Innovative Telecommunication System SmartBarracks project

Gaudiano Antonio Mat.744102
Muggiasca Luca Mat.744565

June 18, 2024

Contents

List of Figures

1

Introduction

1.1 Scenario

In a military barracks scenario, the integration of sensors and IoT technologies can enhance security, improve infrastructure management, and enhance the well-being of military personnel.
In the considered scenario some of the rooms in the building will be monitored through multiple kind of digital and analogical sensors in order to make the management easier.

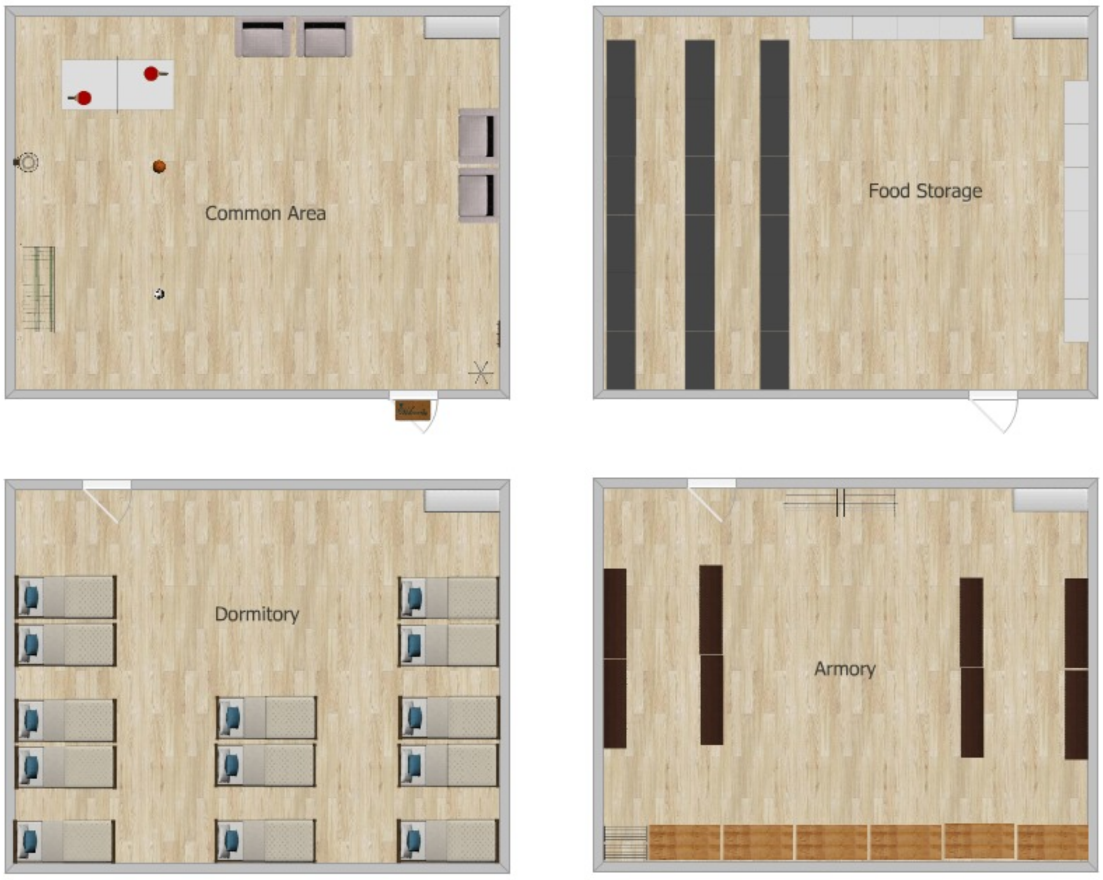


Figure 1.1: Barracks structure

1.1.1 Sensor and Devices Involved

- Common Areas (Recreation Room, Lounge):
  1. PIR (Movement sensor) to control lighting automatically
  2. Temperature sensor to control temperature level automatically through air conditioning and thermostat
  3. CO2 sensors to monitor air quality and ensure adequate ventilation during the night
  4. Fire Alarm device
  5. T-HVAC for fixing temperature and co2 values.
- Food Storage:
  1. PIR (Movement sensor) to control lighting automatically
  2. Temperature sensor to control temperature level automatically through air conditioning and thermostat
  3. Humidity sensor to check the humidity level to prevent and ensure food safety
  4. CO2 sensors to monitor air quality and ensure adequate ventilation during the night
  5. Fire Alarm device
  6. T-HVAC for fixing temperature and co2 values.
  7. H-HVAC for fixing the humidity value.
- Dormitory:
  1. Temperature sensor to ensure a comfortable environment for rest
  2. CO2 sensors to monitor air quality and ensure adequate ventilation during the night
  3. Fire Alarm
  4. T-HVAC for fixing temperature and co2 values.
- Armory Room:
  1. PIR (Movement sensor) to control lighting automatically
  2. Temperature sensor to monitor and ensure optimal conditions for storing weapons and ammunition
  3. Humidity sensor to prevent weapon corrosion
  4. CO2 sensors to monitor air quality and ensure adequate ventilation during the night
  5. Biometric sensor to control the identity of people entering the room
  6. A security system that triggers an alarm in case of unauthorized access
  7. Fire Alarm device
  8. T-HVAC for fixing temperature and co2 values.
  9. H-HVAC for fixing the humidity value.

1.1.2 High-level rules

- Lights:
  - the lights will be turned on or off based on the movement sensors,
  - in the dormitory the light will always be switched off from 22 p.m. to 5 a.m.
- Temperature:
  - the temperature must be in a given range, from 19°C to 21°C otherwise the HVAC systems (air conditioning / thermostat) will be activated.
- CO2:
  - the CO2 level must be in a given range, from 200 to 1000ppm otherwise the HVAC systems (air conditioning / thermostat) will be activated
- Humidity (Relative humidity (RH) is measured using a hygrometer, a device specifically designed to measure the moisture content in the air. The commonly used unit of measurement for relative humidity is percent (%), indicating the amount of water vapor present in the air relative to the maximum amount of water vapor the air can hold at a given temperature):
  - in order to ensure food quality and safety the humidity level must be in a given rage, from 50% to 70%, otherwise HVAC systems will be activated. It helps to:
    1. **Preventing moisture loss**: RH helps prevent moisture loss from foods, which is important for maintaining their freshness, texture, and taste. Foods with inadequate moisture levels can become dry, tough, and unpalatable.
    2. **Preventing moisture gain**: While maintaining moisture is important, excessive moisture can promote microbial growth, spoilage, and mold development. RH levels above 70% can create conditions conducive to mold growth and spoilage, particularly in perishable foods.
    3. **Controlling bacterial growth**: RH levels within the ideal range can help control bacterial growth and proliferation. Bacteria thrive in environments with high moisture content, so keeping RH within the recommended range helps minimize the risk of bacterial contamination and foodborne illnesses.
    4. **Preventing condensation**: RH levels that are too high can lead to condensation forming on food surfaces or within packaging, which can promote bacterial growth and compromise food safety. Maintaining RH within the recommended range helps prevent condensation-related issues.
  - in order to ensure the safety of the guns and ammunition the humidity level must be in a given range, from 30% to 50% otherwise HVAC systems will be activated. It helps to:
    1. **Preventing corrosion**: Firearms and ammunition are susceptible to corrosion when exposed to moisture. Maintaining RH levels below 50% helps prevent moisture buildup, reducing the risk of rust and corrosion on metal surfaces.
    2. **Preserving gunpowder**: Excessive moisture can affect the stability and performance of gunpowder in ammunition. Keeping RH levels within the recommended range helps preserve the integrity of gunpowder, ensuring reliable ignition and safe operation of firearms.
    3. **Preventing mold and mildew**: High humidity levels can promote the growth of mold and mildew, which can damage firearms, ammunition, and other equipment stored in the armory. Maintaining RH below 50% helps inhibit mold and mildew growth, preserving the condition of firearms and ammunition.

