



**UNIVERSITÀ DELLA CALABRIA**

**DIPARTIMENTO DI INGEGNERIA INFORMATICA,  
MODELLISTICA, ELETTRONICA E SISTEMISTICA**

**Telecommunication Engineering:  
Smart Sensing,  
Computing and Networking**

**Project Report**

**SMART MUSEUM  
ENVIRONMENTAL MONITORING**

**IoT Device Programming:  
Module 1&2  
2024-2025**

**Oliverio Ilenia - 263924**

**Damico Vincenzo - 269656**

**Alvarenga Ortezz Josseline Michelle - 251905**

**Prof. Giancarlo Fortino**

**Prof. Francesco Pupo**

# Contents

<b>1. Introduction.....</b>	<b>3</b>
<b>1.1 Goal.....</b>	<b>3</b>
<b>1.2 Requirements .....</b>	<b>4</b>
<b>1.2.1 Functional Requirements .....</b>	<b>4</b>
<b>1.2.2 Non-Functional Requirements .....</b>	<b>4</b>
<b>2. Architecture .....</b>	<b>5</b>
<b>3. Implementation.....</b>	<b>7</b>
<b>3.1 Hardware Setup .....</b>	<b>7</b>
<b>3.2 Software Setup.....</b>	<b>7</b>
<b>3.3 Features .....</b>	<b>12</b>
<b>4. Conclusion .....</b>	<b>13</b>

# 1. Introduction

Within a museum, the preservation of artworks depends not only on the direct care of the pieces themselves but also – and most importantly – on the quality of the environment in which they are displayed. Parameters such as temperature, humidity, and light intensity significantly affect the conservation state of materials: thermal fluctuations, air that is too dry or too humid, or excessive light exposure can cause irreversible damage to paintings, fabrics, manuscripts, and ancient objects. For this reason, the constant monitoring of environmental conditions is a priority for any museum institution committed to protecting its heritage.

In this context, the project falls within the scope of the Internet of Things, IoT, proposing a solution for environmental monitoring in a Smart Museum. The system developed enables the real-time collection of key environmental data through the use of distributed sensors, with the aim of providing a clear and continuous overview of the exhibition rooms' conditions, helping to prevent potentially invisible but progressive damage.

At this stage, the project focuses exclusively on monitoring, without directly intervening in the control of environmental parameters. However, such functionality represents a natural evolution of the system and will be discussed in the conclusions as possible future development.

## 1.1 Goal

The main goal of the project is the development of an IoT system for environmental monitoring in a Smart Museum, aiming to preserve artworks through non-invasive control of environmental conditions.

Specifically, the project focuses on:

- the use of low-power TelosB sensors managed through the TinyOS operating system;
- reliable data communication based on the MQTT protocol;
- a data pipeline for collection and storage in InfluxDB and data flow management via Node-RED;
- an interactive dashboard for real-time and historical visualization of environmental parameters developed with Grafana;
- a mobile app for monitoring developed with Android Studio;
- a cloud backend for user management and data synchronization through Firebase.

This integrated architecture aims to ensure continuous and reliable monitoring of environmental conditions, laying the foundation for future developments where the system could also actively intervene in controlling the exhibition environments.

## 1.2 Requirements

### 1.2.1 Functional Requirements

Among the functional requirements that the designed IoT system must guarantee are:

- **Environmental data detection:** real-time acquisition of key parameters through sensors distributed across the museum's exhibition rooms.
- **Reliable data transmission:** must ensure continuous and secure sending of data collected by the sensors to the central platform.
- **Data storage:** measurements must be stored in a time-series database to enable long-term monitoring.
- **Data flow management:** must manage the flow of data between sensors, database, and applications.
- **Visualization:** implementation of a dashboard for displaying environmental parameters.
- **Mobile access:** development of a mobile application that allows operators to monitor data from anywhere.
- **Authentication and synchronization:** includes authentication, authorization, and secure data synchronization.

