Technical Report **- Product specification**

SensorSafe

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 17 December 2023
Students:	108317: Miguel Aido Miragaia 107572: Gonçalo Rafael Correia Moreira Lopes 108536: Cristiano Antunes Nicolau
Project abstract:	Interactive system to manage Humidity/Temperature/Smoke Sensors

Table of contents:

1 Introduction

The creation of a system for tracking temperature, humidity, and smoke seems to be the ideal project. It's valuable in logistics, healthcare, security, and manufacturing. We were able to implement some of the learned concepts in the IES course, and learn about new technologies, the different types of Architectures. We had also have the opportunity to see the behind the scenes of building a project with the the collaboration of a team where every member as his own role, the efficiency of working with GitHub to control the workflows, among using other for other functionalities.

In a world where real-time monitoring is becoming the norm, our system, SensorSafe, stands as a solution poised to make a substantial impact. The ability to manage and control sensors in a building opens avenues for enhanced safety, comfort, and operational efficiency. The vision extends beyond individual users to encompass the broader spectrum, from meticulous Building Managers to residents keen on elevating the quality of their living spaces.

Beyond the technical intricacies lies the true essence of our project — its potential impact on end-users. As we embark on this journey, SensorSafe not only promises to meet the current demands of the market but sets the stage for future scalability and innovation.

This is not just a project; it's a forward-looking exploration into the convergence of technology and human-centric solutions.

Team Roles

Team Manager: Miguel Miragaia (miguelmirgaia@ua.pt);

Architect: Gonçalo lopes (goncalorcml@ua.pt);

Product Owner: Cristiano Nicolau (cristianonicolau@ua.pt);

DevOps Master: Miguel/ Gonçalo/ Cristiano;

2 Product concept

Vision statement

SensorSafe is not just a system; it's a visionary approach to redefine how we interact with and manage the environments we inhabit. In a world where safety, comfort and control are primordial SensorSafe emerges. It has the objective of managing all the Sensors in a building. This system was design to help different types of users, from house owners to the Building Manager, having control of the temperatures/humidity of his own divisions/apartments or even for a security context by having control of a smoke sensor to prevent a fire.

In an era where technological solutions often add layers of complexity, SensorSafe stands out for its clarity and user-centric design. We aspire to revolutionize how users perceive and interact with sensors by offering a unified system that transcends the boundaries of traditional applications. SensorSafe is not just about managing sensors; it's about empowering users with the ability to shape their living and working environments to meet their unique needs.

This system offers a full control of every type of sensors, allows users to have an app to manage them all, to prevent the need of the users to own multiple apps to control each one.

Personas and Scenarios

Personas:

- **Name:** John Smith
- **Age:** 47
- **Gender:** Male
- **Role:** Building Manager
- **Background:** John has several years of experience managing residential and commercial buildings. He is responsible for ensuring the safety and comfort of all residents and occupants.

- **Name:** Peter Williams
- **Age:** 37
- **Gender:** Male
- **Role:** Resident on the third floor of the building
- **Background:** Peter is a resident on the third floor of the building. He is deeply concerned about the safety and comfort of his family. Peter is interested in installing sensors in his apartment to enhance fire safety, monitor humidity in the bathrooms, and control the temperature to create a comfortable living environment for his family. He is actively seeking technological solutions to achieve these objectives within his apartment.

Product requirements (User stories)

User Story 1: As the Building Manager, John requires the ability to configure customized alerts for specific divisions within the building. He should receive immediate notifications if the temperature in the boiler room exceeds a predefined threshold. This feature empowers John to proactively prevent potential equipment damage.

System: Using SensorSafe John can define specific temperature and humidity thresholds for each division.

User Story 2: In his role as Building Manager, John seeks access to historical data regarding smoke sensor events. This access allows him to analyze trends and identify potential areas of concern within the building.

System: The data is presented in a clear and visual format, helping John in identifying trends or patterns.

User Story 3: John intends to automate the generation of reports on sensor status and maintenance schedules for management purposes.

System: There are some types of reports (sensor status, maintenance schedule). Reports are generated when the user pretends, by clicking on a section to create and download it.

User Story 4: As a resident on the third floor, Peter wants to install smoke detectors in multiple rooms of his apartment to ensure the safety and comfort of his family. He would like to receive immediate notifications on his smartphone in case of smoke detection in any area of his apartment.

System: The system allows for the installation of smoke detectors in specific rooms with real-time notifications and monitoring to ensure Peter and his family's safety.