4. **Minimizing condensation**: RH levels that are too high can lead to condensation forming on metal surfaces, which can accelerate corrosion and compromise the functionality of firearms and ammunition. By keeping RH levels within the optimal range, condensation-related issues can be minimized.

- Alarms:
  - when an unauthorized access is detected in the armory (with motion and biometric sensors) an alarm will trigger
  - other kind of not physical alarm (notification and / or email) will fire in case of anomalies in temperature / humidity / CO2 level and so on
  - a Fire Alarm will be triggered if the temperature has a rapid increase such as exceeding 15-20°C above the normal room temperature (from 21°C to 41°C) & the CO2 concentration in the air exceeds 1200 parts per million (ppm).

Most of the more useful data will also be collected and published through MQTT protocol in order to be easily accessible.

1.1.3 Sensors Monitoring Intervals

The randomization and checking intervals for each sensor are determined by its priority in the application context. For example, the fire alarm is checked every second because immediate detection of fire is critical. Conversely, the humidity sensor is randomized every 12 seconds and checked every 10 seconds because precise humidity data is not essential for short-term actions. Details of the randomization and check intervals for each sensor can be seen in the table 1.1.

Sensor	Randomized every	Checked every
Temperature	10 seconds	5 seconds
CO2	10 seconds	5 seconds
Humidity	12 seconds	10 seconds
PIR	10 seconds	7 seconds
Fire Alarm	-	1 second (continuously)
Biometric	7 seconds	7 seconds

Table 1.1: Monitoring times

2

Developed Application

2.1 Developed Solution

The application includes various features to efficiently manage a possible smart barrack scenario, ensuring seamless integration, real-time monitoring, and enhanced security. These features include:

- A graphical user interface (GUI) that shows the status of the sensors for each room: The GUI provides an intuitive visual representation of the sensor data across different rooms in the barrack. This interface allows users to quickly assess the current status of each room, including temperature, humidity and other relevant parameters. The GUI is designed to be user-friendly, enabling personnel with varying levels of technical expertise to interact with the system effortlessly.
- An online dashboard which includes graphs showing sensor values in real time and over time: This dashboard is a powerful tool for real-time monitoring and historical data analysis. It features dynamic graphs that display sensor readings as they are collected, allowing users to observe trends and identify anomalies promptly.
- A webpage for managing the sensors remotely: The remote management webpage provides users with the ability to configure and control sensors from any location with internet access. Remote management ensures that the system remains flexible and responsive to changes in the operational environment.
- An online dashboard that shows the values taken from MQTT (useful for other possible integrations): The MQTT (Message Queuing Telemetry Transport) dashboard displays real-time data streams from sensors using the MQTT protocol. This lightweight messaging protocol is ideal for connecting devices in a low-bandwidth, high-latency network environment. The dashboard supports integration with other systems and applications, facilitating data sharing and interoperability within a broader IoT (Internet of Things) ecosystem.
- An online page that logs all the armory accesses for security purposes: Security is a paramount concern in a smart barrack scenario, particularly with regard to sensitive areas like the armory. This log is essential for audit trails, ensuring accountability, and detecting unauthorized access attempts. It enhances the overall security posture of the barracks by providing a robust mechanism for monitoring and reviewing access events.

These features collectively contribute to the efficient and secure management of a smart barrack environment, leveraging modern technologies to enhance operational effectiveness and security.

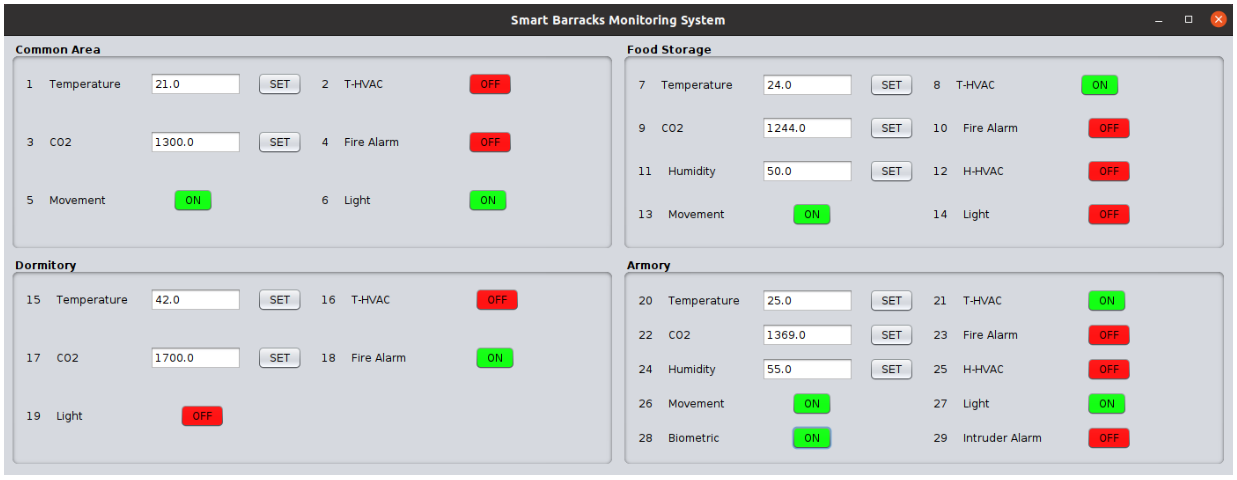
2.1.1 Sensors Interface

There is an interactive interface that displays and allows control of various parameters for all four rooms in the military barrack. The panel is designed to provide a comprehensive view and control of the barrack's environment, integrating both digital and analogical devices to offer a versatile and user-friendly experience.

The digital devices in the interface allow users to check and control the ON/OFF states of several critical systems. These systems include:
T-HVAC / H-HVAC : Users can monitor and switch the HVAC system on or off, ensuring optimal temperature control for comfort / operational efficiency and humidity levels, crucial for maintaining the right environmental conditions to prevent equipment damage and ensure personnel comfort.
Movement Sensors: These sensors detect motion within the rooms.
Lights: Users can control the lighting systems, turning them on or off as needed to ensure proper illumination for activities or to conserve energy.
Biometric Sensor: This sensor monitors access to restricted areas using biometric data (e.g., fingerprints, retinal scans).

The analogical devices in the interface display the current values of certain environmental parameters and also provide functionality to set new values. These include:
Temperature: Users can adjust the temperature settings directly through the interface to maintain a comfortable and suitable environment for personnel and equipment.
CO2 Levels: Real-time CO2 level monitoring ensures air quality is within safe limits. Users can set thresholds or adjust ventilation settings to maintain healthy air quality.
Humidity: Humidity levels are continuously monitored and displayed. Users can adjust humidity settings to ensure optimal conditions, preventing issues such as mold growth or equipment malfunction.