### 1.2.2 Non-Functional Requirements

Among the non-functional requirements are:

- **Reliability:** the system must ensure operational continuity, stable performance over time, and minimal data loss.
- **Scalability:** must support the integration of additional sensors and users without compromising system performance or responsiveness.
- **Energy efficiency:** must use low-power sensors and protocols to reduce maintenance needs and ensure long-term operation without frequent battery replacements.
- **Security:** must protect data during transmission and storage, preventing unauthorized access through secure protocols and authentication systems.
- **Usability:** must ensure that the dashboard and mobile app interfaces are intuitive and easy to use for different types of operators.
- **Maintainability:** the system must be designed modularly to facilitate updates, maintenance, and future extensions.
- **Interoperability:** must ensure compatibility and integration with any existing environmental control platforms or systems.

## 2. Architecture

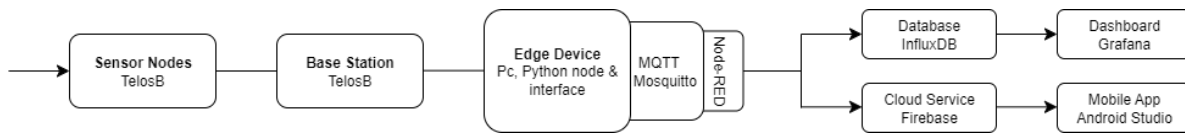


Figure 1

The system architecture, illustrated in Figure 1, is structured on multiple levels that collaborate with each other, from data acquisition to visualization. TelosB sensors, distributed throughout the museum rooms, detect environmental data and send it to the base station, which is also a TelosB device acting as a bridge to an edge device. The edge device locally processes the data using Python scripts and routes it to the MQTT broker, Mosquitto, facilitating efficient management and redirection of information flows through Node-RED. The data is then stored both in InfluxDB, locally, and Firebase, in cloud, the latter via an HTTP Request node that enables communication. Data from InfluxDB is used for the Grafana dashboard, while Firebase provides the data needed by the mobile app developed with JetPack Compose Paradigm in Kotlin. Finally, Firebase completes the infrastructure by offering cloud services for user management and real-time synchronization of information.

The following part describes each individual component and its role within the overall system architecture.

### TelosB sensors and TinyOS

The TelosB sensors represent the hardware foundation of the system, these are low-power energy devices designed to operate in distributed environments. Each sensor node is capable of detecting environmental parameters such as temperature, humidity, and luminosity, and transmitting them wireless mode to the base station, which is also a TelosB sensor, connected via USB to an edge device, typically a PC. The collected data are transmitted periodically and managed by TinyOS, an embedded operating system known for its efficiency in resource management and compatibility with wireless sensor networks.



Figure 2

### Python Script and MQTT Broker

Once the raw data are collected from the sensors and transmitted to the base station, these are forwarded to the edge device through the serial port. At this point, a Python script is used to process, decode, and format the data, making them readable and properly structured to be published on specific MQTT topics using the Mosquitto broker. This broker acts as a communication channel between the physical devices and the software infrastructure, ensuring efficient and real-time data transmission. The data transfer is carried out using the MQTT, Message Queuing Telemetry Transport, protocol which is lightweight and reliable.

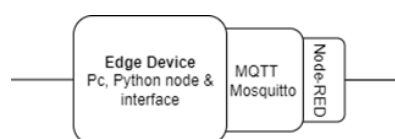


Figure 3

## Node-RED

Node-RED represents the central point of the software part, through its graphical interface, it enables the modeling and management of data flow via interconnected nodes, each with a specific function. Using the "mqtt in" node, Node-RED subscribes to the MQTT topics and acquires the data published by the Python script. These data are then processed in the "function" nodes, where it can be filtered, transformed, and subsequently routed to the InfluxDB database for local storage. After further processing, the data are sent to the Firebase cloud service using the "HTTP Request" node.

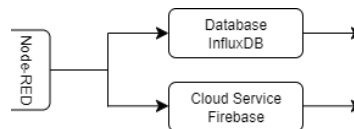


Figure 4

## InfluxDB

For local data storage is used InfluxDB, a time-series database suitable for managing and storing large quantity of real-time data, such as those generated by environmental sensors. InfluxDB is suitable for the project because it allows optimal data management due to its temporal indexing capability, ensuring not only quick access but also the ability to perform complex queries on historical data.

