Logo: https://freesvg.org/abstract-wind-symbol (Creative Commons)

GL-UML : AirWatcher

Sense it, process it, breathe it

Maxence Drutel, Marie Guillevic, Romain Gallé, Florian Rascoussier

Maxence Drutel
Team manager and Algorithm Engineer

Maxence is the manager of the team meaning he is the least qualified person here. He's in charge of time management and task distribution.
He is also in charge of designing and coding the main algorithms of the application

Marie Guillevic
Application Quality Engineer

A good program is a working program. Marie is charged of designing and creating tests scenarios for the application, making sure of its robustness and quality.

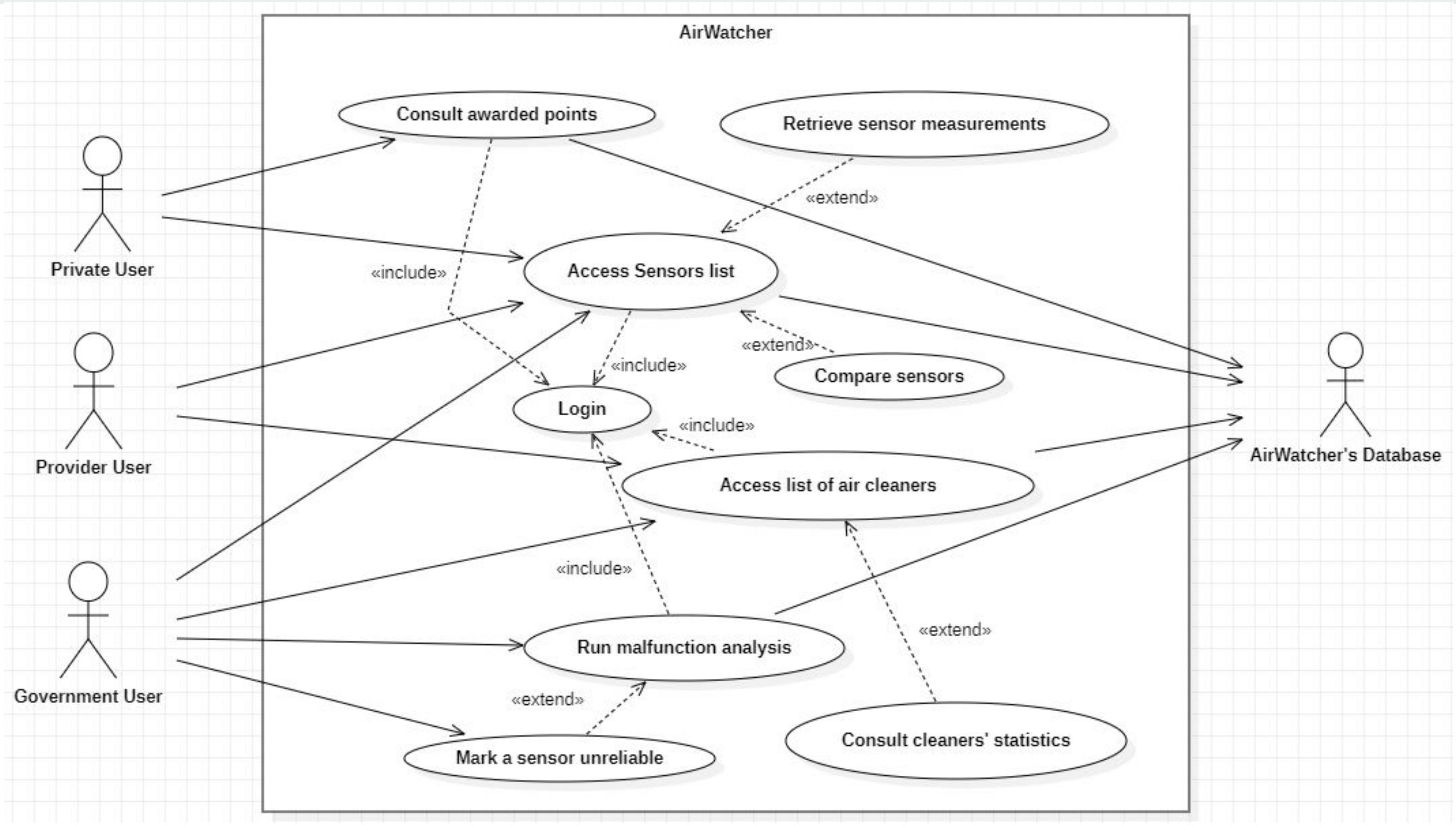
Florian Rascoussier
Secure Application Architect

Florian listened carefully to a Security By Design lecture, and thus is perfectly qualified to be in charge of cybersecurity. He is tasked with the application's architecture and with the User/Data interaction, ensuring the app is easy to use, secure and its datas coherents.

Romain Gallé
Core Application Engineer

Romain is the cornerstone of our project. He's the main team member in charge of the application's core functionalities. He is first tasked with defining and designing the app's functions, and will then be supervising their developpement.