As shown in figure 2.1, the interface provides a comprehensive overview and control of the environment within the military barrack.


Smart Barracks Monitoring System

Common Area

1	Temperature	21.0	SET	2	T-HVAC	OFF
3	CO2	1300.0	SET	4	Fire Alarm	OFF
5	Movement	ON		6	Light	ON

Food Storage

7	Temperature	24.0	SET	8	T-HVAC	ON
9	CO2	1244.0	SET	10	Fire Alarm	OFF
11	Humidity	50.0	SET	12	H-HVAC	OFF
13	Movement	ON		14	Light	OFF

Dormitory

15	Temperature	42.0	SET	16	T-HVAC	OFF
17	CO2	1700.0	SET	18	Fire Alarm	ON
19	Light	OFF				

Armory

20	Temperature	25.0	SET	21	T-HVAC	ON
22	CO2	1369.0	SET	23	Fire Alarm	OFF
24	Humidity	55.0	SET	25	H-HVAC	OFF
26	Movement	ON		27	Light	ON
28	Biometric	ON		29	Intruder Alarm	OFF

Figure 2.1: Sensor's control graphic interface

2.1.2 Sensor's Dashboard

There is an interactive dashboard that displays sensor values in real time and over time, divided by room. This dashboard is accessible at 127.0.0.1:1880/ui in the Home section in the top-left corner menu, providing a centralized platform for monitoring and analyzing environmental data across the military barrack.

The dashboard includes several key features designed to enhance user experience and facilitate efficient data monitoring:

Real-time Sensor Data Display: The dashboard features gauge graphs that show the current values of various sensors in real time. These gauges provide an immediate visual representation of parameters such as temperature, humidity, CO2 levels allowing users to quickly assess the current environmental conditions in each room.
Historical Data Visualization: In addition to real-time data, the dashboard includes graphs that display the change in sensor values over time. This historical data visualization helps users identify trends, patterns, and anomalies. For instance, users can analyze temperature fluctuations throughout the day or monitor humidity levels over a week, enabling better understanding and management of the barrack's environment.
User-friendly Interface: The interactive nature of the dashboard allows users to navigate through different sections easily by he Home section in the top-left corner menu.

Figure 2.2 illustrates the layout and functionality of the dashboard, showcasing the gauge graphs and historical data charts. These visual tools provide a comprehensive overview of the environmental conditions within the barrack, enabling efficient monitoring and active management.

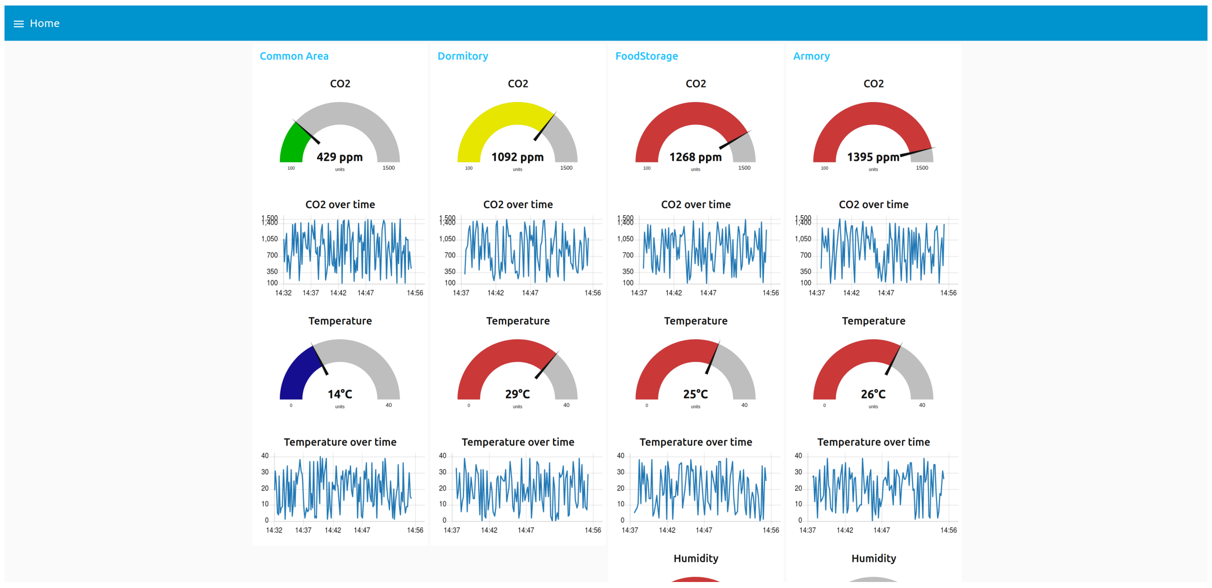
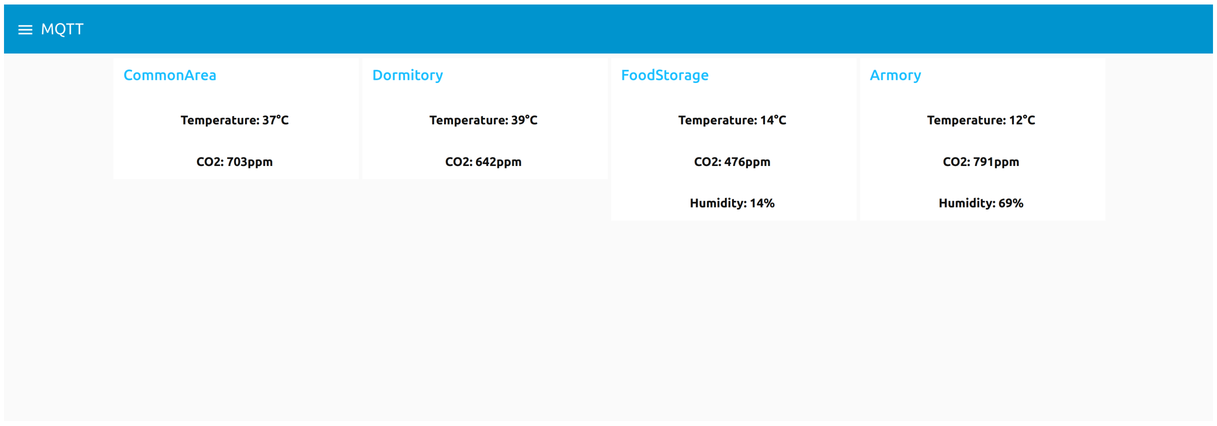


Figure 2.2: Sensor's Dashboard Overview

2.1.3 MQTT Dashboard & Email Notifications

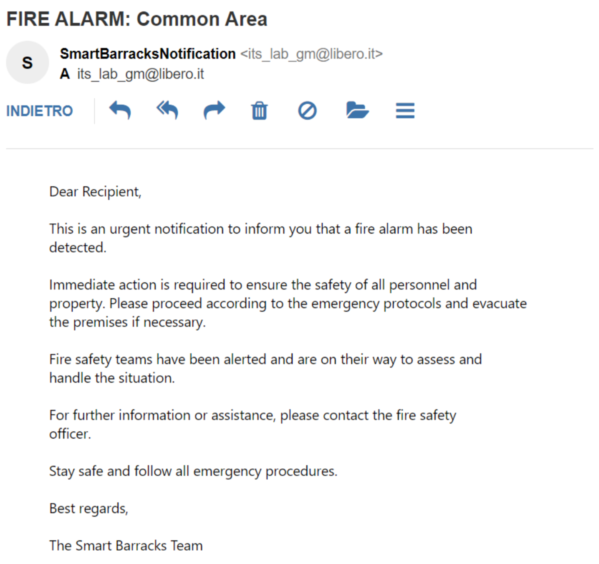
There is also a dashboard for displaying the data published through MQTT.
It can be accessed at 127.0.0.1:1880/ui in the MQTT section in the top-left corner menu.
The dashboard displays the payload of the MQTT messages for each area, published in real time on Mosquitto, as shown in figure 2.3.



