UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA (DIMES)

Dipartimento di Eccellenza 2018-2022

Low level and Embedded System programming project - Computer Engineering for IoT 2022/2023

Smart security detector for smart lab: a proposal for an Embedded system to safe monitor chemical laboratory.

Luigi Rachiele[†],

This report presents the design and implementation of a smart smoke sensor system focused on embedded system programming and low-level hardware components. The project utilizes the ATmega328P microcontroller and Arduino Uno board as the development platform. The southbound architecture consists of four main parts: FreeRTOS for system management, sensing components including TMP36 and MQ135 sensors, actuation elements such as a fan and a buzzer, and user interaction facilitated by an LCD screen. The system incorporates an ESP8266 device for cloud connectivity, enabling data exchange with the cloud platform. The report highlights the importance of secure communication protocols and data privacy in the context of IoT and emphasizes the implementation of robust security measures, including authentication and TLS encryption. The northbound architecture is briefly introduced, highlighting the communication between the embedded system and the cloud platform for data storage, analysis, and remote monitoring. The report concludes with insights into the challenges faced during the development process and suggests future improvements and expansions for the system's functionality.

1 Introduction

The following report provides an overview and analysis of the **Low Level and Embedded** System programming aspects implemented in a project focused on developing a **smart lab sensor**. The primary objective of this project is to create an advanced embedded system capable of detecting the presence of toxic gases and fire incidents in laboratory. The embedded system is designed to enable prompt actions in response to safety alerts, such as activating a fan for air circulation and sounding a buzzer to alert users.

The project emphasizes the utilization of low-level programming techniques and embedded systems to enhance safety and security measures. By leveraging advanced sensing capabilities, the system can accurately identify the presence of toxic gases and fire incidents, providing timely warnings to mitigate potential risks.

The embedded system's architecture and programming techniques are designed to ensure efficient data acquisition, processing, and transmission. Through the use of low-level programming, precise control over the hardware components is achieved, optimizing system resources and enhancing overall performance and reliability.

Throughout this report, the implementation details of the embedded system are explored, emphasizing the utilization of low-level programming techniques tailored to the specific requirements of the project. The report also highlights the significance of integrating various sensors, such as temperature sensors, to enhance the system's functionality and improve safety measures.

Additionally, the report examines the implementation of actuation mechanisms, including fan activation for air circulation and buzzer activation for alerting users during safety alerts. The utilization of low-level programming ensures efficient control and seamless integration of these actuation mechanisms within the embedded system.

By analyzing the role of low-level programming and embedded systems in the project, this report aims to provide a comprehensive understanding of their importance in developing smart smoke sensors capable of detecting toxic gases and fire incidents. The insights and recommendations derived from this analysis contribute to the broader field of Low Level and Embedded System programming, offering valuable guidance for future research and development in safety-enhancing embedded systems.

2 Project scope

The project has been designed to address the safety needs of a chemical laboratory where chemical gas releases and fires may occur. Its main purpose is to ensure a safe working environment for the laboratory personnel, prevent potential incidents, and minimize the negative effects resulting from such situations.

The key features of the project include timely detection of chemical gas releases and fires, immediate alerting of the personnel, and implementation of mitigation measures to minimize the harmful effects. The specific objectives of the project are outlined below:

1. Detection of chemical gas releases: The implemented system allows for the real-time identification and monitoring of any leaks or releases of chemical gases within the laboratory.

[†] University of Calabria

Using specially designed sensors, the system can detect the presence of hazardous substances in the air and generate an immediate alert to the laboratory personnel.

2. Fire detection: The project also encompasses early fire detection. Smoke, heat, and flame sensors are strategically placed in the laboratory to promptly detect signals of potential fires. These sensors trigger an alarm and send an immediate notification to the personnel so they can take necessary evacuation and extinguishing actions.

3. Alerting and notifications: When a chemical gas release or fire is detected, the system activates an audible and visual alarm within the laboratory to immediately alert the personnel present. Simultaneously, automatic notifications are sent to the safety officers and designated personnel through mobile devices or an internal communication system.

4. Mitigation measures: In addition to timely alerts, the project includes the implementation of mitigation measures to minimize the negative effects of chemical gas releases and fires. These measures may involve activating ventilation systems, automatically closing gas valves, isolating specific areas, and deploying specialized teams to handle the emergency.

By implementing this project, the chemical laboratory will significantly enhance the safety of its operations. The timely detection of chemical gas releases and fires, coupled with appropriate alerting and mitigation measures, will help protect the personnel, prevent material damages, and preserve the surrounding environment.

The project aims to provide the laboratory with a solid safety foundation, enabling the personnel to carry out their activities in a controlled and protected environment, in accordance with the best industrial safety practices.

2.1 Architecture

The architecture comprises three essential components, as mentioned befor: the **Southbound**, **Cloud**, and **Northbound**.

The **Southbound** component encompasses the physical device responsible for sensing environmental data, such as temperature and air quality, as well as managing alarms triggered by threshold exceedances. It also includes actuating mechanisms, including a buzzer for alerting users to potential dangers and a fan for enhancing air circulation.

The **Cloud** component focuses on the cloud infrastructure, incorporating an MQTT broker (Mosquitto) for reliable device communication, a database (MySQL) for secure data storage, and Node-RED for efficient data flow management.

