Revue

Mattia Matteini
mattia.matteini@studio.unibo.it

Alberto Paganelli
alberto.paganelli3@studio.unibo.it

March 2023

Revue is a real-time video surveillance and environment monitoring system. It is designed to be used in multiple scenarios such as home, office or warehouses following the user needs. Composed by multiple features, it permits high modularity, allowing a more complex usage through the video streaming detection of predefined set of object classes and sending alerts to the user or simply to monitoring camera streams.

1 Requirements

The goal of the project is to develop a distributed software system which is able to monitor the environment of a certain area through sensors and cameras, providing real-time data and video streaming.

Moreover, to enhance the usefulness of the system, it should also be able to notify the user when specificconditions are met. These conditions include detecting if sensor data exceeds a predetermined range or if the camerarecognises a particular object. This notification feature ensures that the user is promptly informed about critical events or anomalies in the monitored environment.

The outcome should be a reliable system adaptable to different scenarios, such as smart cities, industrial, or simply home monitoring.

In the following are listed the main requirements of the system.

1.1 User Requirements

1. The user can authenticate to the system through a web interface.
2. The user can monitor the environment data produced by the sensors.
3. The user can monitor the video stream produced by the cameras.

4. The user can add/delete a device to the system.
5. The user can enable and disable a device.
6. The user can modify a device configuration.
7. The user can add/delete a security rule regarding a camera/sensor.
8. The user can modify a security rule.
9. The user can delete received notifications.
10. The user can consult the history of produced data.

1.2 System Requirements

1. The system grants access only to authenticated users.
2. The system provides a web interface as an entry point for the user.
3. The system generates video and data streams.
4. The system monitors streams in order to detect anomalies.
5. The system notifies the user when a security rule is violated.
6. The system persistently stores produced data.

1.3 Non-functional Requirements

1. The system should work even though the recognition component is down or not deployed.
2. The system should work even though the component responsible for the storage of the data is down or not deployed.
3. The system should be replicable in order to increase the availability of the system.
4. The system should work even though the component responsible for the authentication is down.
5. The video compartment should be independent of the data compartment and vice versa.

1.4 Implementation Requirements

1. The recognition component of the system should be implemented in Python.
2. The frontend of the system should be implemented using Vue 3 and Typescript.
3. The storage of data should be implemented using a NoSQL database.
4. The system should be implemented using a microservices architecture.
5. The system should be deployed using Docker.

1.5 Scenarios

The system can be used in various scenarios, depending on the user needs. This system is designed to be used by multiple types of users, from a private user to a company director. In the sections below, we will describe two main of the possible scenarios in which the system can be used.

In the simplest scenario, the system is used by a private user, who wants just to monitor his home or a particular environment without the necessity to recognise the objects in the video. In this case, the user can just rely on the camera, monitoring the home or a proprietary field. The user is free to monitor the live video by the camera whenever and wherever he/she wants, using the browser on the smartphone. The user can also set up sensors to monitor data from the environment.

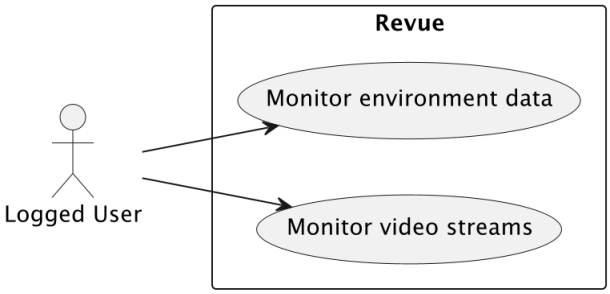

Figure 1: Simple Use Case

A more complex scenario could involve both sensor and camera usages with the support of a neural network to detect intrusion. For example, the director of food wholesale could monitor the temperature of the warehouse and the presence of unauthorised people during the night. In this case, the recognition part of the system is necessary to detect whenever an intrusion occurs. Moreover, supply chain monitoring can be done for who's needs to ensure that the temperature of the warehouse is always in the right range and be alerted whenever the temperature exceeds a certain range to detect or prevent problems.