User Story 5: Peter envisions automation for the various sensors installed in different rooms of his apartment. Specifically, he wants to configure the system to activate the air conditioner when the temperature falls below a certain threshold, trigger the bathroom fan if humidity levels rise beyond a specified maximum, and the activation of the fire sprinklers in case of smoke detection .

System: The system will provide a user-friendly interface to set personalized automation rules for the sensors in their apartments.

3 Architecture notebook

Key requirements and constrains

Our system's data flow depends on data generated that is registered by a set of different sensors (humidity, temperature, smoke). It is guaranteed that each of these sensors have common characteristics but may also have specific characteristics. Therefore, modulating a relational database is very complicated, as it would be necessary to create several tables to adjust each of the sensors to their characteristics. Because of this, we decided to use a document-based *NoSQL* database - ***MongoDB***. The Mongo database gives us the possibility of storing all sensors in the same collection, without the need to create several tables, where would be necessary in a *relational database*.

With the increase in the number of devices registered in our system (sensors) the amount of data received will also increase, which means that the processed data will also increase dramatically. Because of this, we need to have a *robust* and *scalable* system to process all data without compromising the *availability* and *integrity* of our system. We implemented an event-driven architecture. This allows us to create and deploy more processors/workers to process the growing amount of data - **scalability**. This approach improves the availability of our system. Each type of data (temperature, humidity, smoke) has a large set of processors available, meaning that if one of them breaks, there will always be another to replace it and, therefore, keep the system available.

Our internal services - processors and generators - carry out administrative operations. These operations must be handled and made available by our Middleware, which must be capable of performing aggregation operations and sending notifications to front-end integrations.

API endpoints use authorization and authentication mechanisms to protect the API from unauthorized access and operations. With this, we used access tokens to protect endpoints.

The application is to be also accessible from the web browser, allowing our users to connect to their accounts whenever and wherever they want. The application also needs to perform well as it needs to display and change a lot of data in real time.

Architectural view

The data generation layer implements some virtual agents in Python. Each agent represents and simulates a real-world sensor – there are agents for temperature data, humidity, and smoke. The data from those agents is sent through message queues to the backend for processing (RabbitMQ, in our case).

The database stores all our persistent data. It was first thought to be implemented in MySQL, using a relational-type database. However, after taking a closer look at our data classes, we realized that a documentational database would be more fitting, so we decided to go with MongoDB, to store the results using the middleware/services layer as a wrapper.

The Middleware layer is written in Java and integrated within the Spring Boot application. The Middleware also performs some aggregation operations such as calculating the statistics of a room. It also has the responsibility to notify the front-end when some important change occurs in the data in general. This Middleware exposes a Spring Controller (protected by Authentication and Authorization mechanisms) which allows our services to contact it with ease.

To secure our API, we are also using Authentication and Authorization mechanisms. The Authentication is implemented using access tokens (JWT) that are emitted for a user when he logs in with his username and password. This token is then stored in the local storage of the browser and used as Authentication for the user endpoints. As stated before, the Middleware endpoints are private and should only be used by our internal services.

To protect our users, their sensitive data (i.e. passwords) is also encrypted in our database using *Bcrypt*.

The API also exposes some public methods where no Authorization is required; that is the case are the register and login endpoints.

The system front-end was implemented using ReactJS once it eases the creation of a SPA (Single-Page application).

To connect our integrations and clients to the system, we use a REST API written with Spring Boot (Java) which also connects to our Middleware to access the database. Our applications/clients will then use HTTP requests to query the API and get the necessary information to present in the front-end.