Lastly, the **Northbound** component consists of user-facing applications, namely the Android App and Node-RED Dashboard, providing intuitive interfaces for data visualization and interaction. By integrating these three components, the architecture ensures a secure, efficient, and user-friendly IoT system capable of safeguarding data privacy, detecting anomalies, and facilitating informed decision-making

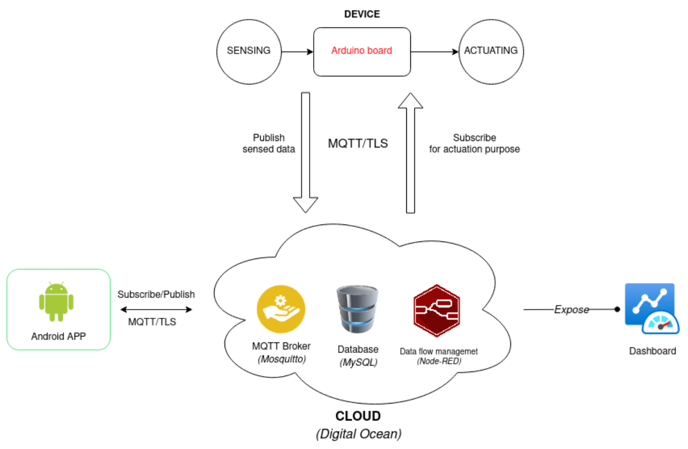


Fig. 1 System architecture

The overall system architecture is illustrated in the 1. This image provides a visual representation of how the different components interact and highlights the protocol used within the system.

2.2 Southbound

This section provides an overview of the part of the project where data are generated. The generation of the data that flow in the architecture are sensed from the external and are in the form of Temperature (ř) and Quality of Air. The data are sensed using the physical device discussed below.

2.2.1 Southbound architecture

The Southbound architecture is a crucial component of the embedded system and it is based on the utilization of the ATmega328P microcontroller. It consist of four main parts. It leverages various technologies and modules to enable seamless operation and interaction within the system. It is shown in the figure 2.

FreeRTOS: At the core of the Southbound architecture lies FreeRTOS, a real-time operating system specifically designed for microcontrollers like the ATmega328P. FreeRTOS provides advanced task management, scheduling, and resource handling capabilities, enabling efficient multitasking and optimal utilization of system resources.

Sensing: The sensing part forms an integral component of the Southbound architecture, enabling data acquisition from the environment. It incorporates two sensors, namely TMP36 and MQ135. The TMP36 sensor measures temperature, while the MQ135 sensor detects the presence of toxic gases. These sensors interface directly with the ATmega328P, allowing the system to monitor and respond to environmental changes effectively.

Actuation: The actuation part of the Southbound architecture empowers the embedded system to execute appropriate actions based on the collected sensor data. It consists of two actuators, a fan and a buzzer, which are controlled by the ATmega328P. In response to detected hazardous conditions, the system can activate the fan to circulate air and employ the buzzer to provide audible alarms, ensuring prompt and necessary interventions.

User Interaction: User interaction is facilitated through an

LCD (Liquid Crystal Display) and the integration of an ESP8266 device. The LCD provides a visual interface for displaying relevant information, while the ESP8266 allows the Arduino board to exchange data with the cloud. This capability opens up possibilities for remote monitoring, data logging, and control of the embedded system.

The Southbound architecture brings together these interconnected components, enabling a comprehensive and efficient embedded system. It leverages FreeRTOS for task management, utilizes sensors for data acquisition, employs actuators for response mechanisms, and incorporates user interaction elements for enhanced control and monitoring. Together, these components form a cohesive architecture that ensures reliable and intelligent operation of the embedded system.

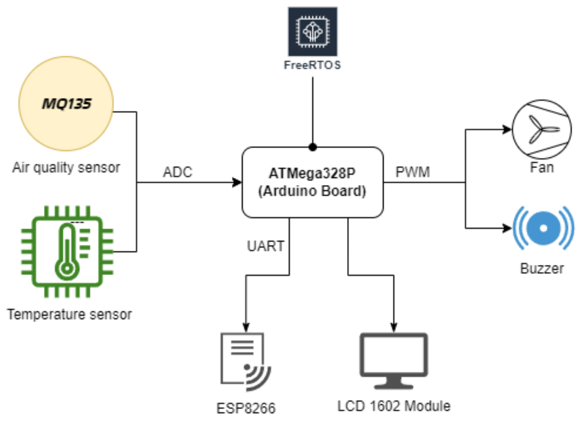


Fig. 2 Southbound architecture

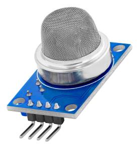
2.2.2 Arduino UNO - Sensing

For the sensing phase, two distinct analogic sensors are utilized: the MQ135 and the TMP36.

The MQ135 sensor is employed to monitor the quality of the surrounding air, enabling the system to detect any potential air pollution or harmful gases. On the other hand, the TMP36 sensor is responsible for measuring the ambient temperature, providing valuable data for temperature monitoring purposes. These sensors are directly connected to the Arduino Uno, leveraging its analog-to-digital converter (ADC) capabilities to convert the analog sensor readings into digital values that can be processed by the microprocessor.

By leveraging the Arduino Uno's versatile capabilities and integrating the MQ135 and TMP36 sensors, the system can accurately sense and monitor both the air quality and temperature. This information serves as crucial input for subsequent stages of the project, such as generating alerts in case of threshold exceedance or initiating the actuating phase for appropriate actions. The Arduino Uno acts as a reliable and robust foundation, enabling seamless integration between the physical sensing components and the overall IoT system architecture.

(a) TMP36



(b) MQ135

Fig. 3 Sensors used.