CommonArea	Dormitory	FoodStorage	Armory
Temperature: 37°C	Temperature: 39°C	Temperature: 14°C	Temperature: 12°C
CO2: 703ppm	CO2: 642ppm	CO2: 476ppm	CO2: 791ppm
		Humidity: 14%	Humidity: 69%

Figure 2.3: MQTT's Dashboard Overview

Fire and Unauthorized Access alarm messages are sent through an email notification as shown in figures 2.4 and 2.5.



FIRE ALARM: Common Area

SmartBarracksNotification <its_lab_gm@libero.it>
A its_lab_gm@libero.it

Dear Recipient,

This is an urgent notification to inform you that a fire alarm has been detected.

Immediate action is required to ensure the safety of all personnel and property. Please proceed according to the emergency protocols and evacuate the premises if necessary.

Fire safety teams have been alerted and are on their way to assess and handle the situation.

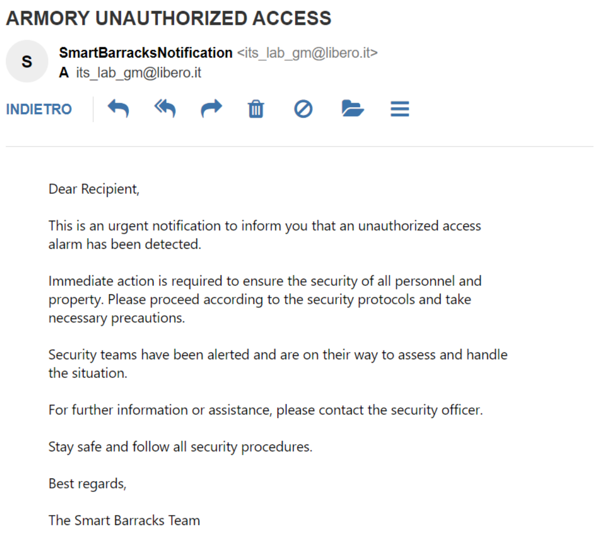
For further information or assistance, please contact the fire safety officer.

Stay safe and follow all emergency procedures.

Best regards,

The Smart Barracks Team

Figure 2.4: Fire Alarm email example



ARMORY UNAUTHORIZED ACCESS

SmartBarracksNotification <its_lab_gm@libero.it>
A its_lab_gm@libero.it

Dear Recipient,

This is an urgent notification to inform you that an unauthorized access alarm has been detected.

Immediate action is required to ensure the security of all personnel and property. Please proceed according to the security protocols and take necessary precautions.

Security teams have been alerted and are on their way to assess and handle the situation.

For further information or assistance, please contact the security officer.

Stay safe and follow all security procedures.

Best regards,

The Smart Barracks Team

Figure 2.5: Unauthorized Access email example

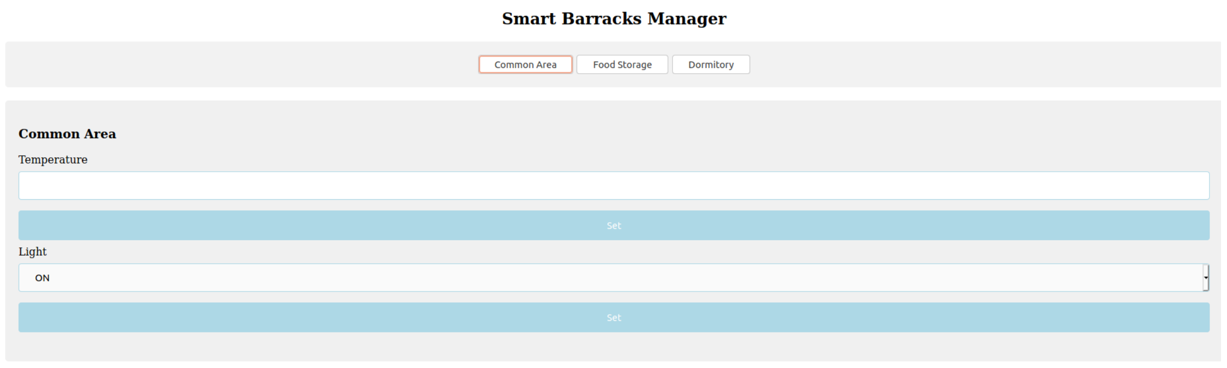
2.1.4 Webform

In order to manage the barracks' rooms, a web interface is implemented, allowing users to remotely set the sensors on and off. This interface is accessible at 127.0.0.1:1880/managebarracks, providing a convenient and centralized platform for environmental control and monitoring.

The main page of the web interface includes buttons for controlling three specific rooms: the Common Area, the Sleeping Quarters, and the Operations Room. Each button corresponds to one of these rooms, offering users the ability to manage their respective sensors and systems. The armory room is excluded from this interface due to its sensitive nature, and it cannot be accessed remotely for security reasons. This restriction ensures that the armory's security is not compromised by potential remote access vulnerabilities.

Figure 2.6 illustrates the form for the Common Area, serving as a reference for the layout and functionality of the room-specific management pages. Each form is designed to be straightforward and easy to use, with clear labels and controls for each sensor.

In summary, the web interface plays a crucial role in the efficient and secure management of the barracks' environmental systems, providing a user-friendly platform for remote monitoring and control.


Smart Barracks Manager

Common Area

Temperature

Light

ON

Figure 2.6: Sensor's Webform manager

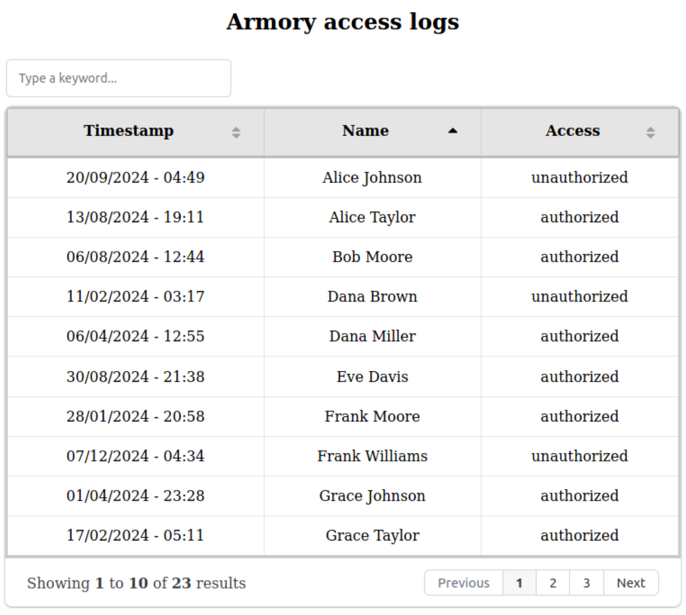
2.1.5 Armory Access Logs

A crucial aspect of managing the Armory is the ability to track and log access attempts, both authorized and unauthorized. This ensures that all entries are documented, enhancing security and accountability. The logs provide detailed information about who attempted to access the room and when, helping to maintain a secure environment.