## Grafana

Integrated into the project for data visualization and analysis, Grafana is an open-source platform that allows for the creation of interactive and customizable dashboards. Through it is, therefore, possible to observe in real time the trend of environmental variables, the history of measurements and the detection of any anomalies. This interface, therefore, represents an important control tool both for system maintenance and for evaluation of the museum environment.



Figure 5

## Firebase

The system also provides for cloud synchronization of the data using Firebase, a real-time development platform. This manages the reception of environmental data coming from Node-RED via HTTP messages and automatically synchronizes them. In addition, Firebase manages user authentication, and the storage of data related to the mobile application.

## App Android

Finally, to enable museum staff to continuously and remotely monitor environmental parameters, an Android application has been developed and connected to the cloud Firebase. Thanks to real-time synchronization, the app allows users to receive and immediately display updated data, through a simple and accessible interface, even outside the local network.



Figure 6

### 3. Implementation

In this chapter, the implementation of the developed IoT system is described in detail. It presents the components that were actually built and then integrated into the working prototype of the environmental monitoring system for the Smart Museum. Unlike the previous chapter, which illustrated the overall architecture and the functional role of each hardware and software component, this section provides a concrete overview of how these elements were effectively used in practice.

#### 3.1 Hardware Setup

For environmental data collection, three TelosB sensors were used:

- Two sensor nodes, programmed to detect environmental parameters such as temperature, humidity, and luminosity. These were placed in different locations to simulate two distinct areas of the museum hall.
- One TelosB node configured as a base station, connected via serial port to a central computer to receive and forward the data transmitted by the remote nodes.

The choice to use TelosB devices was made due to their energy efficiency, their availability, reliability and compatibility with wireless sensor networks.

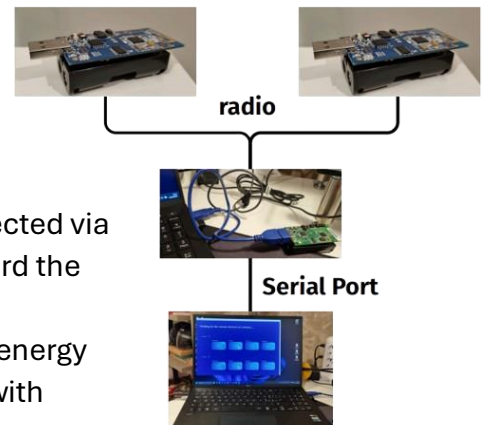


Figure 7

Each sensor node sends data regularly each second to base station via a wireless connection. Furthermore, using this platform introduced to a different programming paradigm: event-driven programming.

#### 3.2 Software Setup

##### TinyOS Configuration

Each TelosB sensor was programmed with TinyOS, using the nesC language.

The implemented code enables the acquisition of environmental parameters – temperature, humidity, and luminosity – detected by the sensors placed in the museum’s indoor areas. The data are then packed into a serial-compatible format and transmitted periodically to the base station.

The base station, in turn, receives this data in real time and forwards them to the central machine via the serial port – in this case, a PC.

This operating system operates around a hierarchical concept, in this hierarchy, components higher up can only receive events from components below them and issue commands to them. When the TelosB platform is programmed, the components are created and for each of them is necessary to provide:

- A **configuration file**, specifying its connections to other components.
- A **makefile**, providing crucial information to the compiler.
- An **implementation file**, indicating its behavior.

In this project, are designed two custom components: one for the sensor nodes and one for the base station:

- **Sensor Node:** SensingAppC.nc (configuration), SensingC.nc (implementation), SensorMsg.h, MAKEFILE
- **Base Station:** SimpleReceiverAppC.nc (configuration), SimpleReceiverC.nc (implementation), SensorMsg.h, MAKEFILE

Additionally, because it is used radio to send environmental information messages from sensor nodes to the base station, it is also needed to provide them with the structure of the packet payload, which is defined in the SensorMsg.h file.