The chosen approach involves manipulating the internal registers of the ATmega328P microprocessor to ensure a robust, secure, optimized, and direct code implementation. This method provides precise control over the microprocessor's functionalities, allowing for customization according to the project's specific requirements.

Listing 1 Setup temperature reading.

```
void adc_setup_temp(void){
  DDRC&=~(1<<0); //DDRC pin 0 as input for ADC
  ADCSRA = 0x87; //make ADC enagle and select ck/128
  ADMUX = 0b11000000; //Vref Internal

}
```

Listing 2 Setup air reading.

```
void adc_setup_air(void){
  DDRC&=~(1<<1); //DDRC pin 1 as input for ADC
  ADCSRA = 0x87; //make ADC enagle and select ck/128
  ADMUX = 0b01000001; //Vref AVCC pin

}
```

In this code snippet, has been used direct the register manipulation to configure the ADC registers for analog-to-digital conversion. By setting the appropriate bits in the ADMUX and ADCSRA registers, they select the ADC input and enable it with a specific prescaler value.

After the configuration, an ADC conversion is initiated by setting the ADSC bit, and the code waits for the conversion to complete. Once the conversion is finished, the raw ADC value is retrieved converted into voltage using the known voltage reference.

This approach of manipulating the microprocessor's internal registers provides precise control over its functionalities, resulting in an optimized and tailored code implementation for the project.

2.2.2.1 FreeRTOS integration The integration of the RTOS task is done by using a sofwtare interrupt. In FreeRTOS, software timers provide a convenient way to schedule and execute tasks at specified intervals or delays. These timers are entirely managed by the software, allowing precise control over timing without relying on external hardware timers. By utilizing the software timer functionality, the Sensing task can be done in a periodic way. The flexibility of software timers enables efficient resource utilization and task synchronization, enhancing the overall responsiveness and functionality of the system.

It has to be noticed that the ADC is used both from Temperature sensor and Air quality sensor. Therefore it must access and modify the parameter of the ADC in mutex way. At this scope it is used a **FreeRTOS Semaphore**. In FreeRTOS, semaphores are synchronization primitives used to control access to shared resources and coordinate tasks in a multitasking environment. Semaphores act as counters that keep track of the availability of resources, allowing tasks to wait or proceed based on their synchronization requirements. With semaphores, developers can implement mutual exclusion and synchronization mechanisms, preventing race conditions and ensuring orderly access to critical sections of code. FreeRTOS semaphores provide a simple yet powerful means of inter-task communication and synchronization, enhancing the efficiency and reliability of concurrent systems.

Listing 3 FreeRTOS Sofwtware timer.

```
xTimerReadings = xTimerCreate (        , pdMS_TO_TICKS(
    1000 ), pdTRUE, ( void * ) 0, vTimerCallback);
```

After the timer expiration a callback function is called. The callback function resume the sensing task, that is *auto-suspended* at the creation.

Listing 4 Callback timer function.

```
void vTimerCallback(xTimerReadings )
 {
 vTaskResume (temperatureReadTaskHandle);
 vTaskResume (AirReadTaskHandle);
 }
```

2.2.3 Arduino UNO - Actuation

The implementation of actuation in your project involves two separate components: a fan for air circulation and a buzzer for alarm signaling. Both of these components operate by utilizing Pulse Width Modulation (PWM) techniques.

The **buzzer**'s PWM modulation is achieved through the utilization of TIMER of the ATmega328p known as Timer/Counter 0 (Timer0). Timer0 is commonly used to generate Pulse Width Modulation (PWM) signals. To utilize Timer0 for PWM, is has been needed to configure its associated registers, including TCCR0A (Timer/Counter 0 Control Register A) and OCR0A (Output Compare Register A). By setting the appropriate values in these registers, it can be ensured the control the duty cycle of the PWM signal, thereby adjusting the average power delivered to the buzzer. PWM signals generated by Timer0 allow for precise control over the output intensity, enabling smooth and efficient operation of various embedded system applications.

The **fan** operates based on the principles of DC motors, and to enable its functionality, additional support components are needed. Firstly, a small DC motor typically requires more power than what the UNO R3 board can directly provide. Attempting to connect the motor directly to a pin of the UNO board would pose a high risk of damaging the board. To address this, a separate power supply is used to ensure the motor receives the appropriate power.

In addition to the power supply, a chip like the L293D is utilized to control the motor's operation. The L293D provides the necessary interface between the microcontroller and the motor, allowing for precise control and protection against potential electrical issues.

Finally, a relay is used to allow the DC motor to rotate in both directions. A relay is a device that enables or interrupts the passage of electric current between two circuits that are intended to be connected. In our case, when it is desired to change the direction of rotation of the motor, the relay is activated which inverts the polarity of the motor's power supply.

By combining the appropriate power supply management, motor control chip, and relay, the actuation system in your project ensures efficient and reliable operation of both the fan and the buzzer, serving their respective purposes in the overall functionality of the system.

2.2.3.1 FreeRTOS integration There are two ways of activation of the actuation part.

The first one is on the present of *alarm state*. The alarm state is registered whenever a sensed date is above the security threshold. For the temperature sensor the heat threshold is setted to 57 řC, same as classic Alarm Grid (famous anti-fire sensor agency) sensor.

For the Air quality sensor it the threshold is generated by empiric experiment, given the hard managing of the value given by the sensor. The threshold is setted to *200*.

The second way of actuation comes from the user-interaction. See 2.2.6.