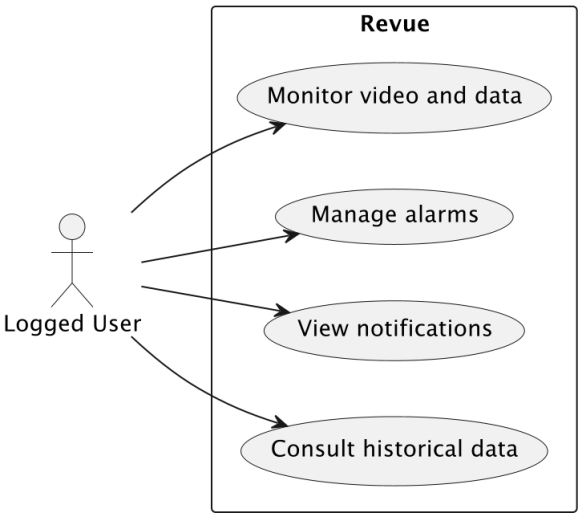

Figure 2: Advanced Use Case

1.6 Self-assessment policy

The project can be considered completed when **all** requirements are satisfied.

In particular, non-functional requirements aim to grant a good general quality of the system. This will be achieved also using automated tests that will ensure, among other things, the quality code production.

2 Requirements Analysis

Drawing up requirements, one relevant thing considered is the recognition feature. This is supposed to be developed using Python and exploiting its main libraries for video processing and object recognition, in order to minimise the abstraction gap.

Another important aspect is the handling of video streams. In fact, to facilitate development, an implicit requirement is necessary: the use of an ad hoc *media server*. This permits also to improve the compatibility permitting to produce and consume using different protocols.

Eventually, internal communications between devices and services need to be guaranteed. This implicitly leads to the use of a message broker (*Kafka*) which guarantees better scalability and at least one message delivery policy.

3 Design

3.1 Architecture

The overall system is designed with microservices architecture. This choice helped us to increase modularity, scalability, reliability and fault tolerance.

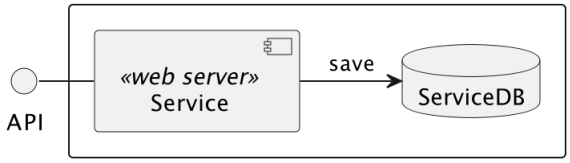

Figure 3: Microservice components

The system is composed of the following microservices:

- **Authentication Service**: responsible for the authentication and access control.
- **Monitoring Service**: responsible for managing devices and environment data.
- **Recognition Service**: responsible for the recognition of the objects in the video streams.
- **Alarm Service**: responsible for the management of the security rules and anomalies.
- **Notification Service**: responsible for sending notifications to the user.
- **Log Service**: responsible for the persistent saving of data.

Each microservice consists of a web server exposing REST APIs and a database to store the data (Figure 3).

Moreover, other components are necessary to make the system work:

- **Frontend**: provides to the user the web interface to interact with the system.
- **Sensors**: capture the environment data and send them to the rest of the system.
- **Cameras**: capture the video streams and send them to the rest of the system.
- **Media Server**: used to consume the produced video streams and make them available using different protocols.
- **Broker**: used to manage some internal communications.

3.2 Structure

Which entities need to be modelled to solve the problem? (UML Class diagram)

How should entities be modularised? (UML Component / Package / Deployment Diagrams)

3.3 Behaviour

How should each entity behave? (UML State diagram or Activity Diagram)