The access logs can be viewed at 127.0.0.1:1880/login, where users are presented with a simple login interface. This interface is part of a proof of concept login system, with the only authorized user being 'admin' with the credentials "admin", "admin". For more information on this logging system, refer to section 3.1.8.

Upon a successful login, the user is redirected to a page that displays a table of access logs, as shown in figure 2.7. This table contains several key pieces of information:
Timestamp: The exact date and time when the access attempt was made.
Name: The name of the individual who attempted to access the Armory.
Access: The access level of the individual, indicating whether the person had the appropriate clearance to enter the Armory.

The table is designed to be user-friendly and functional, with several features to enhance usability:
Column Sorting: Each column in the table can be sorted individually, allowing users to organize the data according to their needs.
Search Functionality: A search bar is available in the top-left corner of the table, enabling users to perform quick searches based on any column data. This feature is particularly useful for finding specific entries or filtering the logs based on certain criteria.


Armory access logs

Timestamp	Name	Access
20/09/2024 - 04:49	Alice Johnson	unauthorized
13/08/2024 - 19:11	Alice Taylor	authorized
06/08/2024 - 12:44	Bob Moore	authorized
11/02/2024 - 03:17	Dana Brown	unauthorized
06/04/2024 - 12:55	Dana Miller	authorized
30/08/2024 - 21:38	Eve Davis	authorized
28/01/2024 - 20:58	Frank Moore	authorized
07/12/2024 - 04:34	Frank Williams	unauthorized
01/04/2024 - 23:28	Grace Johnson	authorized
17/02/2024 - 05:11	Grace Taylor	authorized

Showing **1** to **10** of **23** results

Figure 2.7: Armory Accesses

3

Application Implementation

3.1 Application Implementation

The developed solutions uses different technology:

- Java programming language for building a graphical interface that manages the sensors located in all the rooms of the barrack;
- Node-Red middleware that simulates the presence of real sensors interaction with the environment and acts accordingly with the rules shown in section 1;
- Mosquitto broker for managing the notifications sent with sensor's informations;
- MongoDB for storing users credentials.

3.1.1 Communication Protocols

Node-Red and the Java Simulator communicates with a custom protocol with the following message format:

GET [N] - for getting values from simulator

OK [N] $VALUE - response from simulator

Where N is the ID of the device to interact with and VALUE is the value read from device N.

SET [N] $VALUE - for setting values in simulator

OK [N] $VALUE - response from simulator

Where N is the ID of the device to interact with and VALUE is the value to set the device N with.

3.1.2 UML Diagrams

Use-Case Diagrams

In figure 3.1 are showed all the possible use cases.

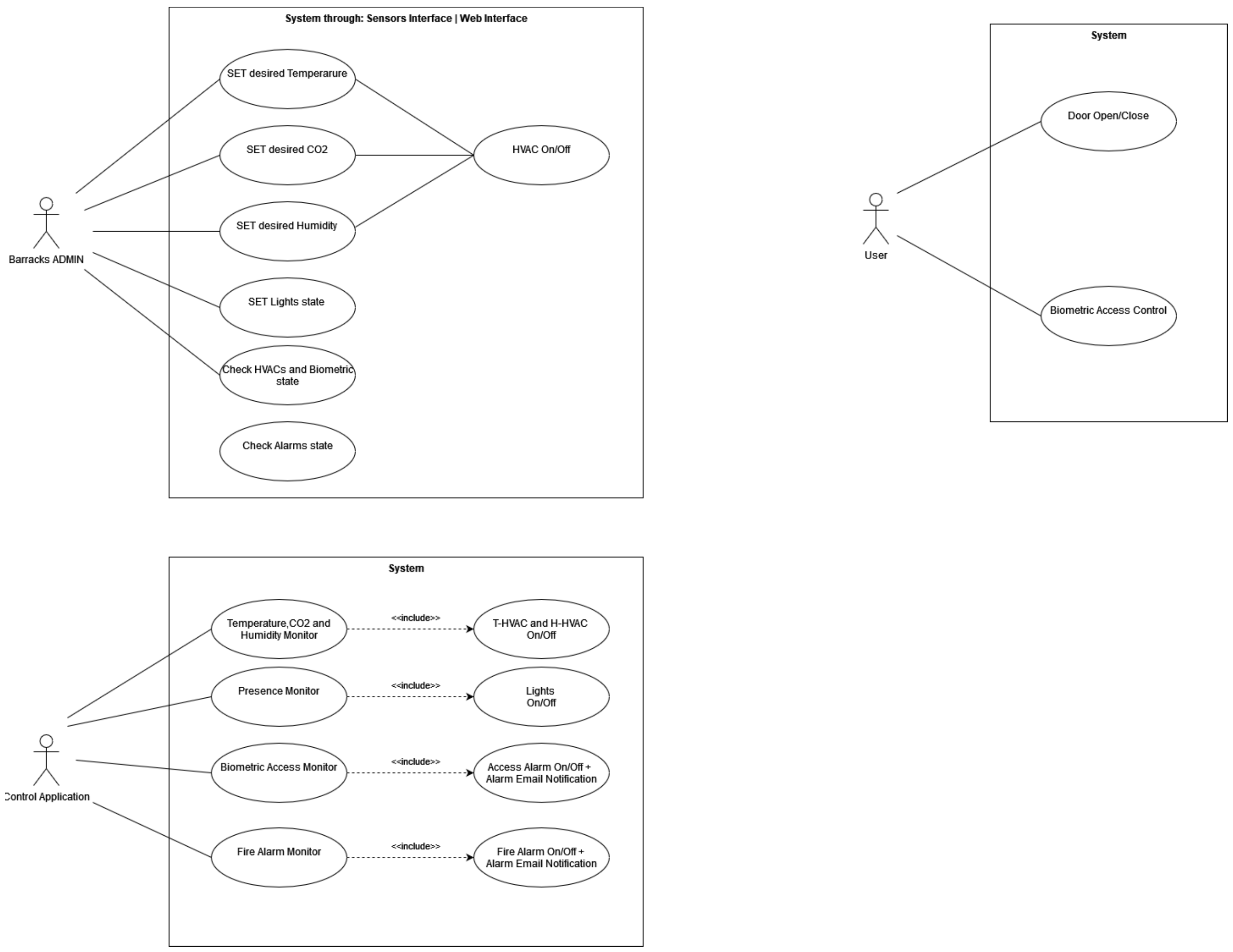


Figure 3.1: Use-case diagrams

Class Diagram

In figure 3.2 are showed the class diagram of the graphical user inteface written in Java for managing the sensors.