In both way the operation are the same. The system callback the function for the interested actuation.

Listing 5 UART initialization.

```
void alarmON() {

  UART_sendString(
                 );
  alarmState = 1;
  stateBuzzer = 1;
  stateFan = 1;
  PORTD |= 1<<PD2;
  buzzerTask();
  fanTask();
}
```

The list of state that the system can ensure are listed in the 2.2.4 section.

2.2.4 Arduino UNO - States and Threshold

The state the system can ensure are different. The table list every Actuation state.

#	Name	FreeRTOS Task	What does?
0	**Alarm ON**	interruptTask	Start alarm. → Start Buzzer and Fan
1	**Alarm OFF**	interruptTask	Stop alarm. → Stop Buzzer and Fan
2	**Buzzer ON**	buzzerTask	Start Buzzer
3	**Buzzer OFF**	buzzerTask	Stop Buzzer
4	**Fan ON**	fanTask	Start Fan
5	**Fan OFF**	fanTask	Stop Fan

Those states are ensured by performing the two way actuation: given by sensed date and given by the user.

After the sense and the threshold check the data are put in a Queue waiting to be sent.

2.2.5 Arduino UNO - Sending data

After sensing the data and checking the threshold the data are sent to the user and to the cloud.

They are sent to the user by two way: LCD present on the board and by Dashboardes present on the Cloud (check next chapter).

2.2.5.1 FreeRTOS interaction This action is performed by using a task. The task remain in listening on a FreeRTOS Queue, waiting for the the sensing phase to sense data. After the data join the queue the task manage the data and send:

- **through serial** for the cloud; check 2.2.7;
- **through LCD** for real-time user interaction.

2.2.6 Arduino UNO - User to Actuation Interaction

Through the dashboard interface, users have the ability to issue various commands to the microprocessor, enabling them to perform specific actions. These commands include turning the alarm on and off, activating or deactivating the buzzer, and controlling the fan by turning it on or off.

To facilitate communication between the user and the microprocessor, the MQTT protocol is employed. When a command is sent by the user through the northbound dashboard, it is transmitted via MQTT. The ESP8266, which acts as the MQTT listener, receives the command and forwards it to the Arduino board using the serial port for further processing.

Upon receiving the command through the serial communication, a system interrupt is triggered. This interrupt suspends the execution of other tasks momentarily, prioritizing the handling of the received command. The command are handled by using another different task, to mantain the interrupt service routine as small as necessary. Consequently, the appropriate task is awakened to process the command and carry out the necessary actions.

It's worth noting that the commands can vary in type, depending on the desired state of the system. Each distinct command serves as a trigger to awaken a specific task within the FreeRTOS framework, ensuring efficient and accurate actuation based on theăuser'săinput.

2.2.6.1 FreeRTOS Interaction Each of these commands directly affects the state of the system and is managed through task scheduling using the FreeRTOS operating system. The system comprises different tasks, namely "fanTask," "alarmOn," "alarmOff," and "buzzerTask," which are responsible for executing the corresponding actions based on the received commands.

2.2.7 Arduino UNO - UART

In order to enable message transmission from the sensors to the cloud, the project utilizes the MQTT protocol. However, a crucial requirement for this implementation is establishing an internet connection. This is achieved by utilizing an ESP8266 module, which comes equipped with built-in Wi-Fi connectivity, to connect to an accessible hotspot. To facilitate data transfer from the Arduino to the ESP8266, the project leverages the UART (Universal Asynchronous Receiver-Transmitter) serial communication. By connecting the RX (receive) and TX (transmit) pins of both the Arduino and the ESP8266, a seamless data flow between the two devices is established.

First of all the UART is initialized:

Listing 6 UART initialization.

```
void UART_init()
{
    // Setup Trasmission rate
    UBRR0H = (BAUD_PRESCALER >> 8);
    UBRR0L = BAUD_PRESCALER;

    // Enable both in and out trasmission
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);

    // Setup data format: 8 data bit, 1 stop
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}
```

Then 2 different function permits the data trasmission:

Listing 7 Send a string on serial.

```
void UART_sendString(const char* data)
{
    // Send each character until the null terminator
    while (*data !=     )
    {
        UART_sendChar(*data);
        data++;
    }
}
```

Listing 8 Send a char on serial.

```
void UART_sendChar(char data)
{
    // Wait until the trasmission buffer is empty
    while (!(UCSR0A & (1 << UDRE0)));

    // Upload the data in the trasmission buffer
    UDR0 = data;
}
```

UART_sendChar wait until the trasmission buffer is empty and then upload the data in the USART Data Register 0. When the data are written on the UDR, it will be trasnferred by the UART peripheral through the serial transimt pin.

2.2.8 ESP8266

The ESP8266 receives from the UART the data from the Arduino UNO and sends it using the Trasport Layer Security (TLS). The connection to the cloud is established using the MQTT (Message Queuing Telemetry Transport) protocol enriched with the TLS (Transport Layer Security) security protocol. This combination ensures a secure and protected connection between the ESP8266 device and the cloud. By using the generated certificates and stored in the device flash memory, the ESP8266 device can establish a secure connection with the cloud. These certificates allow for the verification of the device and server identities in the TLS handshake process, thereby preventing potential man-in-the-middle attacks.

The ESP8266 code is composed by 3 parts: - *ConnectionManager*: this module manage the certs file acquiring the MQTT connection using TLS. - *Service*: the goal of this part is to actually create the MQTT Client; - *main*: this is the main part where the flow of data goes through.

