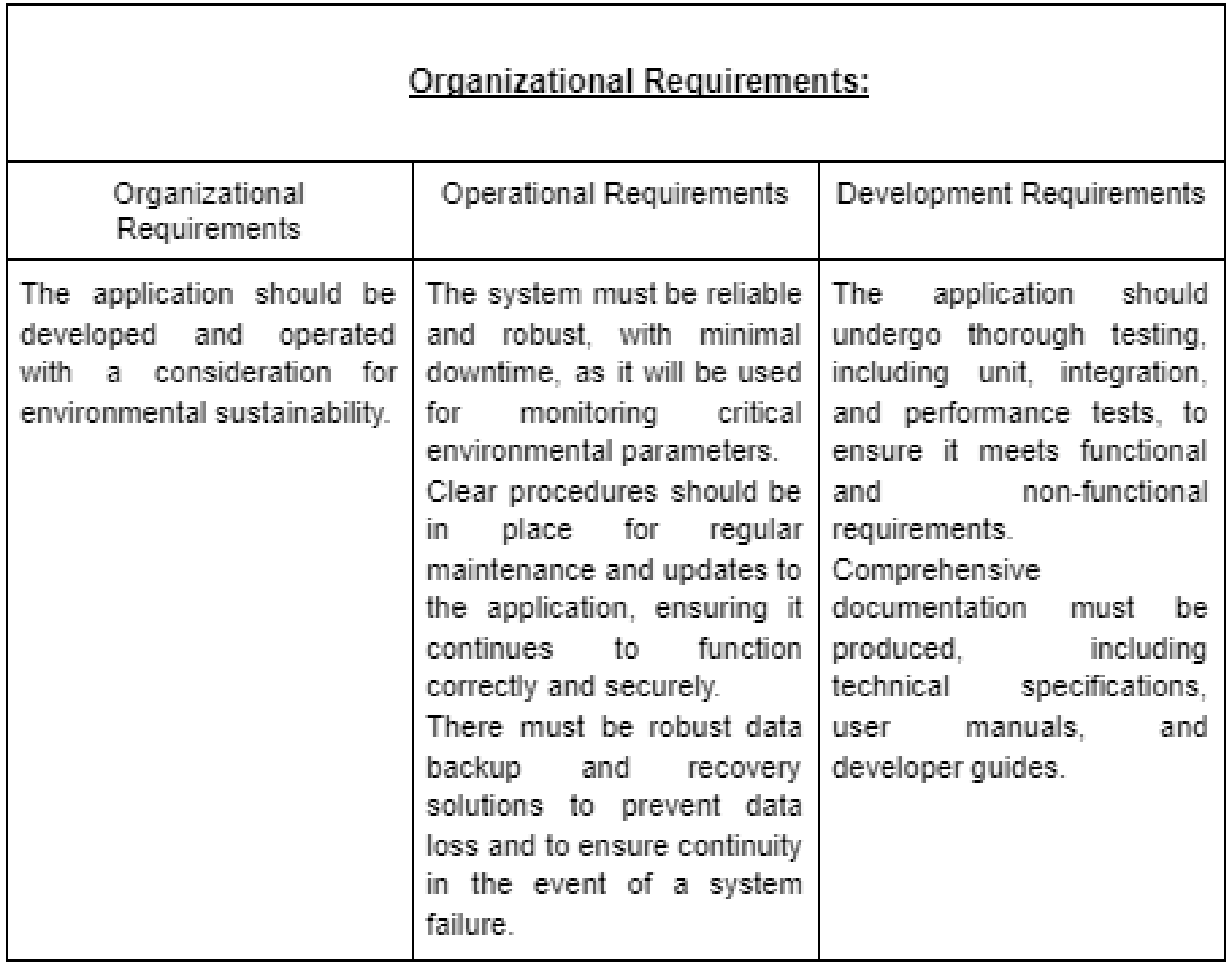
2) Non Functional requirements of the system



Efficiency Requirements:	
Performance Requirements:	Space Requirements:
Algorithms used for analyzing sensor data, calculating air quality indices, and comparing sensors must be optimized for speed. No request should take more than 3000 ms to complete.	The system must have enough storage space to save historical air quality data collected from various sensors. This includes immediate data storage needs and projected growth over time. We advocate for at least 1TB of mass storage.