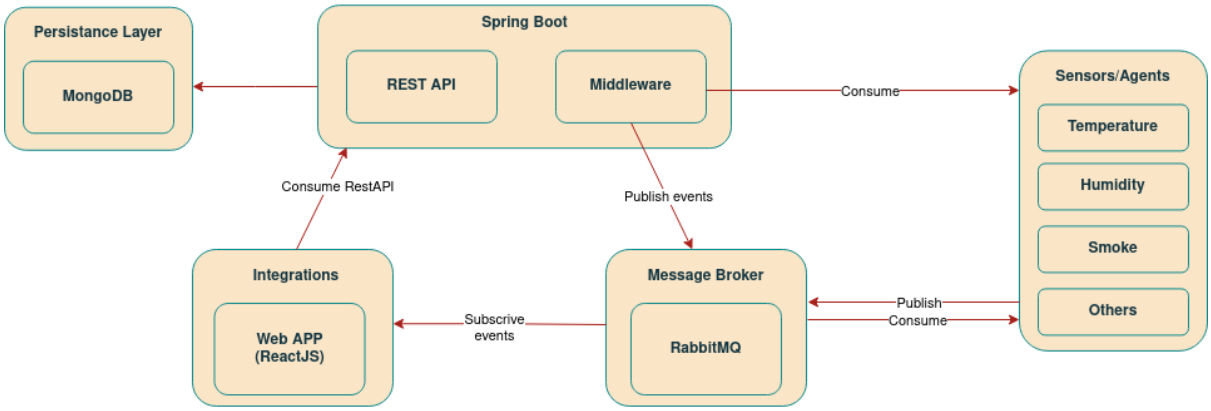
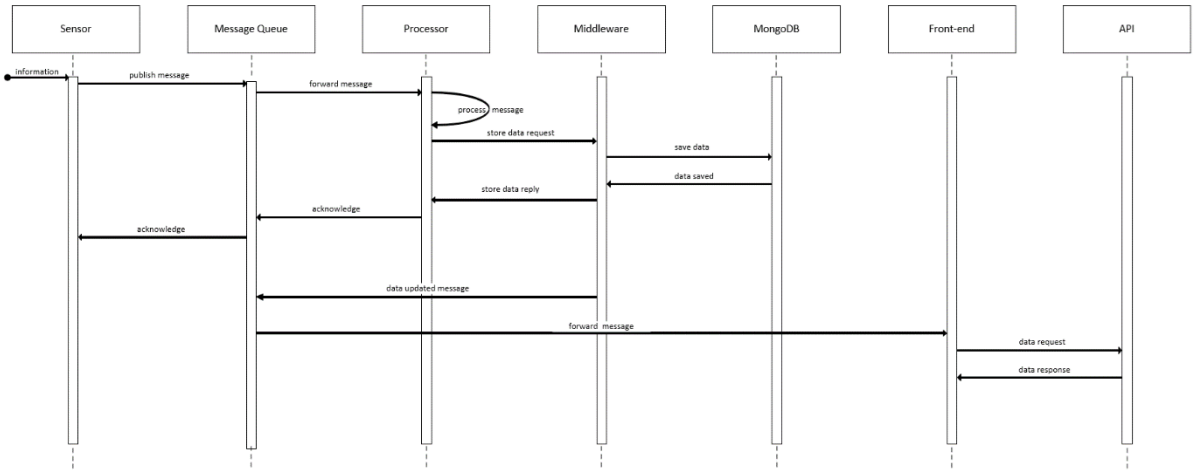


Diagram of the planned architecture.

Module interactions

The communication between the sensors/agents module and the processors module is facilitated through the RabbitMQ library, employing a Message Queue to establish an asynchronous channel of interaction. Periodically, the Message Queue relay messages to the processors, where the received data will be processed and subsequently stored in the MongoDB database using the Middleware interface.

The interactions described can be visualized through the depicted sequence diagram.