The **main.cpp** is the module that describe the *main* part of the sketch.

It takes all the *secrets* information, that are loaded in the **Flash Memory** of the ESP8266 and build the MQTT client.

Once the connection is established, the ESP8266 has one and only task: forward the serial message received by the Arduino board to the MQTT Broker. To do that, the ESP8266 start the serial monitor, scan for messages and when it receives one forward that publishing a message in the 'D001' topic.

3 Cloud

The cloud section focuses on the cloud infrastructure and services employed in the IoT project. It includes various components critical to the IoT project, including the MQTT broker, database, and the utilization of Node-RED for data flow management.

The cloud is managed using a Digital Ocean droplet that serves as the central point of the project.

3.1 MQTT Broker - Mosquitto

The MQTT broker plays a pivotal role in facilitating communication between the IoT devices and the cloud services. It acts as a central hub, receiving messages from the devices and distributing them to the relevant cloud services. The MQTT broker's implementation includes robust security measures to ensure the confidentiality, integrity, and availability of the transmitted data. This involves the use of secure authentication protocols, access controls, and encryption techniques to protect against unauthorized access and data breaches.

The chosen MQTT broker for the project is Mosquitto. The communication between the different subscribers is secured using TLS (Transport Layer Security). Additionally, username and password authentication is implemented to ensure authorized access to the MQTT broker.

To manage the TLS protocol, Certbot is utilized, and the project obtains certificates through "Let's Encrypt". The obtained certificates are associated with the domain "iotprojectunical.me", which is acquired from the namecheap website.

By leveraging TLS, username and password authentication, and obtaining certificates from Let's Encrypt, the project establishes a secure and encrypted communication channel between the MQTT broker and the subscribers. This setup ensures the confidentiality, integrity, and authenticity of the transmitted data within the IoT project.

Three foundamental user are created for authentication of Mosquitto Broker:

- **mqttesp**: used by the esp to enstablish an MQTT connection with the broker;
- **mqttnodered**: used by the Node-RED flow;
- **mqttandroid**: used by the Android App;

3.1.1 Let's Encrypt certificate

To acquire the SSL certificates to enable the TLS MQTT connection is used a nonprofit Cerficate Authority called *Let's Encrypt*. For accomplish the certificate is used a tool called *Certbot*.

The SSL cerficate need a real existing domain and the one used in this project is acquired from the *Namecheap* web hosting agency.

3.1.2 Mosquitto configuration

To enable the Mosquitto TLS communication it is needed to create new configuration file in the mosquitto *conf.d*.

The file contain the following information:

Listing 9 Mosquitto config file.

```
allow_anonymous false
password_file /etc/mosquitto/passwd
listener 1883 localhost
listener 8883
certfile /etc/letsencrypt/live/iotprojectunical.me/cert
    .pem
cafile /etc/letsencrypt/live/iotprojectunical.me/chain.
    pem
keyfile /etc/letsencrypt/live/iotprojectunical.me/
    privkey.pem
```

First of all disallow the anonymous publish and subscribe. Then enable the password file where the password are saved in a sha256 encryption. Setup then the cerficate obtained by the Certbot on the namecheap domain: *iotprojectunical.me*

3.2 Database

In the database section, we discuss the database management system utilized in the IoT project.

The database utilized in the project is MySQL. However, for the encryption of the database, a deliberate decision was made not to employ Transparent Data Encryption (TDE). Instead, a manual approach to data encryption was deemed more suitable for educational purposes. Consequently, the data is manually encrypted using the DES (Data Encryption Standard) algorithm before being inserted into the database.

By employing manual encryption, the project gains greater control over the encryption process and can apply specific encryption techniques tailored to its requirements. This approach allows for the secure storage of sensitive data within the database.

When retrieving the data from the database, it can be decrypted, enabling various functions and operations to be performed on the decrypted data. This provides the necessary flexibility for data analysis, manipulation, and processing while maintaining the security of the sensitive information.

The decision to opt for manual data encryption with DES rather then of TDE demonstrates a purposeful choice made within the project for educational purposes. It offers insights into the manual encryption process and provides an opportunity to gain a deeper understanding of data encryption techniques and their implementation within a secure environment.

The table where the data are stored are created by using the following code:

Listing 10 Mosquitto config file.

```
CREATE TABLE readings (
  device varchar(4) NOT NULL,
  datetime TIMESTAMP NOT NULL,
  sensor varchar (20) NOT NULL,
  value varchar(44) NOT NULL,
  PRIMARY KEY (device, datetime, sensor)
) ENGINE = InnoDB
```

So a *redings* table is created to store both temperature and air quality readings. It has a different column describing the various information of the stored data:

- **device** value to show from wich device the reading comes; in this project is used just one device;
- **Timestamp** value;
- **sensor** value that store if it is a Temperature or Air Quality reading;
- **value** part that store the encrypted form of the Float calculated by the Arduino sensors.

The Primary Key is setup by a triple of device, datetime and sensor.

3.3 Node-RED

Regarding Node-RED, it is also hosted in the cloud. It is protected by username and password authentication, and access to it is secured using the HTTPS protocol through the implementation of the Nginx web server.

Node-RED plays a crucial role in managing the flow of data within the project. Data is received by Node-RED through MQTT, ensuring secure communication with TLS encryption. From there, the data is directed towards two main destinations. Firstly, it flows towards the dashboard, which provides a visual representation of the data for monitoring and analysis. Secondly, the data is routed towards the encryption phase, where it is encrypted before being securely stored in the database.