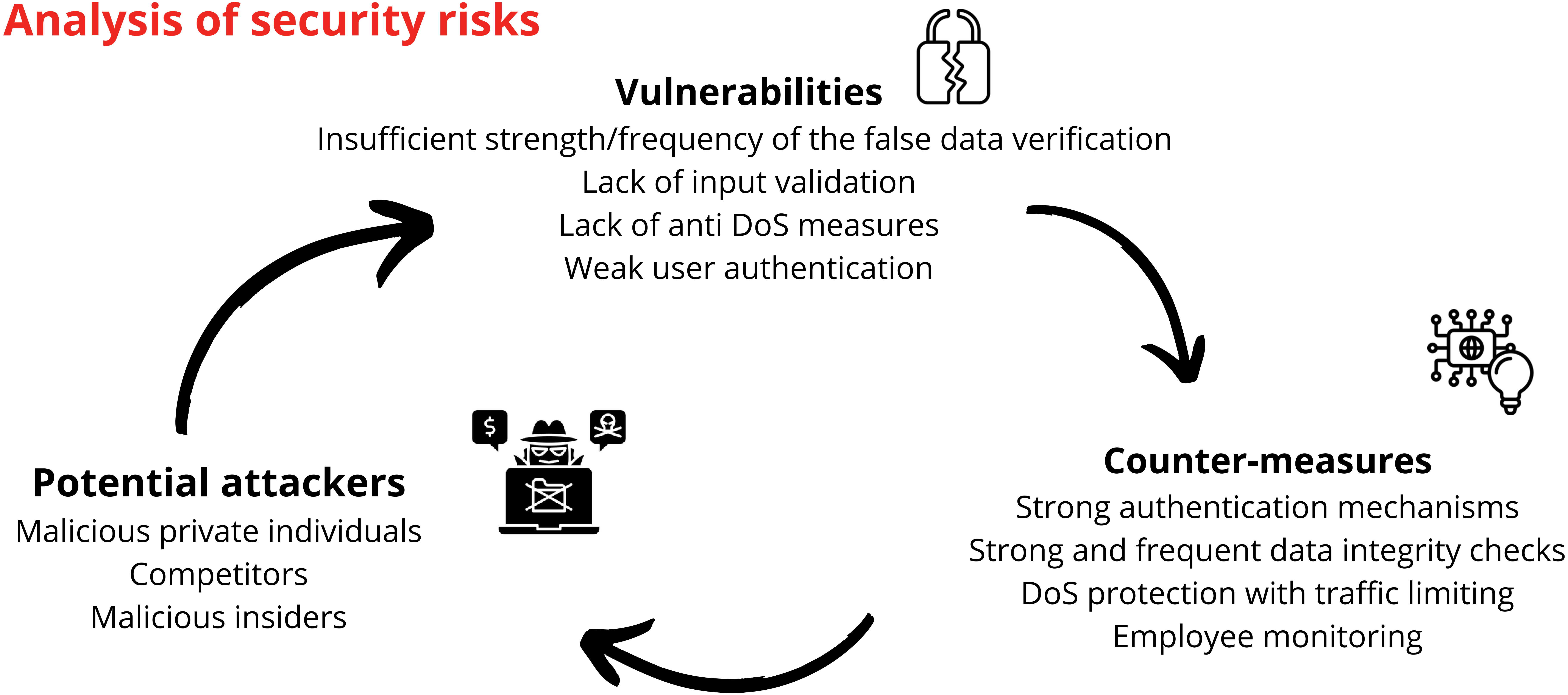
External Requirements			
Regulatory Requirements	Ethical Requirements	Legislative Requirements	
Since the system handles data that may be considered personal (e.g., locations of private sensors), it must comply with relevant data protection laws.	The algorithms used for data analysis and the decision-making processes should be transparent, especially when they affect environmental policies.	Accounting Requirements : The system should be able to track and report on the use of resources, possibly including the operational costs of sensors and air cleaners, to assist with financial planning and budgeting.	Safety & Security Requirements The system must ensure that the operation of air cleaners is monitored and reported safely without causing harm to the public or the environment.



Organizational Requirements:		
Organizational Requirements	Operational Requirements	Development Requirements
The application should be developed and operated with a consideration for environmental sustainability.	The system must be reliable and robust, with minimal downtime, as it will be used for monitoring critical environmental parameters. Clear procedures should be in place for regular maintenance and updates to the application, ensuring it continues to function correctly and securely. There must be robust data backup and recovery solutions to prevent data loss and to ensure continuity in the event of a system failure.	The application should undergo thorough testing, including unit, integration, and performance tests, to ensure it meets functional and non-functional requirements. Comprehensive documentation must be produced, including technical specifications, user manuals, and developer guides.

Requirements specification

3) Analysis of security risks



Vulnerabilities

Insufficient strength/frequency of the false data verification
Lack of input validation
Lack of anti DoS measures
Weak user authentication

Counter-measures

Strong authentication mechanisms
Strong and frequent data integrity checks
DoS protection with traffic limiting
Employee monitoring

Potential attackers

Malicious private individuals
Competitors
Malicious insiders

Requirements specification

4) Validation tests

Analysis and Comparison of Sensors

Verify Detection of Faulty Sensors
Verify Anomaly Detection in Sensor Measurements

Calculation or Estimation of Air Quality

Verify Calculation of Air Quality at the specific location
Verify Consistency of Air Quality Estimations

Data Aggregation

Verify Calculation of Air Quality Statistics
Verify Calculation of the mean Air Quality