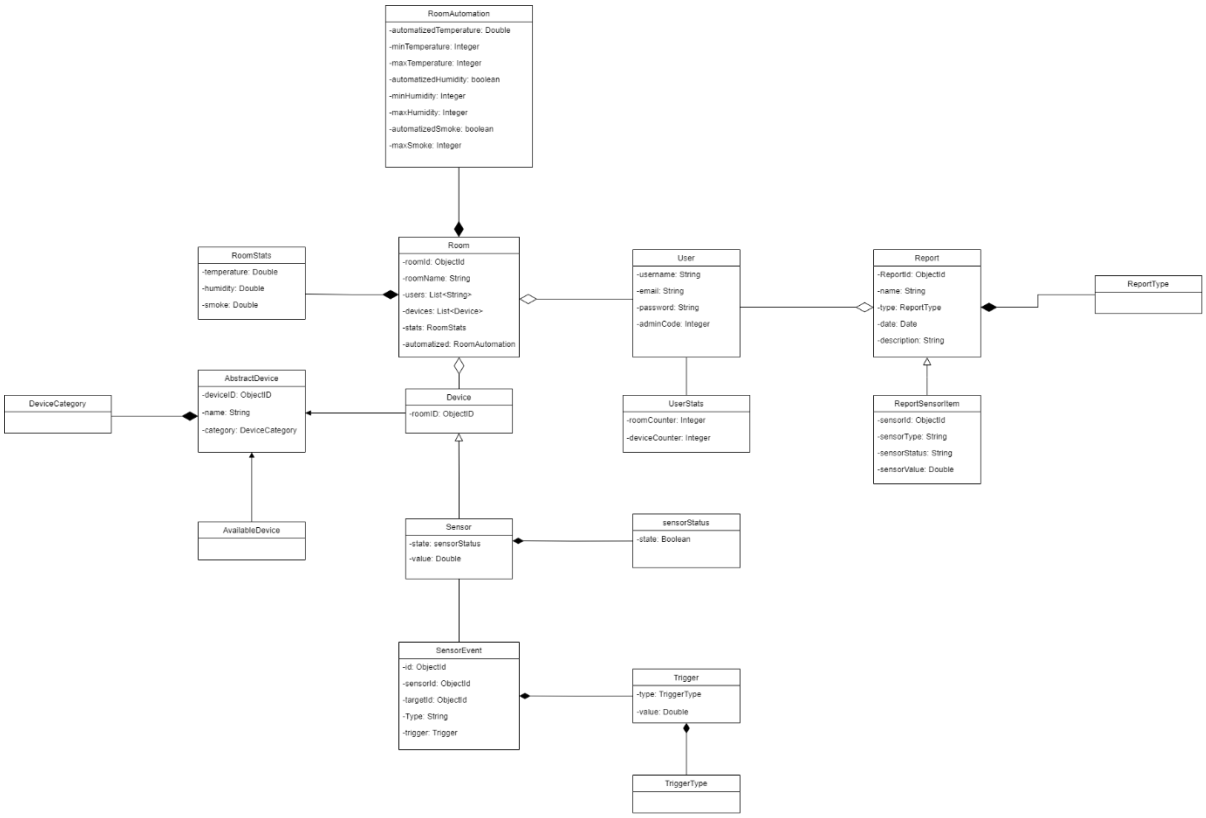


Sequence diagram for data processing.

The utilization of reply messages provides a safeguard against data loss during the process. In the event of a data sending failure, the sender will make repeated attempts until either success is achieved or a predefined limit of retries is reached.

4 Information perspective

To elucidate and clarify the concepts of the domain and their attributes, we employed a domain model, specifically utilizing UML classes. This model serves as a visual depiction of conceptual classes or real-situation entities within a domain, showcasing their various interconnections.



UML domain diagram.

Our diagram uses the entities stated to represent our data:

User

The user is the entity that is going to interact with SensorSafe (web application).

UserStats

Entity that stores the rooms and devices counter that the user has access.

Room

This entity represents the various divisions of a house/building where devices are installed. In our application, each room has a list of associated devices and a list of users who have access to it. Each room contains their own statistics (RoomStats).

RoomAutomation

This entity represents the control of the statistic parameters of a room. Allows automation of temperature, humidity, and smoke based on predefined user defined input criteria.

RoomStats

This entity holds the up-to-date statistics for a room, including the present temperature, the current humidity, and Smoke levels.

Report

An entity that signifies an element about the devices report. It can encapsulate a variety of actions as defined by the ReportType.

ReportType

Specifies the various categories of available reports, maintenance, rooms and devices.

ReportSensorItem

A specialized form of ReportType designed to store historical events associated with changes in sensor data. The application generates and preserves a fresh ReportSensorItem whenever there is a modification in the data of a sensor.

AbstractDevice

Represents all devices (Sensor, “Device in general”) in the application.

Device

This is a subtype of AbstractDevice that represents a “Device in general” (not a sensor) in the application, e.g. air conditioner.

AvailableDevice

This is a subtype of Device, representing an available device. An available device implies that a user has the option to add it to a room.

DeviceCategory

This represents the different categories of devices. We have 4 categories on our app: temperature, humidity, smoke, and others.

Sensor

It is a subtype of Device and represents a sensor in our app.

SensorStatus

This entity will retain the sensor's state, indicating whether it is currently in use (it is associated to a room).

SensorEvent

This entity represents an event/trigger in the application.

Trigger

This entity stores the trigger of a given event. It stores the value at which the event should be triggered and the type of the trigger (instance of TriggerType).

TriggerType

It represents the different types of triggers. In our case, we are going to have triggers on a value higher, equals or less to a given limit, e.g. when the temperature of the room is lower than 20 we activate the air conditioner to heat the room

5 References and resources

https://www.mongodb.com/docs/manual/core/data-modeling-introduction/

https://www.tibco.com/reference-center/what-is-event-driven-architecture

https://www.rabbitmq.com/getstarted.html