Once the base station receives this information via radio, it writes it to the serial port of the Edge device.

```
f~Ed-----
id: 1 ~Ed Temp: 291 Hum: 2172 Lumj~Ed: 91 TimeStamp: 1951721
Message sent: {"idSensor": 1, "timestamp": "2025-07-04_15:50:15", "temperature": 29.1, "humidity": 70.8, "luminosity": 91}
~Ed-----
id: 1 ~Ed Temp: 291 Hum: 2169 Lum~Ed: 92 TimeStamp: 1952721
Message sent: {"idSensor": 1, "timestamp": "2025-07-04_15:50:16", "temperature": 29.1, "humidity": 70.7, "luminosity": 92}
~Ed-----
id: 1 ~Ed Temp: 291 Hum: 2166 Lum)~Ed: 62 TimeStamp: 1953721
Message sent: {"idSensor": 1, "timestamp": "2025-07-04_15:50:17", "temperature": 29.1, "humidity": 70.6, "luminosity": 62}
w6~
```

Figure 8

## Data processing with Python

On the Edge device, a Python script runs that decodes, analyzes, and displays the data on an interface, then publishes it to the MQTT broker, Mosquitto.

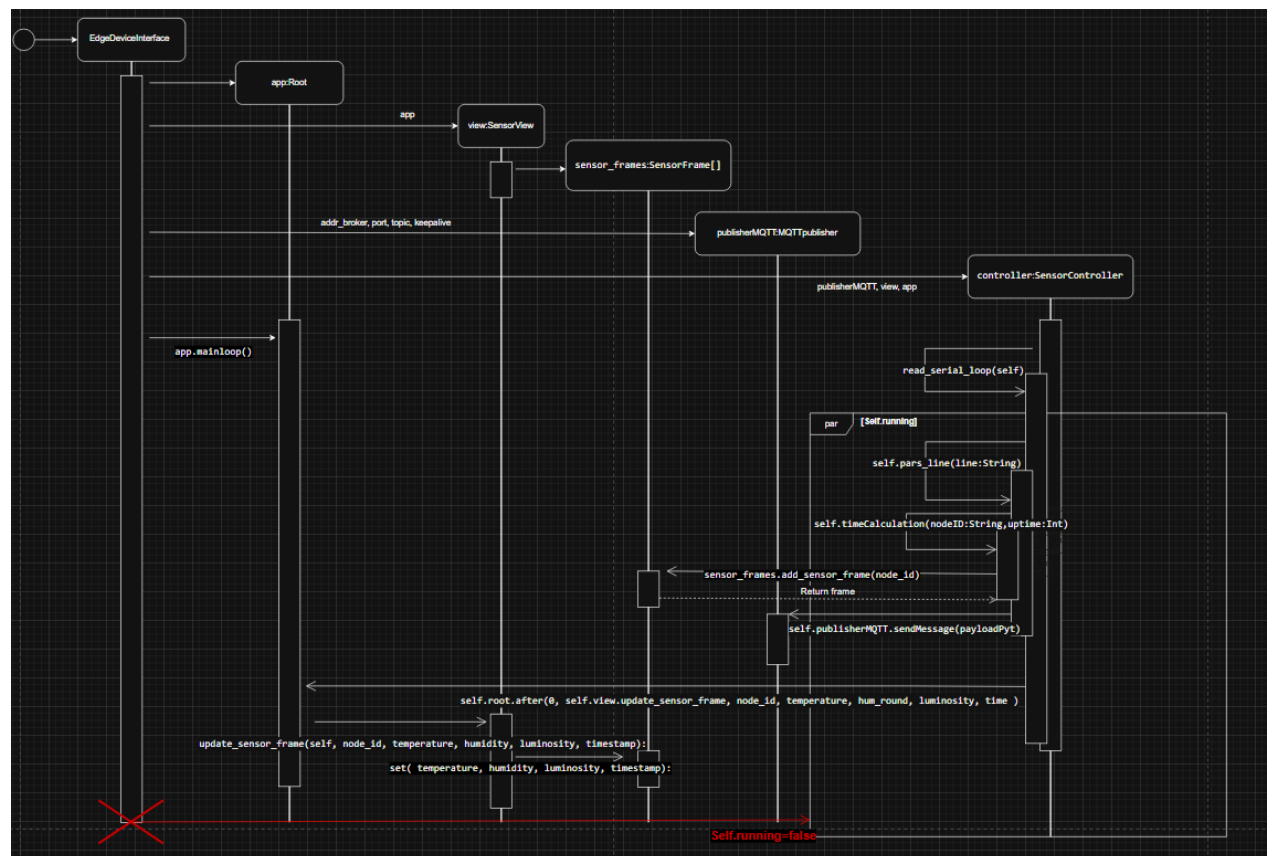


Figure 9



Let's look at how the program works with a sequence diagram: as the diagram illustrates, there are two threads working in parallel: the main thread, which runs the user interface, and a "side thread", indicated by the par frame. This side thread retrieves data from the serial port, decodes and adjusts it, and then sends it to the MQTTpublisher object. This object subsequently publishes the data on the Mosquitto MQTT broker, which is configured through Node-RED. The interface follows the MVC, Model-View-Controller, pattern, where the model is implemented via MQTTpublisher and managed by Node-RED, which saves all information to InfluxDB. The controller part is implemented within the side thread. It takes raw data from the serial port, which then needs to be decoded. After decoding, it is managed noise on the port using regular expressions, regex, extracting only the unprocessed data. This data, after some adjustments, is then ready to be sent to the MQTT publisher.

Regarding the interface, it is designed to easily scale and receive information from multiple sensor nodes without modifications.

### MQTT and Mosquitto Broker

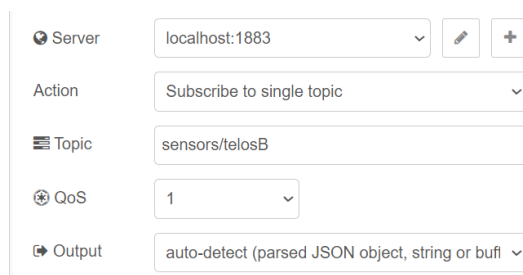
To enable communication between the various components of the system, the MQTT protocol was chosen due to its simplicity, lightweight nature, and reliability.