Detection of Malicious Behavior

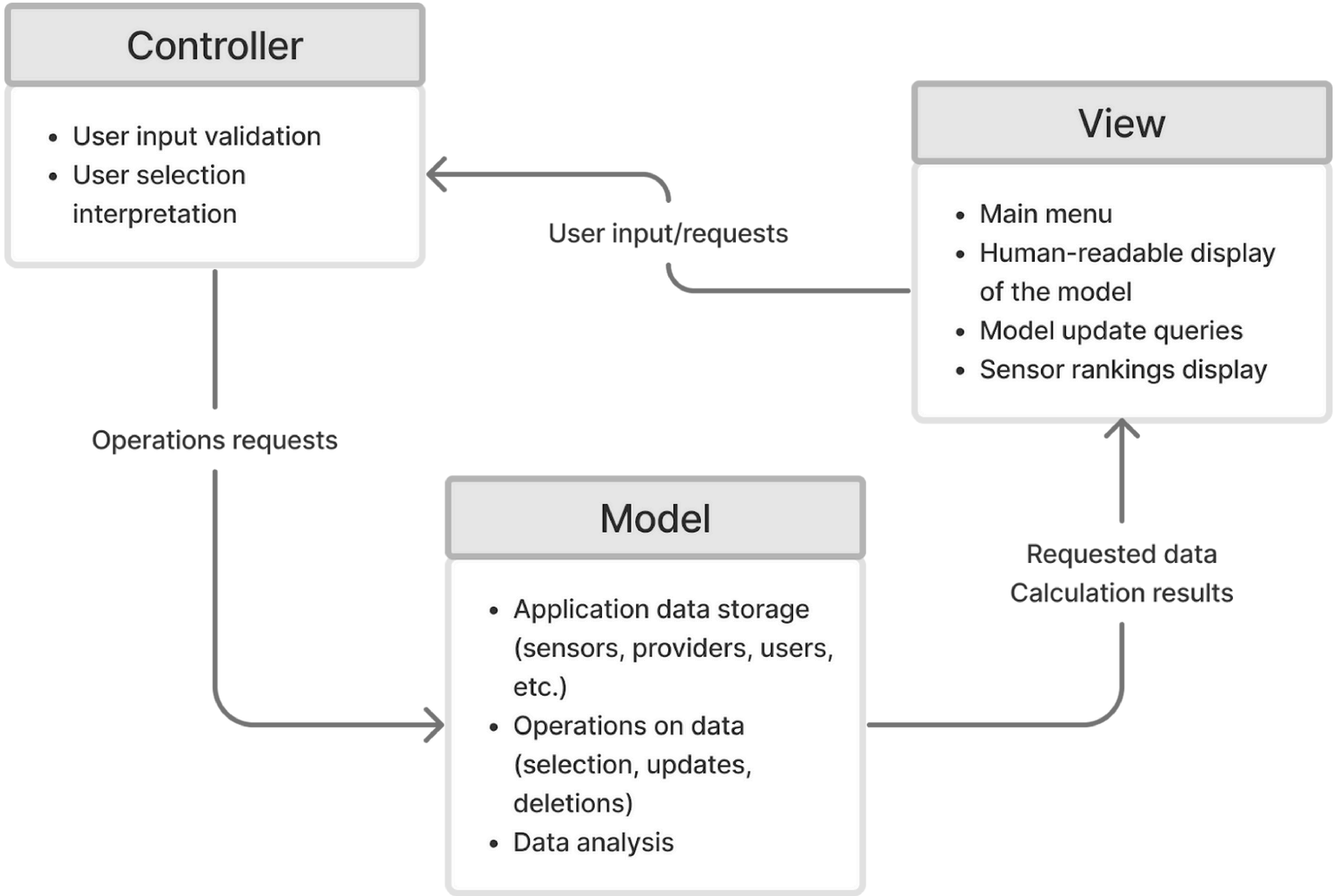
Verify Monitoring and Detection of Malicious Sensor
Verify Exclusion of Suspect Sensor Data from Analysis

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Design Document

AirWatcher Application

1) Architecture



Controller

- User input validation
- User selection interpretation

View

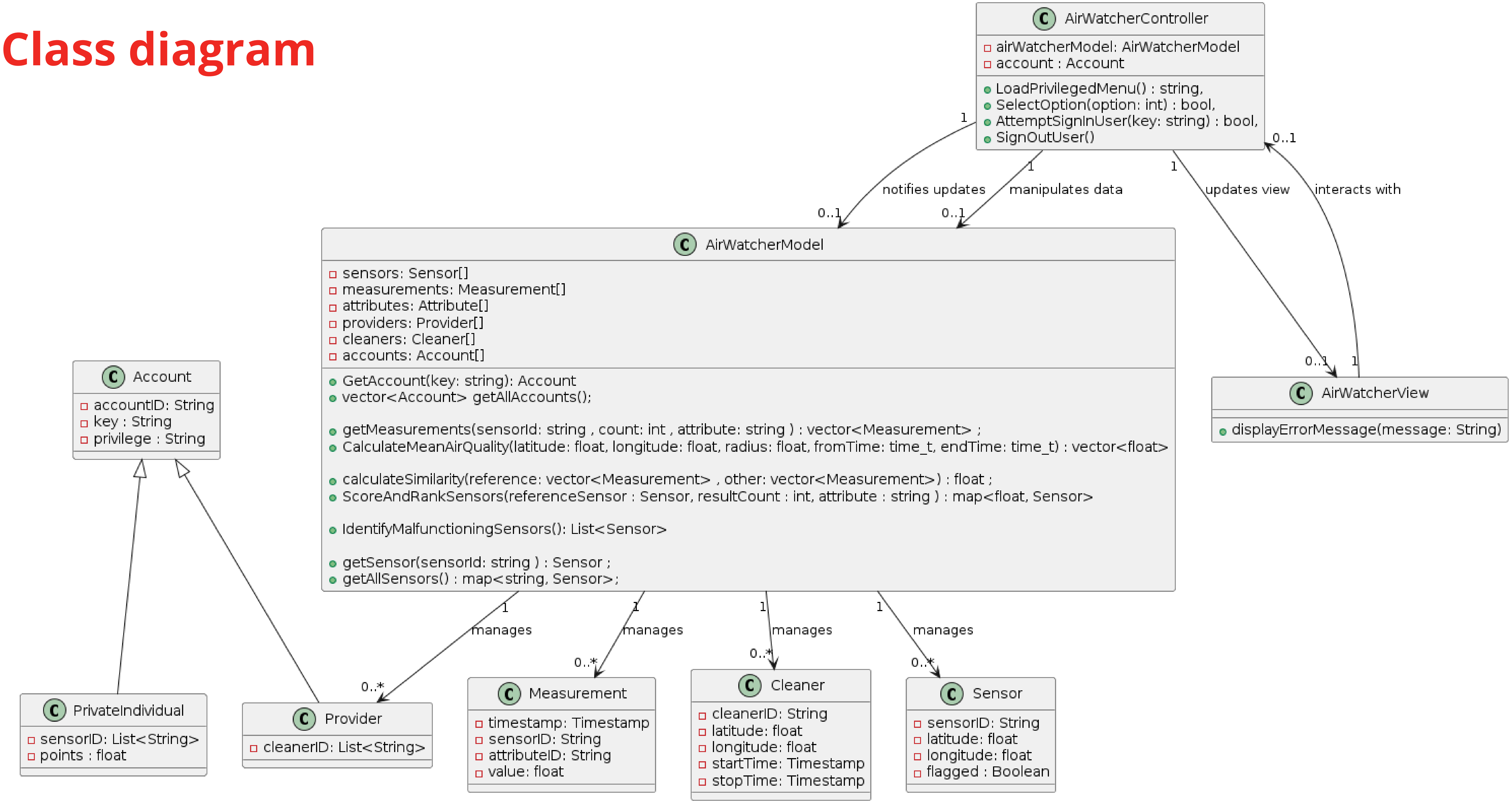
- Main menu
- Human-readable display of the model
- Model update queries
- Sensor rankings display