By employing username and password authentication and implementing HTTPS through Nginx, Node-RED ensures secure access and communication. The integration with MQTT, along with TLS encryption, guarantees the confidentiality and integrity of the data as it traverses through the system. The combined functionalities of Node-RED, MQTT with TLS, and the secure storage of data in the database contribute to the overall security of the IoT project.

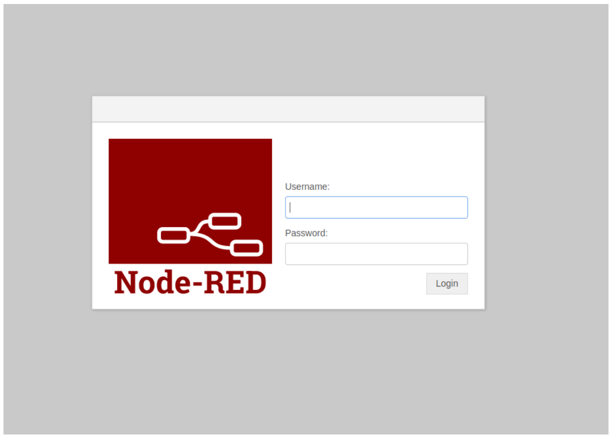


Fig. 4 NodeRED authentication.

The Node-RED app is protected by an authentication system shown in figure 4.

First of all is inject in the system the AES key created in the Cloud using the *OpenSSL Tool*, shown in the figure 5.

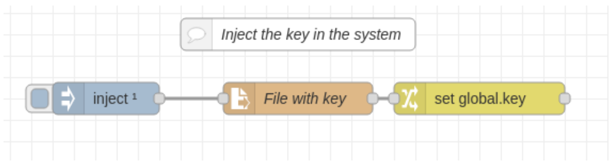


Fig. 5 Node-RED Inject Key.

Then the Node-RED MQTT node subscribe to all the topic of the Mosquitto Broker, using the wildcard "". Data that flow to the Mosquitto broker flow into 2 different subflow, one for the Dashboard and the other one for the ***Encription and Store*** phase.

The third phase of the Node-RED flow start from the user input on the Dashboard and follow 2 flow. One to setup some command on the Arduino Board and one for the ***Query and Decript*** phase.

3.4 Data Flow

Before going in the deep for the Node-RED different phase let's talk about the system **Data Flow**. The data flow from Arduino to the Database are in JSON format. This is an example of *readings*.

Listing 11 JSON Example.

```
{"sensor":"temperature","value": 26.33}
```

The JSON is composed by a map of keyvalue of size 2. The first key/value indicates the sensor used for the reading (e.g. Temperature, Air); the second key/value couple indicates the value of the reading.

Those data has to flow in Node-RED, has to be encrypted and then stored in the MySQL database.

Listing 12 Encryption modes.

```
The encription could have been done in two different
    way in this system: TDE Method and Manual method.
```

```
The TDE method (Transparent data encryption) executes
    encryption and decryption within the database
    engine itself. This method doesnt require code
    modification of the database or application and is
     easier for admins to manage. However is not the
    best eligible in a didactic enviroment. The manual
     method is the best fit for understanding how
    encryption works and to study also the AES
    encryption method.
```

For this project, it is chosen to encrypt only the value part of the JSON data. The encryption is made by using the AES algorithm given by a Node-RED node.

3.4.1 AES

The data pass through the Node-RED flow and is redirected both into the Dashboard as clear data and in the Encrytpion node. The Encryption is made using the AES algorithm. After the Encryption the data is stored in the MySQL database.

From the Dashboard, an user can ask to the Database some datatime based query (see in the Dashboard paragraph). The query is applied to the database and the encrypted data is given as output. The data is then decrypted and is shown in the Dashboard.

3.5 Node-RED - *Encrypt and Store*

The first Node-RED subflow analyzed is the *Encrypt and Store* shown in figure 6

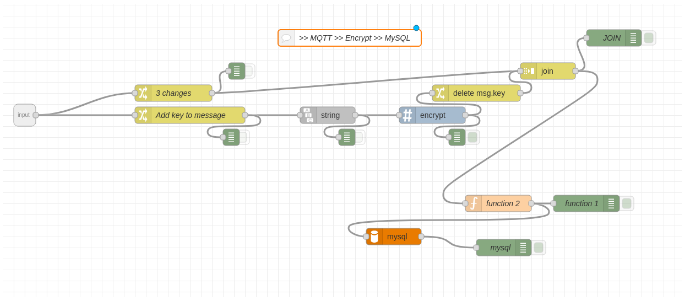


Fig. 6 Node-RED Encrypt and Store module.

The input messages are in the JSON form. The info that the system need from the message in input are:

- *Device*: which device is sending information; it can be retrieved from the MQTT topic;
- *Sensor*: which device sensor is reading; it can be retrieved from the JSON in input;
- *Value*: which value has the reading; it can be retrieved from the JSON in input;

Every message that flow in this subflow has to pass in the Encryption node. The Encryption node take the *msg.key* and the *msg.payload* and makes the AES encryption.

After the encryption the message has to be purged from the key (or it will be shown in clear) and to be sent in the *function node* that build the query for the Database. An example of query is shown belove:

Listing 13 JSON Example.

```
USE sensors;
INSERT INTO Readings(device, datetime, sensor, value)
    VALUES(      , NOW(),      ,      );‘
```

The query has to be set in the *msg.topic* and has to flow in the *MySQL* node.

