Revue

Mattia Matteini
mattia.matteini@studio.unibo.it

Alberto Paganelli
alberto.paganelli3@studio.unibo.it

March 2023

Up to ∼2000 characters briefly describing the project.

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 specific conditions are met. These conditions include detecting if sensor data exceeds a predetermined range or if the camera recognises 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 in order to reduce the abstraction gap.
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 offers various ways of usage.

In the simplest scenario, the user can just rely on cameras, maybe monitoring the home or a proprietary field. In this case, the user is free to monitor the video registered by the camera whenever and wherever he/she wants, using the browser on the smartphone.

Similarly, the system can be used just with sensors, it is a choice based on the user needs

A more complex scenario could involve both sensor and camera usages. 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 the temperature exceeds a certain range or the camera records objects that are not supposed to be there.

It should describe *how* and *why* a user should use / interact with the system.

Detailed description of the project goals, requirements, and expected outcomes. Use case Diagrams, examples, or Q/A simulations are welcome.

1.6 Self-assessment policy

- How should the *quality* of the *produced software* be assessed?
- How should the *effectiveness* of the project outcomes be assessed?

2 Requirements Analysis

Is there any implicit requirement hidden within this project's requirements? Is there any implicit hypothesis hidden within this project's requirements? Are there any non-functional requirements implied by this project's requirements?

What model / paradigm / techonology is the best suited to face this project's requirements? What's the abstraction gap among the available models / paradigms / techonologies and the problem to be solved?

3 Design

This is where the logical / abstract contribution of the project is presented.

Notice that, when describing a software project, three dimensions need to be taken into account: structure, behaviour, and interaction.

Always remember to report **why** a particular design has been chosen. Reporting wrong design choices which has been evalued during the design phase is welcome too.

3.1 Structure

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

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

3.2 Behaviour

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

3.3 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

Show how to use the produced software artefacts.

Ideally, there should be at least one example for each scenario proposed above.

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.