In the project implementation, the Mosquitto MQTT broker was installed and configured locally on the central PC. Its role is to manage, in real time, the message exchange between the TelosB base station, via the Python script, and Node-RED, which will forward the processed data to the subsequent modules.

The data received from the serial port is read by the Python script, decoded, and formatted before being published to specific MQTT topics.

```
addr_broker = "localhost"
port = 1883
topic = "sensors/telosB"
keepalive=60
```

Figure 10



The screenshot shows the configuration for an MQTT input node in Node-RED. The 'Server' field is set to 'localhost:1883'. The 'Action' dropdown is set to 'Subscribe to single topic'. The 'Topic' field is set to 'sensors/telosB'. The 'QoS' dropdown is set to '1'. The 'Output' dropdown is set to 'auto-detect (parsed JSON object, string or buff)'. There are also icons for editing and adding new configurations.

Figure 11

The Mosquitto broker is responsible for receiving these publications and sending them to all clients subscribed to the same topic. This subscription can be observed in the MQTT input node within Node-RED, which runs on the same machine.

In this implementation, the MQTT messages are published with a Quality of Service (QoS) level of 1, which guarantees that messages are delivered at least once to each subscriber, providing a good trade-off between reliability and performance.

The use of this local broker ensures, therefore, an efficient and real-time data flow between the physical and software layers of the system.

## Node-RED flow design

The Node-RED environment was used to organize the data flows through its graphical interface. The following nodes were used and configured:

- the "mqtt in" node, to subscribe to the MQTT topics published by the Python script;
- the "function" nodes, to filter and transform the incoming data depending on its destination;
- the "link out" and "link in" nodes, to connect certain nodes in a clean and organized way;
- the "influxdb out" node, to send the message containing the data to the local InfluxDB database;
- the "http request" node, to send the data to the Firebase cloud in JSON format.

The processing and transformations performed on the incoming messages in the different function nodes are as follows:

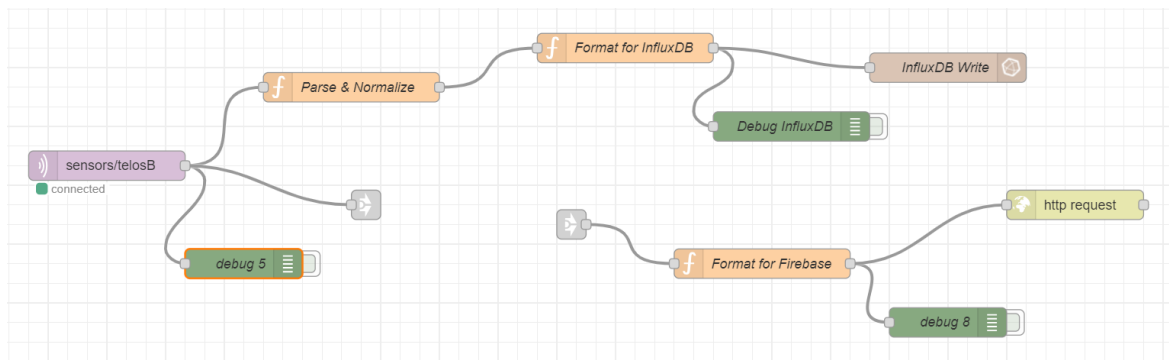


Figure 12

## Local Data Storage with InfluxDB

For the local storage of the data collected by the sensors, the InfluxDB time-series database was used, also installed on the PC. This database organizes the data received according to timestamps, allowing for effective optimization.

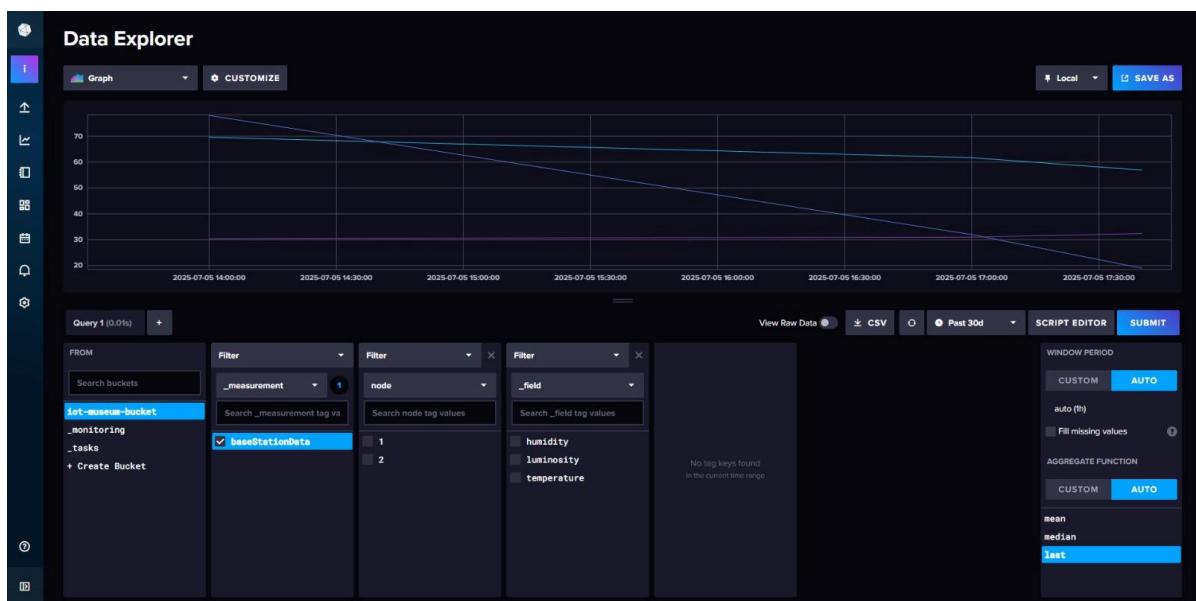


Figure 13

## Data Visualization with Grafana

The data stored in InfluxDB are visualized through Grafana. A customized dashboard is configured to display the current sensor values in real time, historical graphs of environmental data, and any alerts or anomalies that may occur.