3.6 Node-RED - *Query and Decrypt*

The second Node-RED sublflow analyzed is the *Query and Decrypt* shown in figure 7.

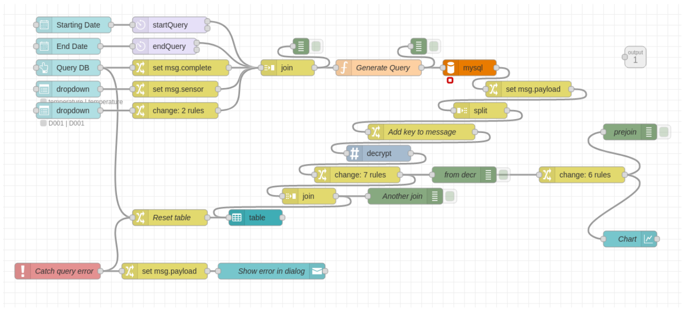


Fig. 7 Node-RED Query and Decrypt module.

The input this time is from the user dashboard, analyzed below.

After the user input the data are converted in good format to be joined in a Query for the database. This an example query for the database:

Listing 14 JSON Example.

```
USE sensors;
SELECT * FROM Readings WHERE device=
  AND sensor=
  AND datetime>=
  AND datetime<           ;
```

After the DB interrogation the output is decrypted and from them are generated two different output: a *table* with all value founded from the query and a *graph* with the curves that the sensing made in the datatime range.

4 Northbound

The Northbound section of this report focuses on the applications that transparently utilize the data generated by the physical devices for visualization and interaction. This section encompasses two key components: the Android App and the Node-RED Dashboard.

4.1 Android App

The Android App serves as a user-friendly interface, allowing users to access and interact with the IoT system. It seamlessly integrates with the data generated by the physical devices, providing real-time updates, control options, and visualizations. The

Android App employs robust security measures, including secure user authentication and encrypted communication, to ensure the confidentiality and integrity of the data exchanged between the app and the IoT system.

The Android App is built using the framework **Flutter**.

Using the library *mqtt-client* written in Dart programming language, is it possible to connect every smarthphone to the Iot Project Unical project.

The app is composed by 4 module:

- **CmdValueHandler.dart**: utils module to manage the command value sent by the android app to the system;
- **SensorValueHandler.dart**: utils module to manage the sensing value sent by the system to the android app;
- **main.dart**: main class that manage the app frontend;
- **MqttHandler.dart**: that manage the MQTT through TLS connection.

4.1.1 MqttHandler.dart

The most important part of the Mqtt handler is the connection one.

Listing 15 JSON Example.

```
client = MqttServerClient.withPort(
        'iotprojectunical.me', 'telefono_di_gixs1',
             8883);
    client.logging(on: true);
    client.onConnected = onConnected;
    client.onDisconnected = onDisconnected;
    client.onUnsubscribed = onUnsubscribed;
    client.onSubscribed = onSubscribed;
    client.onSubscribeFail = onSubscribeFail;
    client.pongCallback = pong;
    client.keepAlivePeriod = 60;
    client.logging(on: true);
    client.secure = true;
```

Using the flag *client.secure* the flutter app knows that have to use the default *Secure Context*.

The other code written is about the handling of topic and JSon message exchanged.

4.1.2 Front End

The app frontend is show in the figure 8

The app shows in two text field the last values of *Air quality* and *Temperature*.

The buttons are disposed to achieve the system actuation. The one on the left close the alarm generated when the values overcome the thresholds.

The one on the right change the vent state. If the vent is on, put it off and viceversa.

That is done by sending through mqtt the command messages.

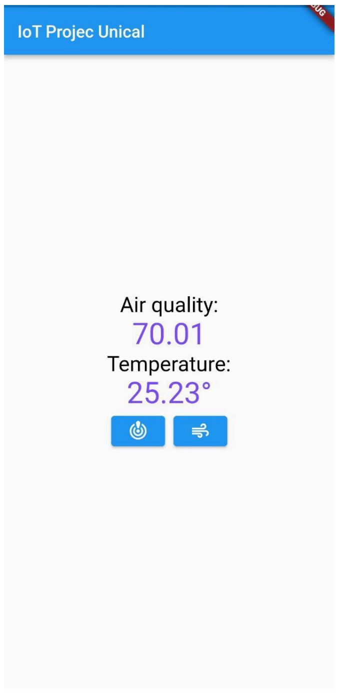


Fig. 8 Android App Front End.

4.2 NodeRED Dashboard

The Node-RED Dashboard complements the Android App by providing an additional platform for data visualization and interaction. It offers a customizable and intuitive interface that allows users to monitor and control the IoT system's data flow in real-time. The Node-RED Dashboard incorporates security features, such as secure access controls and encrypted communication channels, to safeguard the data as it flows between the IoT system and the dashboard.

The Northbound section highlights the significance of the Android App and the Node-RED Dashboard in enabling users to effectively visualize and interact with the data generated by the physical devices. By ensuring the security and usability of these applications, the IoT system becomes more accessible and user-friendly, empowering users to make informed decisions based on the insights derived from the data.

4.2.1 Back-end

The Dashboard back-end is composed of two part.

The one shown in the Query and Decrypt]7, for the output from the database. The second one is shown in the figure 9 and it is a realtime dashboard, showing the last value received from the sensor.

Data already formatted flow into a Non-editable Text Field and is show realtime. Also a chart of the last hour readings is made realtime.