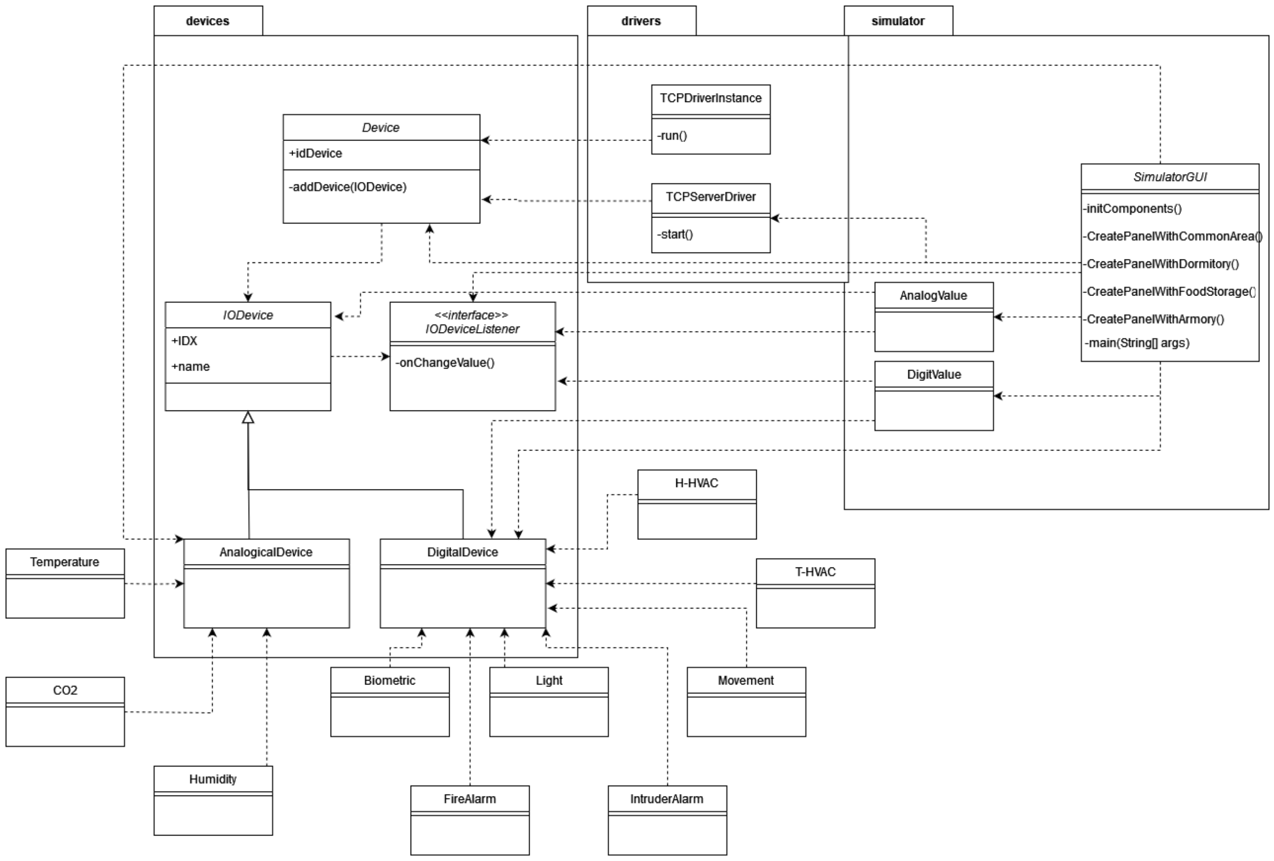


Figure 3.2: Class diagram

Sequence Diagram

In order to better understand how the system works here are the sequence diagrams for some of the core functionalities.
In figure 3.3 is shown a simple diagram for a sensor with no interactions with MQTT nor MongoDB.
In figure 3.4 is shown the diagram for the trigger of the fire alarms, here we can see the interaction with MQTT broker.
In figure 3.5 is shown the more complex sequence regarding the armory access alarm that intercts with MQTT, MongoDB and the email server.

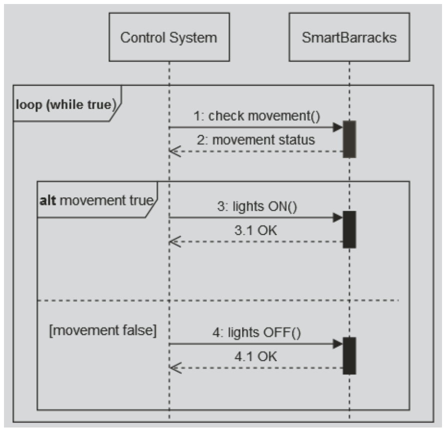


Figure 3.3: Simple sensor sequence diagram

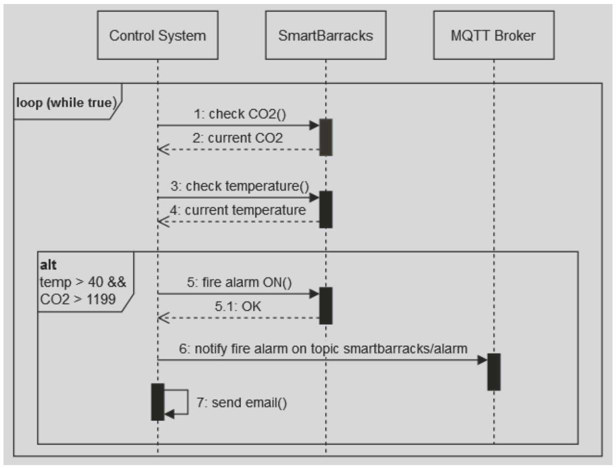


Figure 3.4: Fire Alarm sequence diagram

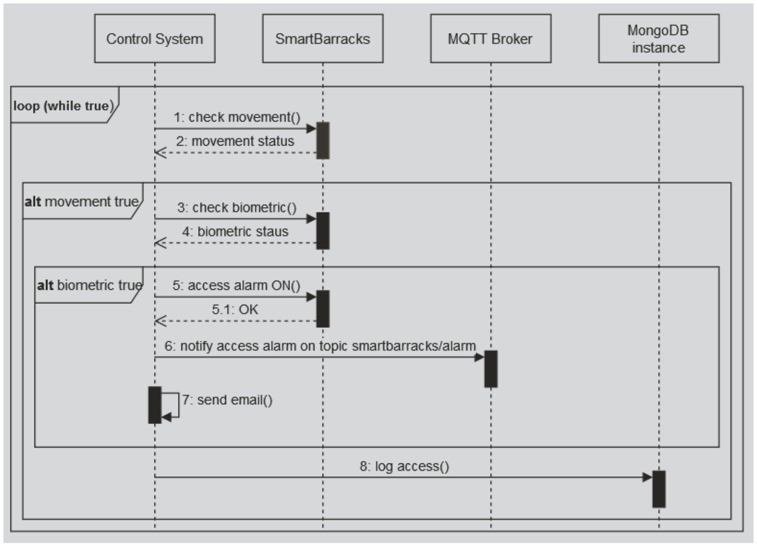


Figure 3.5: Access Alarm sequence diagram

3.1.3 Simulator

The Sensors Interface is created by the main class SimulatorGUI in Java. It utilizes standard Java Swing components and layouts to construct the interface. The main functionalities of SimulatorGUI are encapsulated in the following methods:

- The constructor SimulatorGUI() initializes the graphical user interface (GUI) components, sets up the layout for different rooms using JPanel components, and prepares for device initialization and communication.

```java
public SimulatorGUI() {

    try {
        UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
    } catch (UnsupportedLookAndFeelException | ClassNotFoundException
    | InstantiationException | IllegalAccessException e) {
        e.printStackTrace();
    }
    setTitle("Smart Barracks Monitoring System");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    initComponents();

    JPanel mainPanel = new JPanel(new GridLayout(2, 2, 6, 6));
    mainPanel.setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));

    commonAreaPanel = createPanelWithCommonArea();
    foodStoragePanel = createPanelWithFoodStorage();
    dormitoryPanel = createPanelWithDormitory();
    armoryPanel = createPanelWithArmory();

    mainPanel.add(commonAreaPanel);
    mainPanel.add(foodStoragePanel);
    mainPanel.add(dormitoryPanel);
    mainPanel.add(armoryPanel);

    instance = this;

    setContentPane(mainPanel);
    mainPanel.setPreferredSize(new Dimension(1400, 500));
    pack();

    Runnable task = () -> {
        devices = new Device("SmartBarracks");

        ///////////// COMMON AREA //////////////////////
        IODevice tmp = new AnalogicalDevice("Temperature");
        devices.addDevice(tmp);
        analogValue1.setDevice(tmp);
```

```
            tmp = new DigitalDevice("T-HVAC");
            devices.addDevice(tmp);
            digitValue1.setDevice((DigitalDevice) tmp);

            ...
            ...
            ...

            TCPServerDriver tp = new TCPServerDriver(devices);
            tp.start();
        };
        Thread thread = new Thread(task);
        thread.start();
    }
```

- The initComponents() method initializes all graphical components for each room in the interface, distinguishing between analogical devices represented by AnalogValue graphical components, digital devices represented by DigitValue graphical components and panels represented by JPanel graphical components.

```
    private void initComponents() {

        commonAreaPanel = new javax.swing.JPanel();
        jLabel3 = new javax.swing.JLabel();
        analogValue1 = new simulator.AnalogValue();
        digitValue1 = new simulator.DigitValue();
        digitValue2 = new simulator.DigitValue();
        digitValue3 = new simulator.DigitValue();
        digitValue4 = new simulator.DigitValue();
        analogValue8 = new simulator.AnalogValue();

        foodStoragePanel = new javax.swing.JPanel();
        jLabel4 = new javax.swing.JLabel();
        ...
        ...
        ...
    }
```

- The createPanelWith...() methods are utilized to configure the panels representing each room within the GUI with their respective components.

```
    private JPanel createPanelWithCommonArea() {

        JPanel panel = new JPanel();
        GridLayout layout = new GridLayout(3, 2);
        layout.setHgap(10);
        panel.setLayout(layout);
        panel.setBorder(BorderFactory.createTitledBorder("Common Area"));

        panel.add(analogValue1);
        panel.add(digitValue1);

        panel.add(analogValue8);
        panel.add(digitValue2);

        panel.add(digitValue3);
        panel.add(digitValue4);

        return panel;
    }
```

- The main(String[] args) method serves as the entry point to the application, where it calls the SimulatorGUI constructor to initialize and start the GUI.