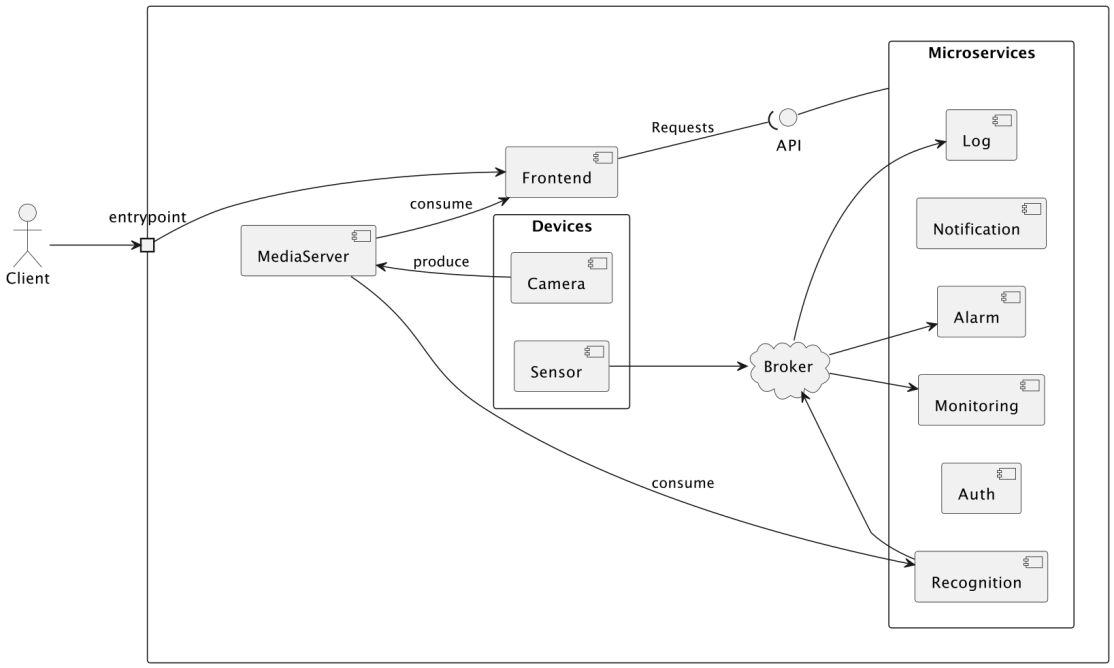

Figure 4: Revue Architecture

3.4 Interaction

How should entities interact with each other? (UML Sequence Diagram)

4 Implementation Details

Just report interesting / non-trivial / non-obvious implementation details.

This section is expected to be short in case some documentation (e.g. Javadoc or Swagger Spec) has been produced for the software artefacts. This this case, the produced documentation should be referenced here.

5 Self-assessment / Validation

Choose a criterion for the evaluation of the produced software and **its compliance to the requirements above**.

Pseudo-formal or formal criteria are preferred.

In case of a test-driven development, describe tests here and possibly report the amount of passing tests, the total amount of tests, and possibly the test coverage.

6 Deployment Instructions

Explain here how to install and launch the produced software artefacts. Assume the softaware must be installed on a totally virgin environment. So, report **any** configuration step.

Gradle and Docker may be useful here to ensure the deployment and launch processes to be easy.

7 Usage Examples

For every usage, the system require a login for security reasons.

LOGIN PIC

In the simplest scenario the user can see the section designated for sensor environment data or the camera video streams consultation.

HOME PIC e CAMERA PIC

In the more complex scenario the user can add security rules in order to detect exceeding values or unauthorized objects in the video streams. When adding a new security rule the user can choose the object class to detect or the threshold value (for a particular measure) in addition to the device and time slot in which the rule is active.

SECURITY RULES PIC

Both usage of the system consent to the user to add, delete or modify a device or a security rule configuration.

UPDATE DEVICE E SECURITY RULES PIC

Moreover, the user can consult the history of produced data or all the notifications received by the system.

HISTORY PIC e NOTIFICATIONS PIC

8 Conclusions

Recap what you did

8.1 Future Works

Recap what you did *not*

8.2 What did we learned

Racap what did you learned

Stylistic Notes

Use a uniform style, especially when writing formal stuff: X, X, $\mathbf{X}$, $\mathcal{X}$, $\mathtt{X}$ are all different symbols possibly referring to different entities.

This is a very short paragraph.

This is a longer paragraph (notice the blank line in the code). It composed by several sentences. You're invited to use comments within `.tex` source files to separate sentences composing the same paragraph.

Paragraph should be logically atomic: a subordinate sentence from one paragraph should always refer to another sentence from within the same paragraph.

The first line of a paragraph is usually indented. This is intended: it is the way LaTeX lets the reader know a new paragraph is beginning.

Use the `listing` package for inserting scripts into the LaTeX source.