Model

- Application data storage (sensors, providers, users, etc.)
- Operations on data (selection, updates, deletions)
- Data analysis

Design Document

2) Class diagram



AirWatcherController
- airWatcherModel: AirWatcherModel
- account : Account
+ LoadPrivilegedMenu() : string,
+ SelectOption(option: int) : bool,
+ AttemptSignInUser(key: string) : bool,
+ SignOutUser()

AirWatcherModel
- sensors: Sensor[]
- measurements: Measurement[]
- attributes: Attribute[]
- providers: Provider[]
- cleaners: Cleaner[]
- accounts: Account[]
+ GetAccount(key: string): Account
+ vector<Account> getAllAccounts();
+ getMeasurements(sensorId: string , count: int , attribute: string) : vector<Measurement> ;
+ CalculateMeanAirQuality(latitude: float, longitude: float, radius: float, fromTime: time_t, endTime: time_t) : vector<float>
+ calculateSimilarity(reference: vector<Measurement> , other: vector<Measurement>) : float ;
+ ScoreAndRankSensors(referenceSensor : Sensor, resultCount : int, attribute : string) : map<float, Sensor>
+ IdentifyMalfunctioningSensors(): List<Sensor>
+ getSensor(sensorId: string) : Sensor ;
+ getAllSensors() : map<string, Sensor>;

AirWatcherView
+ displayErrorMessage(message: String)

Account
- accountID: String
- key : String
- privilege : String

PrivateIndividual
- sensorID: List<String>
- points : float

Provider
- cleanerID: List<String>

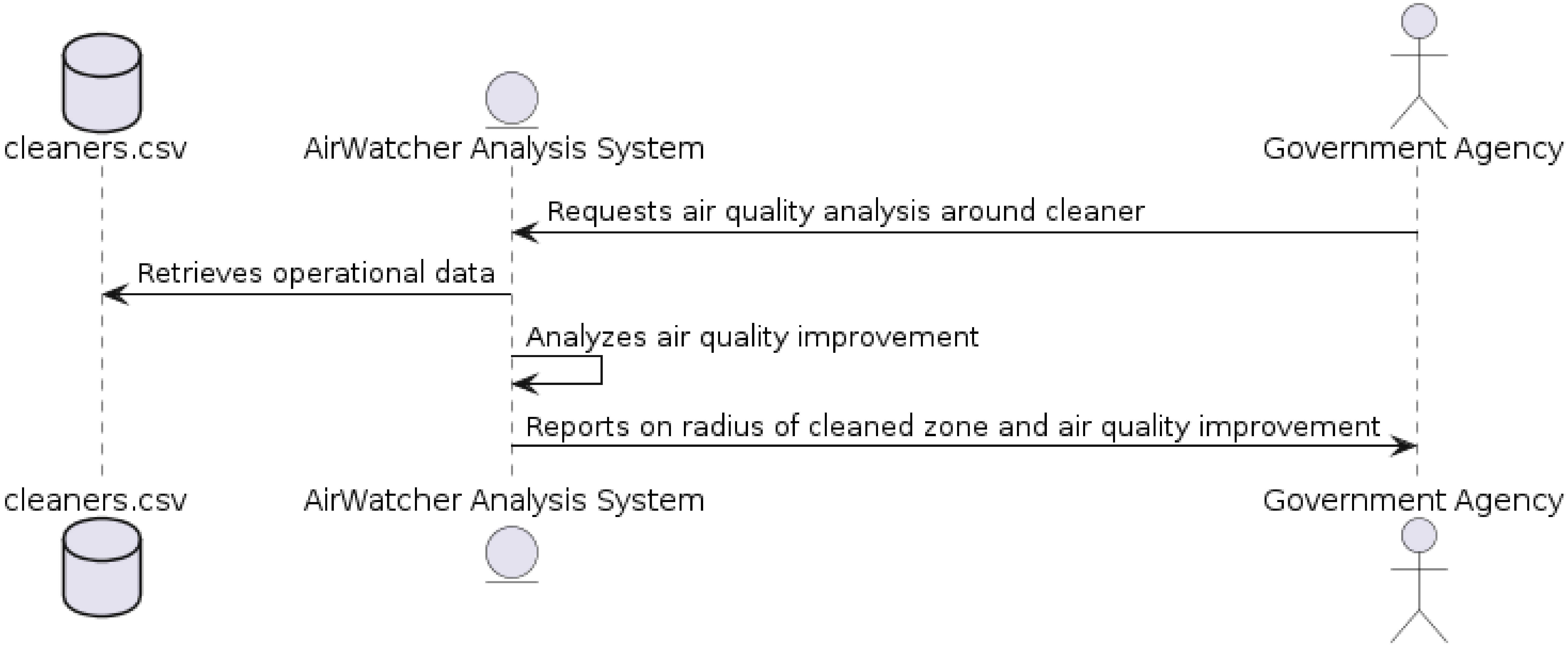
Measurement
- timestamp: Timestamp
- sensorID: String
- attributeID: String
- value: float