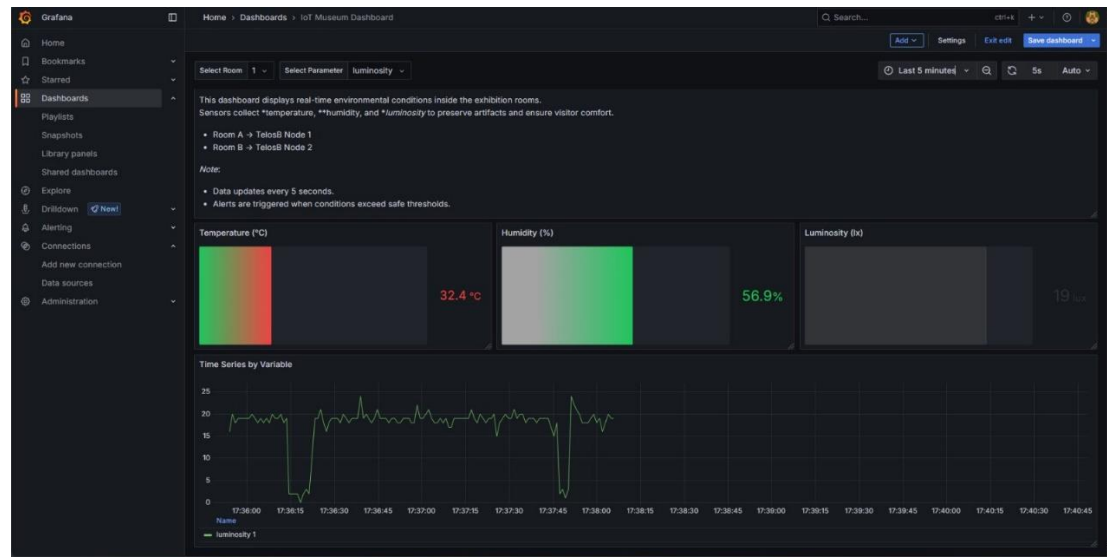


Figure 14

## Real-Time synchronization with Firebase

In this project, to enable cloud synchronization of data and remote access, Node-RED sends the data to Firebase Realtime Database via HTTP requests. The database URL on Firebase must also be present in the HTTP Request node within Node-RED. The data is structured in JSON format before being sent and is stored in a tree structure that is easily navigable and visible on the platform. In addition to serving as a real-time database, Firebase also manages the user authentication system, using email and password credentials.

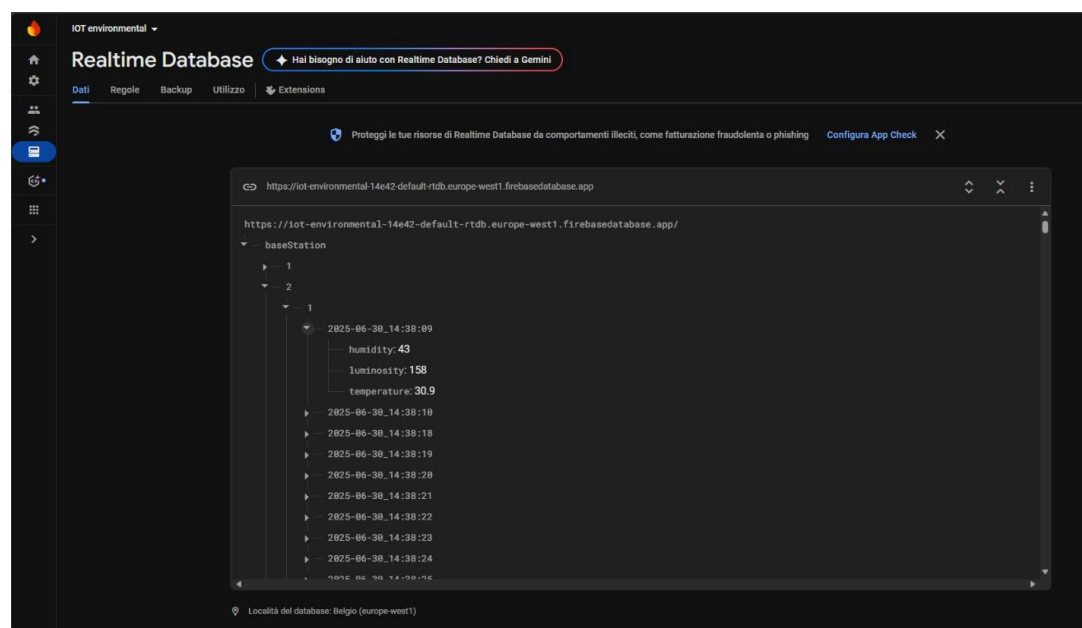


Figure 15

## Mobile App Integration

The final step of this project is the development of an Android app to allow museum operators to remotely monitor environmental conditions. Thanks to real-time synchronization with Firebase, the app displays up-to-date data within a simple and