4.2.2 Front End

To navigate toward the two part of the of the dashboard is build a *side navigation menu*.

Here the user can switch from the realtime dashboard and the dashboard showing the output query from database.

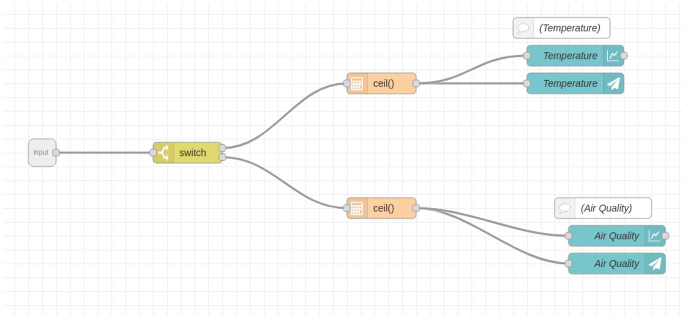


Fig. 9 Node-RED dashboard.

The **realtime dashboard** show the last value received from the sensors and give also a graph of the last values received, shown in the figure 10

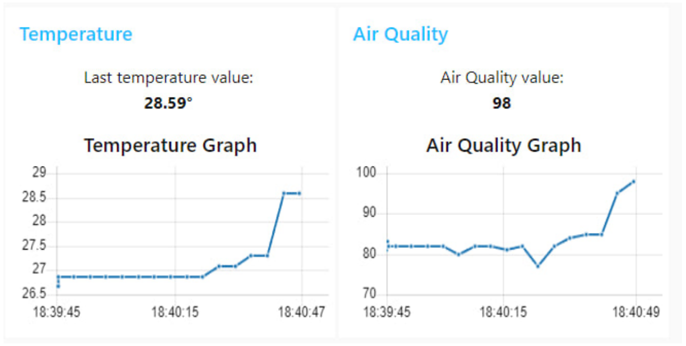
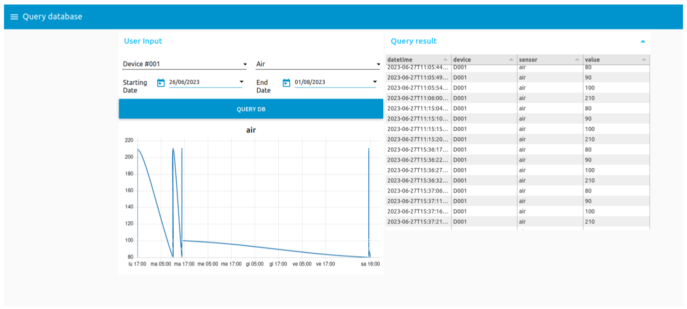


Fig. 10 Node-RED realtime dashboard.

The **query database** dashboard part is divided in two part: *user input* and *output*, shown in figure 11.



datetime	device	sensor	value
2023-06-27T11:05:44...	D001	air	80
2023-06-27T11:05:49...	D001	air	90
2023-06-27T11:05:54...	D001	air	100
2023-06-27T11:06:00...	D001	air	210
2023-06-27T11:15:04...	D001	air	80
2023-06-27T11:15:10...	D001	air	90
2023-06-27T11:15:15...	D001	air	100
2023-06-27T11:15:20...	D001	air	210
2023-06-27T15:36:17...	D001	air	80
2023-06-27T15:36:22...	D001	air	90
2023-06-27T15:36:27...	D001	air	100
2023-06-27T15:36:32...	D001	air	210
2023-06-27T15:37:06...	D001	air	80
2023-06-27T15:37:11...	D001	air	90
2023-06-27T15:37:16...	D001	air	100
2023-06-27T15:37:21...	D001	air	210

Fig. 11 Node-RED realtime dashboard from DB.

The *user input* give the possibility to insert information to query the database.

The *output* part give the output of the query as table and as graph.

5 Conclusion

In conclusion, this project has demonstrated the successful implementation of an embedded system, with a specific focus on the southbound architecture. By utilizing the ATmega328P microcontroller and FreeRTOS, we have developed a robust and efficient system capable of smoke detection, fire prevention, and user interaction.

The southbound architecture, which encompasses the main.cpp file, serves as the backbone of our system. It comprises several key components, including FreeRTOS for system management, sensing modules such as the TMP36 and MQ135 sensors, actuation components such as the fan and buzzer, and user interaction elements like the LCD display. The integration of these components allows for seamless data acquisition, processing, and response within the embedded system.

Furthermore, the utilization of the ATmega328P microcontroller as the core hardware platform has proven to be highly effective. Its low-power consumption, ample I/O capabilities, and compatibility with the Arduino Uno board provide a solid foundation for the development of embedded systems. The integration of hardware timers, such as Timer0 for PWM control, further enhances the system's functionality and performance.

Additionally, the project's integration with the cloud, facilitated by the ESP8266 device, enables data exchange and remote monitoring. The northbound aspect of the system establishes a connection between the embedded system and the cloud services, allowing for real-time data transmission, storage, and analysis. This cloud integration enhances the system's scalability, accessibility, and enables advanced monitoring and control capabilities.

In conclusion, the development of the southbound architecture, coupled with the integration of cloud services in the northbound, has resulted in a powerful and versatile embedded system. The utilization of the ATmega328P microcontroller, along with FreeRTOS, has provided a solid foundation for low-level and embedded system programming. The successful implementation of the system's components, including sensing, actuation, and user interaction, demonstrates the effectiveness of the chosen architecture and highlights the potential for real-world applications in various domains.