Cleaner
- cleanerID: String
- latitude: float
- longitude: float
- startTime: Timestamp
- stopTime: Timestamp

Sensor
- sensorID: String
- latitude: float
- longitude: float
- flagged : Boolean

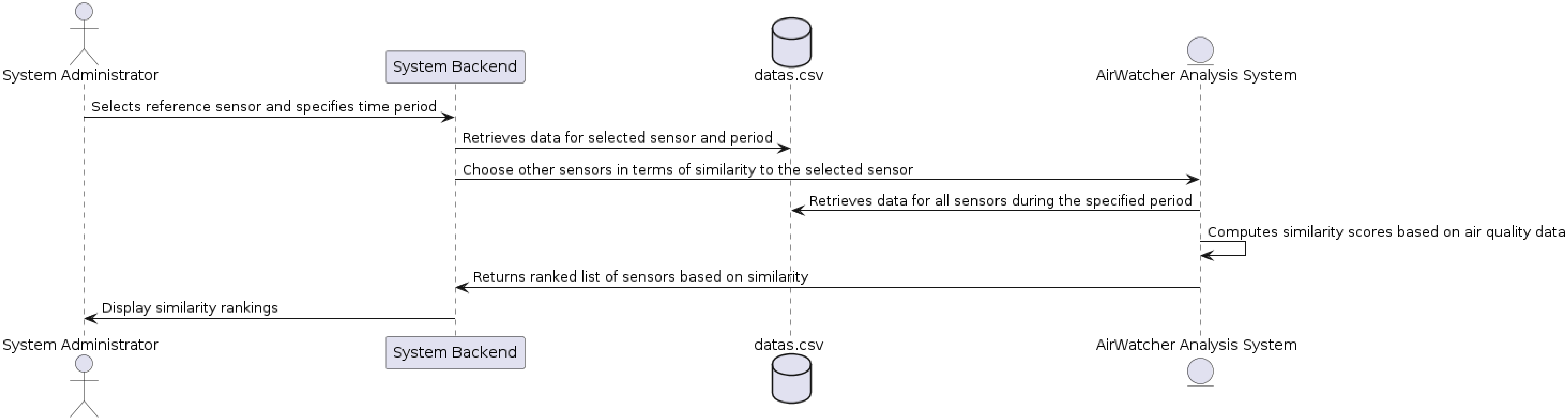
3) Major usage scenarios

Scenario 1 : Analyzing sensor data


cleaners.csv
AirWatcher Analysis System
Government Agency
Requests air quality analysis around cleaner
Retrieves operational data
Analyzes air quality improvement
Reports on radius of cleaned zone and air quality improvement
cleaners.csv
AirWatcher Analysis System
Government Agency