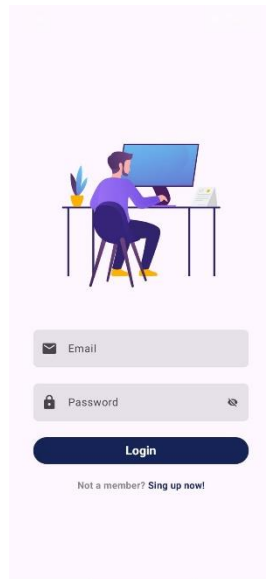


Figure 16



Figure 17

intuitive interface. In this simulation, the interface presents the list of base station and for each of them the list of nodes, in this case, two sensors – Node 1 and Node 2 – along with their respective real-time environmental parameters and the timestamp associated with each reading.

The app is divided into an initial dashboard where users can register or log in using email and password credentials, and a main dashboard where the current sensor values are displayed.

## 3.3 Features

The implemented system offers several key features that enhance its functionality and usability within a smart museum:

- **Real-time data visualization:** A dedicated Grafana dashboard has been set up to provide real-time visualization of environmental parameters collected by the sensors. This allows for continuous monitoring and immediate detection of anomalies or variations in the museum environment.
- **Real-time synchronization with mobile devices:** Through the integration with Firebase Realtime Database, the system ensures continuous and instantaneous synchronization of data with Android mobile devices. This enables museum staff to access updated information at any time and from any location.
- **Cloud integration for data storage and sharing:** The system is designed to store data both locally, using InfluxDB, and in the cloud, through Firebase. This approach facilitates data sharing and remote access, making the system more flexible and collaborative.
- **Scalability:** The architecture of the system has been designed to be easily scalable. Additional sensors or modules can be integrated with minimal configuration, allowing the system to adapt to museums of different sizes or to future expansions of the monitoring infrastructure.

## 4. Conclusion

At the end of the work carried out, the system developed demonstrates how an IoT-based approach can offer effective and continuous environmental monitoring within a museum, with particular focus on the preservation of artworks through a non-invasive control of environmental conditions.

The implemented architecture – based on low-power sensors, lightweight communication protocols, and tools for real-time visualization and analysis – proved to be stable, scalable, and easily extendable.

However, looking ahead, the work carried out represents only a first step towards further improvements. Among the most relevant future developments are:

- the implementation of **actuation mechanisms** capable of adjusting temperature, humidity, and light levels, in order to automatically maintain these parameters within optimal thresholds;
- the integration of a **Reinforcement Learning model** for the self-regulation of actuators, enabling the system to dynamically adapt based on real-time environmental data and optimize efficiency;
- the development of a **data inference model** to analyze the collected data and estimate environmental conditions even in areas not directly covered by sensors, thus improving overall system coverage and efficiency;
- the integration of an **AI-based system** for the optimal positioning of sensors within exhibition spaces, considering architectural features and the type of artworks, thereby enhancing data collection and increasing system reliability.

Finally, to provide a clear overview and encourage potential future extensions, the source code and documentation are publicly available on the GitHub platform.

## Table of figures

<i>Figure 1 - Project Workflow</i>	5
<i>Figure 2 - Hardware flow</i>	5
<i>Figure 3 - Edge flow</i>	5
<i>Figure 4 - Data flow</i>	6
<i>Figure 5 - Link InfluxDB-Grafana</i>	6
<i>Figure 6 - Link Firebase-Android App</i>	6
<i>Figure 7 - Hardware organization</i>	7
<i>Figure 8 - Information from sensors</i>	8
<i>Figure 9 - Sequence Diagram</i>	8
<i>Figure 10 - Topic on Python script</i>	9
<i>Figure 11 - Topic in node MQTT-IN</i>	9
<i>Figure 12 - Node-RED flow</i>	10
<i>Figure 13 - InfluxDB Dashboard</i>	10
<i>Figure 14 - Grafana Dashboard</i>	11
<i>Figure 15 - Firebase Dashboard</i>	11
<i>Figure 16 - Initial Dashboard of the App</i>	12
<i>Figure 17 - Main Dashboard of the App</i>	12