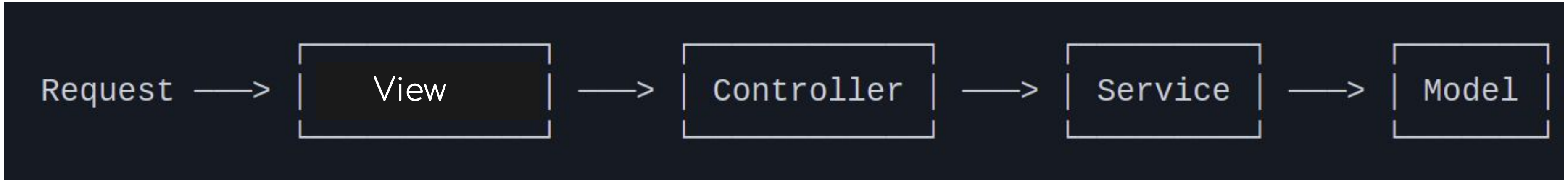
Use cases


AirWatcher
Consult awarded points
Retrieve sensor measurements
«extend»
Private User
«include»
Access Sensors list
«extend»
«include»
Compare sensors
Login
Provider User
«include»
AirWatcher's Database
Access list of air cleaners
«include»
Government User
Run malfunction analysis
«extend»
«extend»
Mark a sensor unreliable
Consult cleaners' statistics