3) Major usage scenarios

Scenario 2 : Find similarity between sensors



System Administrator

System Backend

datas.csv

AirWatcher Analysis System

Selects reference sensor and specifies time period

Retrieves data for selected sensor and period

Choose other sensors in terms of similarity to the selected sensor

Retrieves data for all sensors during the specified period

Computes similarity scores based on air quality data

Returns ranked list of sensors based on similarity

Display similarity rankings

System Administrator

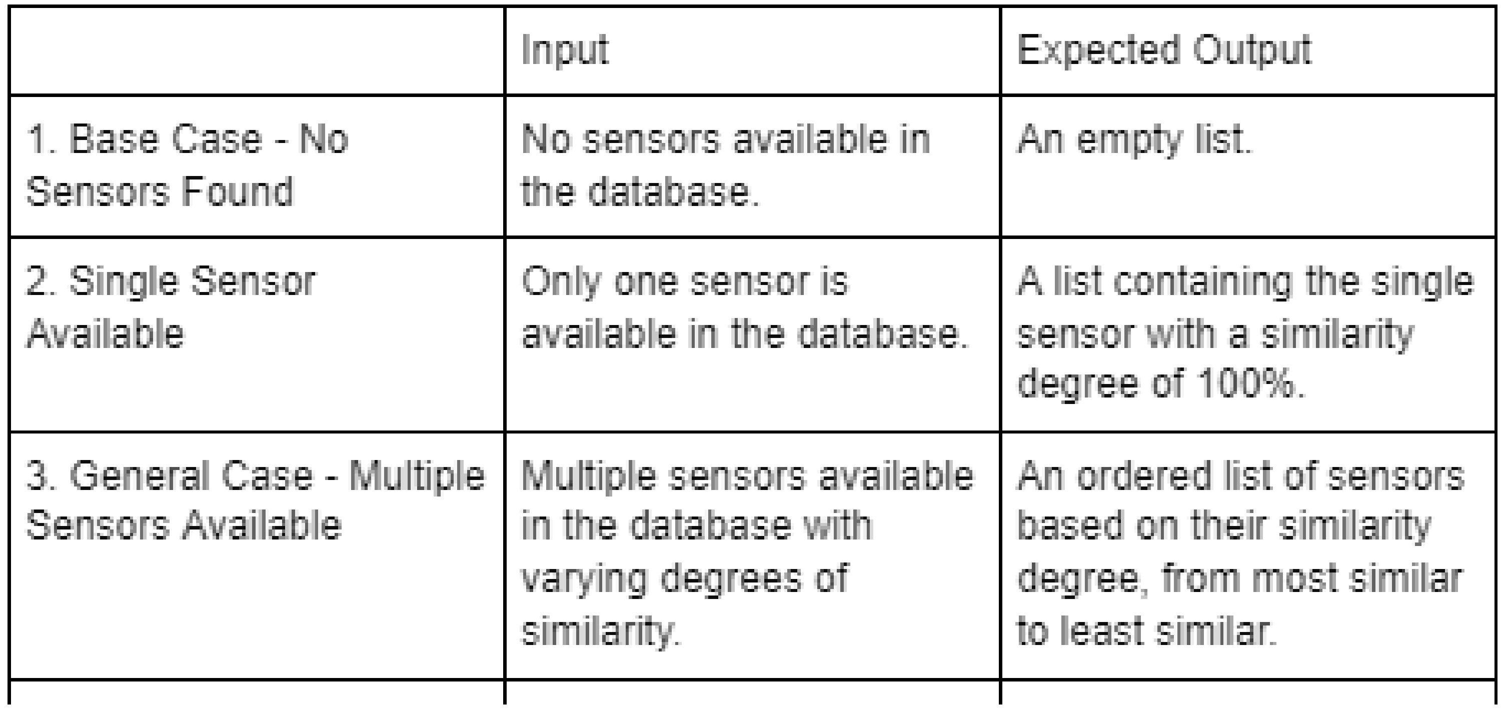
System Backend

datas.csv

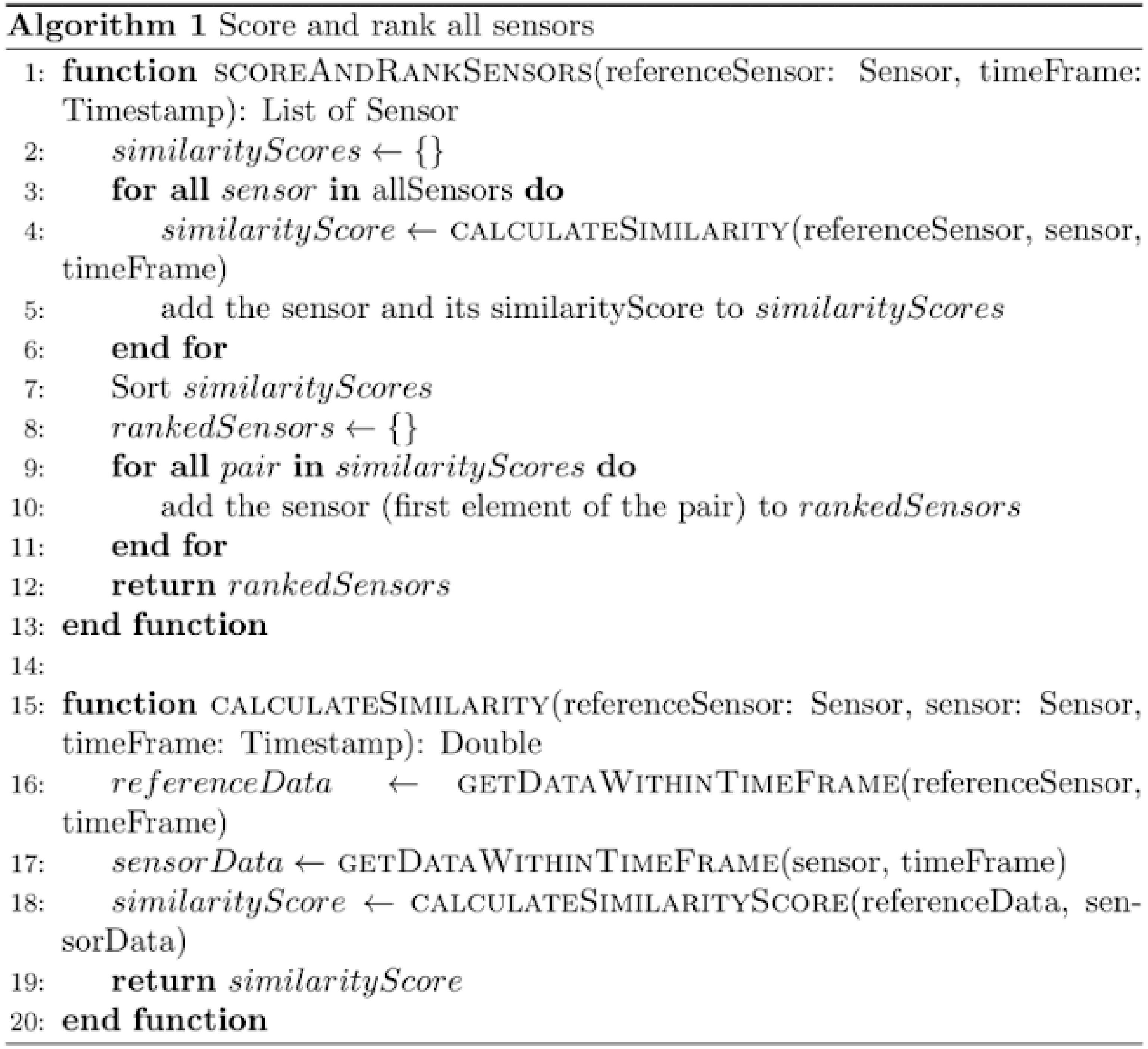
AirWatcher Analysis System

3) Major algorithms & Unitests

1) Rank sensors by similarity



	Input	Expected Output
1. Base Case - No Sensors Found	No sensors available in the database.	An empty list.
2. Single Sensor Available	Only one sensor is available in the database.	A list containing the single sensor with a similarity degree of 100%.
3. General Case - Multiple Sensors Available	Multiple sensors available in the database with varying degrees of similarity.	An ordered list of sensors based on their similarity degree, from most similar to least similar.