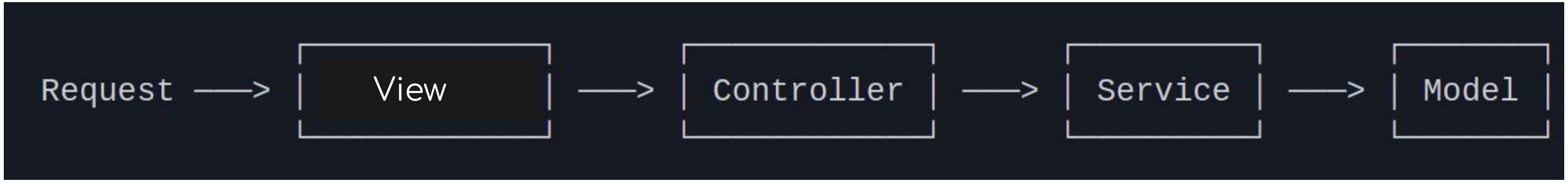
Architecture

Architecture (Design Pattern): MVCS

Model - View - Controller - Service



Adapted from
https://quantiphi.com/an-introduction-to-mvcs-architecture/

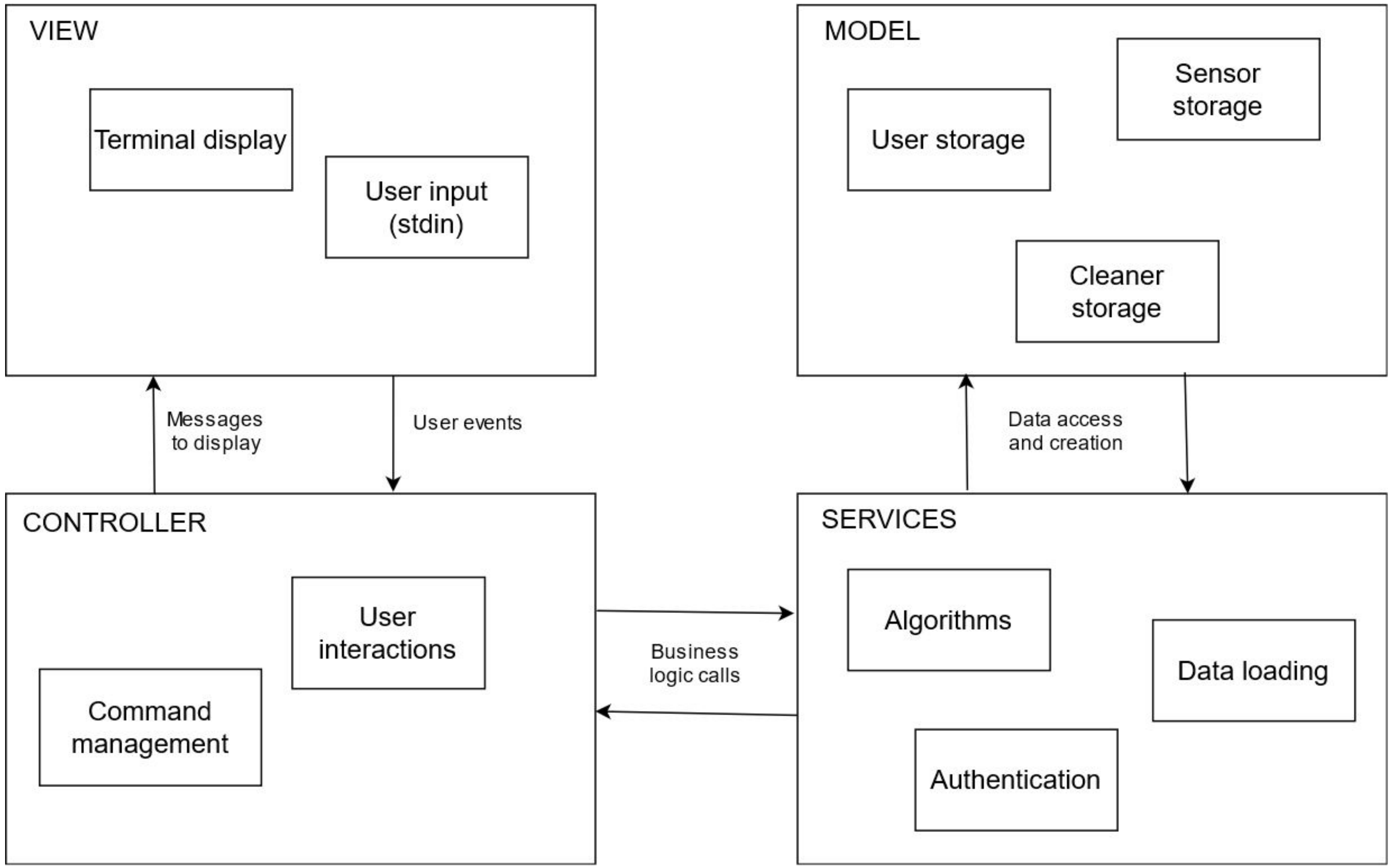


Adapted from
https://quantiphi.com/an-introduction-to-mvcs-architecture/

What is MVCS

- **Model Layer**: Responsible for defining how is defined the data model (Sensor, User, Measurement...)
- **Service Layer**: Also known as the **Business Layer**, contains the functionality that compounds the core of the application, thus becoming highly reusable for controllers and services (our Services)
- **Controller layer**: Responsible only for receiving the request and displaying the result of the operation (here : the Terminal class)
- **View Layer**: Represents how data will be rendered to the user

MVCS architecture of AirWatcher



Benefits of MVCS

“The service layer can be interpreted in many ways, but this is usually where you have your main business processing logic and are below your MVC architecture, but above your data access architecture.” [1]

main benefits :

- Helps to separate business logic from models
- Retrieve and create a "model" from various data sources (or data access objects).
- Update values in different repositories / resources.
- Execute application-specific logic and manipulations, etc.
- Simplifies the distribution of tasks between developers

[1] https://quantiphi.com/an-introduction-to-mvcs-architecture (accessed June 3rd)

UML Diagrams

See in StarUml

Algorithms

3 algorithms:

- FR8 – Determine the quality of air at a given time and location
- FR5 – Help determine if a sensor is reliable
- FR7 – Find the level of similarity between one specified sensor and all the others for a specific period

We have first made the algorithms with pseudocode before implementing them in the C++ application.

FR8 - Determine the quality of air at a given time and location

Goal: determine **ATMO quality of air** at a given time and location, based on the **neighbouring sensors records**.

-> We have to estimate the 4 air attributes for the location

-> Ponderate the sensor with its distance : **the further away, the lesser it counts**

-> Calculate the **ponderated average** to get the position’s air quality

$$weight_{sensor} = \frac{distance_{referencial}}{distance_{sensor}}$$

-> **Convert the air attributes to ATMO Score**

FR5 – Help determine if a sensor is reliable

Goal: help determine if a **private sensor is reliable or not** :

-> **Remove the target Sensor** from our measurement list

-> **Call the FR8_AirQuality for the target Sensor's position** to estimate quality with all other sensors

-> **Relative gap** between the target sensor's value and the algorithm's predicted value

-> **The lower the relative gap, the higher the sensor's confidence**

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

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

Goal: estimate a **level of similarity between one specified sensor and all the others** for a specific period.

->Returns a **relative gap** for the values associated with the sensor compared with the others

-> reuse relative gap used for the FR5 algorithm

-> calculate **average value** for all sensors

-> Compute **average the relative gaps of all attributes**

-> get a **global relative gap** between the specified sensor and the others.

-> Finally, transform relative gap -> level of similarity]0;1]

-> Return: map sensor & level of similarity

$$\lim_{x \to +\infty} f(x) = 0 \qquad f(0) = 1$$

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

worst-case complexity **O(max(s,m))**, ‘s’ is the number of sensors and ‘m’ the number of measurements.

Functional tests

1- Login fail
2- Access Sensor List
3- Retrieve sensor’s measurements
4- Run malfunctioning sensor detection analysis
5- Mark specific sensor as unreliable (Only for government users)
6- Compare the similarity between the other sensors
7- Consult a sensor: failed attempt
8- Retrieve quality of air at a given position and time
9- Access the list of owned air cleaners (Only for providers)
10- Access individual users list (Only for the government agency)
11- Access all air cleaners providers list (Only for the government agency)

Thank you for listening

Feel free to ask questions