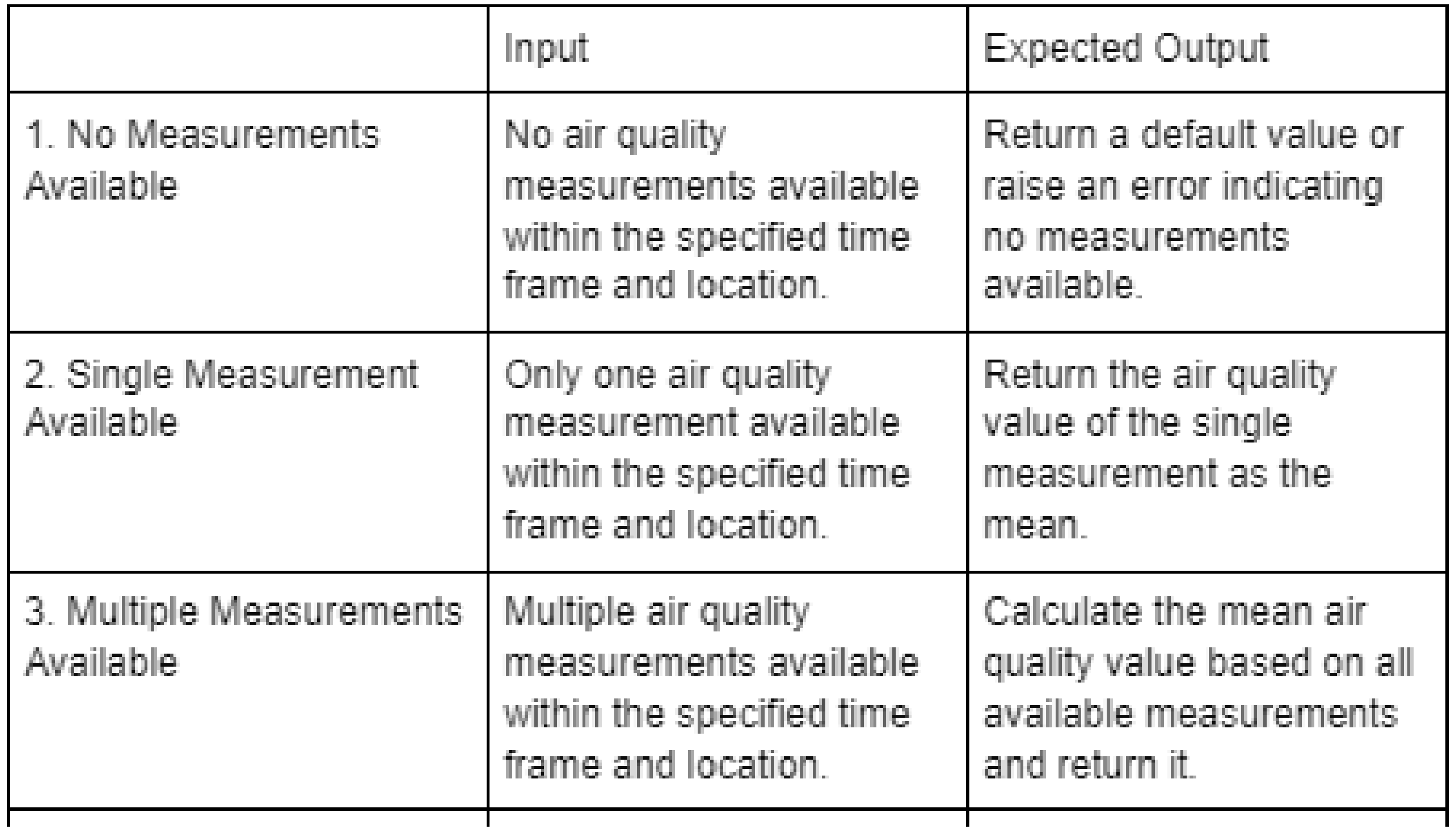


Algorithm 1 Score and rank all sensors

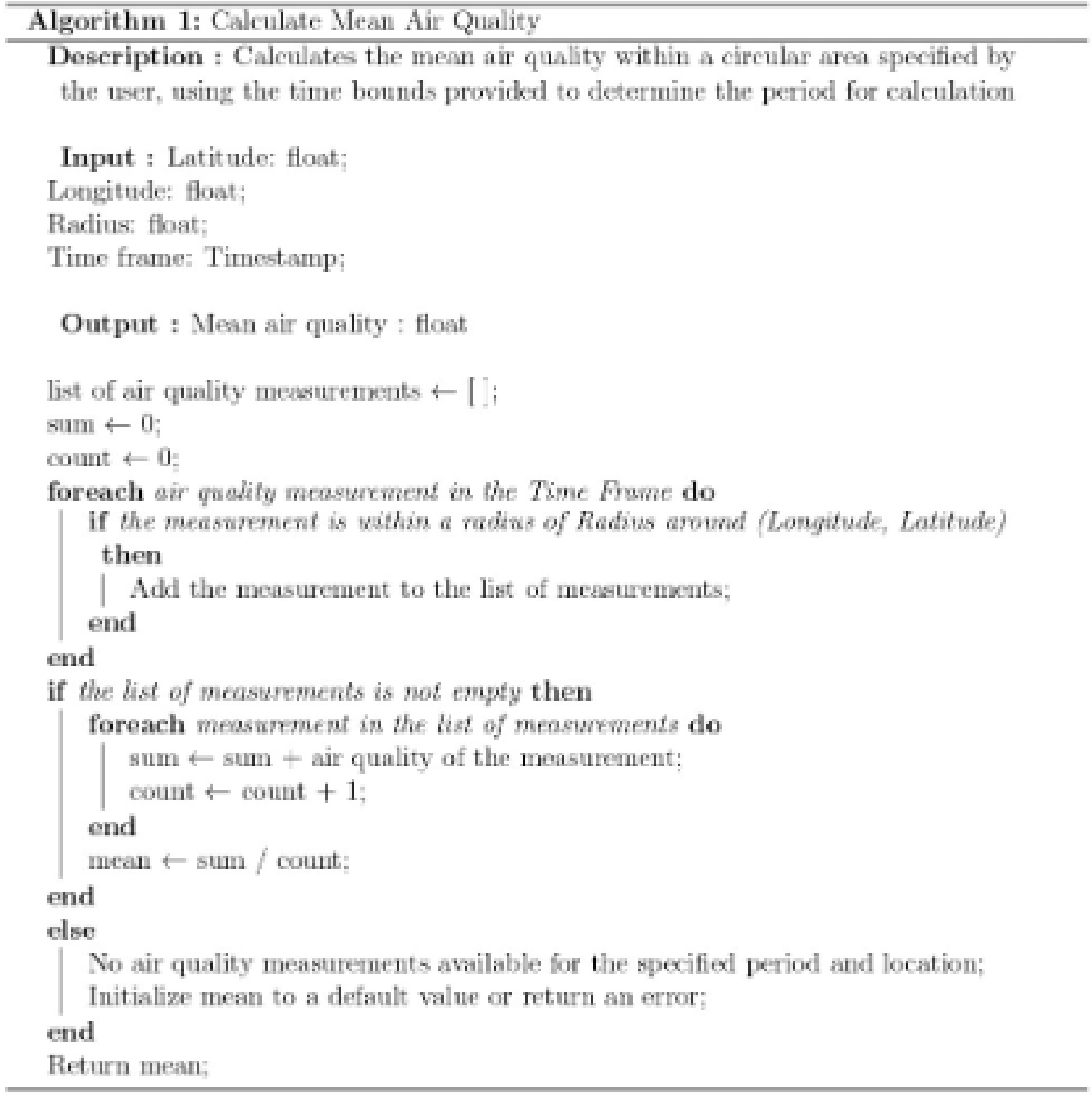
```
function scoreAndRankSensors(referenceSensor: Sensor, timeFrame: Timestamp): List of Sensor
    $similarityScores \leftarrow \{\}$
    for all $sensor$ in allSensors do
        $similarityScore \leftarrow$ calculateSimilarity(referenceSensor, sensor, timeFrame)
        add the sensor and its similarityScore to $similarityScores$
    end for
    Sort $similarityScores$
    $rankedSensors \leftarrow \{\}$
    for all $pair$ in $similarityScores$ do
        add the sensor (first element of the pair) to $rankedSensors$
    end for
    return $rankedSensors$
end function

function calculateSimilarity(referenceSensor: Sensor, sensor: Sensor, timeFrame: Timestamp): Double
    $referenceData \leftarrow$ getDataWithinTimeFrame(referenceSensor, timeFrame)
    $sensorData \leftarrow$ getDataWithinTimeFrame(sensor, timeFrame)
    $similarityScore \leftarrow$ calculateSimilarityScore(referenceData, sensorData)
    return $similarityScore$
end function
```

3) Major algorithms & Unitests

2) Calculate mean air quality



	Input	Expected Output
1. No Measurements Available	No air quality measurements available within the specified time frame and location.	Return a default value or raise an error indicating no measurements available.
2. Single Measurement Available	Only one air quality measurement available within the specified time frame and location.	Return the air quality value of the single measurement as the mean.
3. Multiple Measurements Available	Multiple air quality measurements available within the specified time frame and location.	Calculate the mean air quality value based on all available measurements and return it.



```
Algorithm 1: Calculate Mean Air Quality
Description : Calculates the mean air quality within a circular area specified by
 the user, using the time bounds provided to determine the period for calculation

 Input : Latitude: float;
Longitude: float;
Radius: float;
Time frame: Timestamp;

 Output : Mean air quality : float

list of air quality measurements ← [ ];
sum ← 0;
count ← 0;
foreach air quality measurement in the Time Frame do
    if the measurement is within a radius of Radius around (Longitude, Latitude)
     then
        Add the measurement to the list of measurements;
    end
end
if the list of measurements is not empty then
    foreach measurement in the list of measurements do
        sum ← sum + air quality of the measurement;
        count ← count + 1;
    end
    mean ← sum / count;
end
else
    No air quality measurements available for the specified period and location;
    Initialize mean to a default value or return an error;
end
Return mean;
```

Demonstration of the implemented functionalities