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SimulatorGUI frame = new SimulatorGUI();
            frame.setVisible(true);
        });
    }
```

Moreover the simulator at startup contextually initializes the server at port 9090 where it listens for upcoming packets from Node-Red.

3.1.4 Sensor's control flows

The flows are organized essentially in two parts:

- first, some dummy data are randomly generated so that it is possible to simulate the behaviour of the systems without the need for physical devices,
- second, the control logic itself with the repeated cjecks on the sensors' values with the rules that will fire accordingly as explained in section 1.

As an example, in figure 3.6 is shown the random generation flow for the Common Area; it essentially perform a random generation after a time interval and sets the value in the simulator.

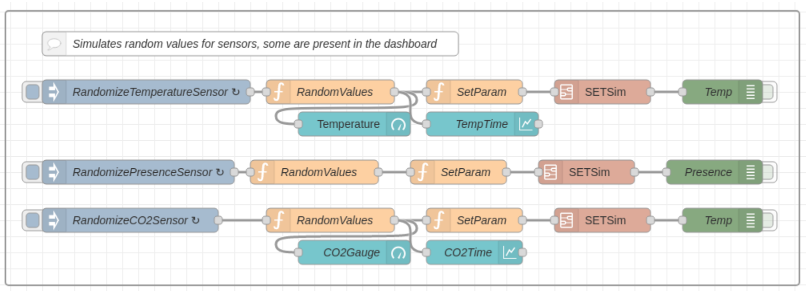


Figure 3.6: Generate random dummy data flow

Regarding the control logic, in figure 3.7 is shown a fragment of the control logic flow of the Common Area, for some of the sensors. Basically the check of sensors' values are performed, then, according to the needed use case, the simulator is set with the proper values.

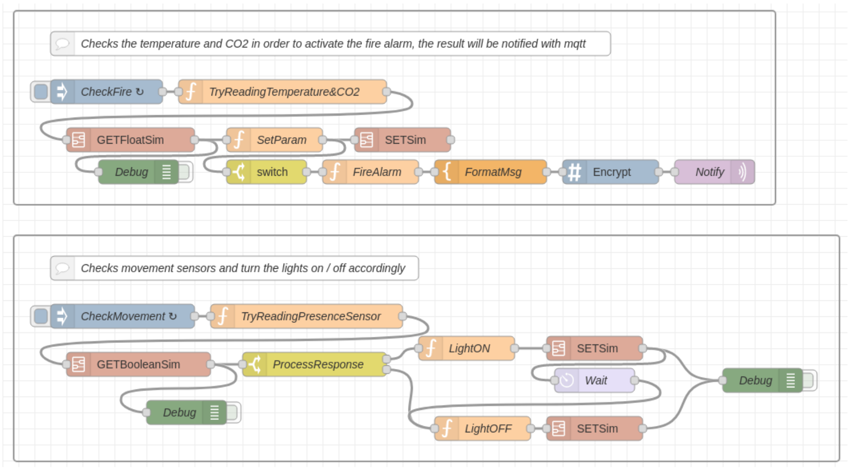

Checks the temperature and CO2 in order to activate the fire alarm, the result will be notified with mqtt
CheckFire ↻
TryReadingTemperature&CO2
GETFloatSim
SetParam
SETSim
Debug
switch
FireAlarm
FormatMsg
Encrypt
Notify
Checks movement sensors and turn the lights on / off accordingly
CheckMovement ↻
TryReadingPresenceSensor
LightON
SETSim
GETBooleanSim
ProcessResponse
Wait
Debug
Debug
LightOFF
SETSim

Figure 3.7: Generate random dummy data flow

3.1.5 UI - Dashboard

The sensors' dashboard is implemented through node-red simply with the node-red-dashboard plugin. In particular "gauge" and "chart" nodes has being used, attached after the randomization value segment in the flows that simulate the environment conditions that trigger the sensors.
In figure 3.8 an example of flows highlighting the dashboard's nodes.

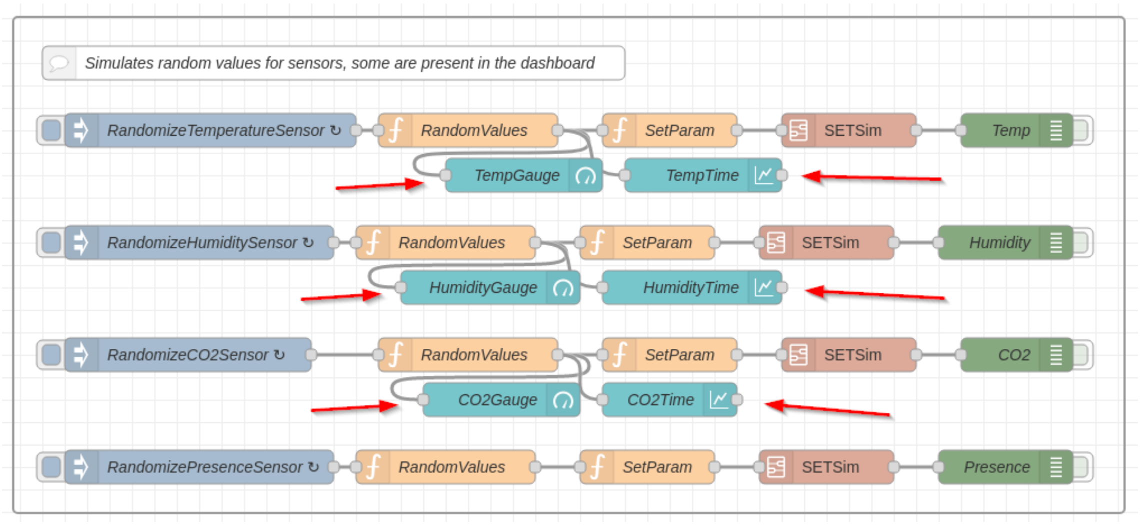


Figure 3.8: UI Dashboard reference flow

3.1.6 MQTT - Dashboard & Email Notifications

Similar to the UI dashboard, the MQTT dashboard is implemented using the node-red-dashboard plugin.
It employs an ”MQTT in” node to receive messages by subscribing to specific topics. The messages are then decrypted using the AES algorithm in a ”decryption” node and displayed using a ”text” node in the dashboard.
In figure 3.9 an example of the CommonArea dashboard flow highlighting the dashboard’s nodes.

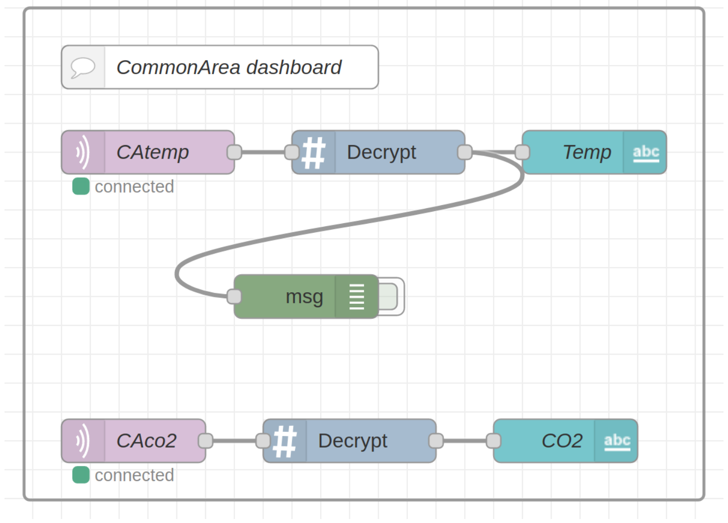


Figure 3.9: MQTT Dashboard reference flow

Data are sent from each Check flow of every barrack area. First, in the ”Log” function node, the payload description, content, and publication topic are set.
Then, the message is formatted in a ”template” node by including the payload description, payload content, and the measurement unit related to the checked value.
After formatting, the message is encrypted using the AES algorithm in an ”encryption” node. Finally, the message is published through an ”MQTT out” node.
In figure 3.10 an example of the Common Area CheckTemperature flow highlighting the message sending nodes.

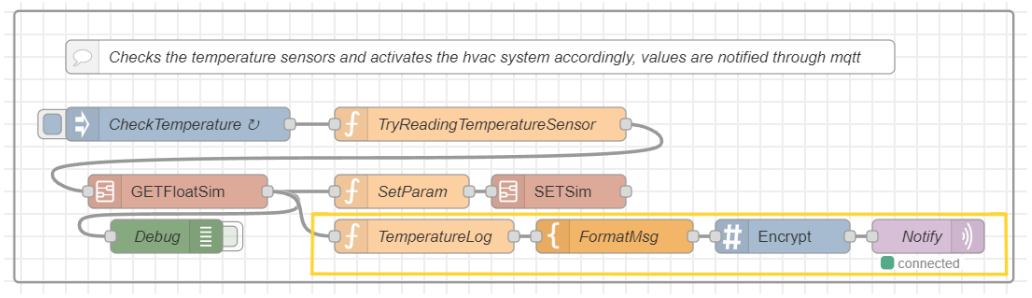

Checks the temperature sensors and activates the hvac system accordingly, values are notified through mqtt
CheckTemperature ↻
TryReadingTemperatureSensor
GETFloatSim
SetParam
SETSim
Debug
TemperatureLog
FormatMsg
Encrypt
Notify
connected

Figure 3.10: MQTT out reference flow

For both the Fire and Unauthorized Access alarms, the same approach is utilized. All previously described nodes are implemented, with the addition in the ”Log” function node where the payload.description is set to denote the type of active alarm and, specifically for the Fire alarm, the name of the respective room.

The process of sending email notifications is implemented similarly to the Dashboard flow. Initially, the message is received via subscription to the relevant topic in an 'MQTT in' node. Subsequently, it undergoes decryption.
Following this, the email contents such as sender address, sender name, topic, and message body are configured. Finally, the message is forwarded through an 'e-mail out' node, which is provided by the node-red-contrib-email-out module.
This node allows specification of parameters such as the destination address, SMTP server and port provided by the email service provider, and mailbox credentials.
In figure 3.11 the Alarm notifications flow highlighting the message sending nodes.

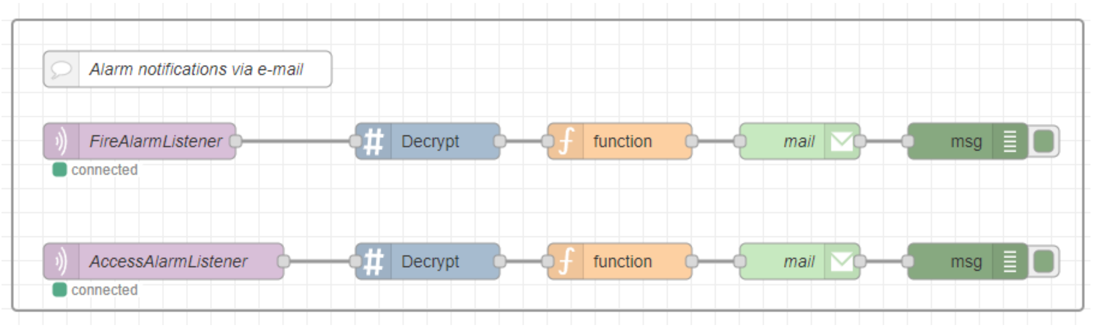


Figure 3.11: Email notifications reference flow

Moreover in figure 3.12 is provided the MQTT topics hierarchy used.

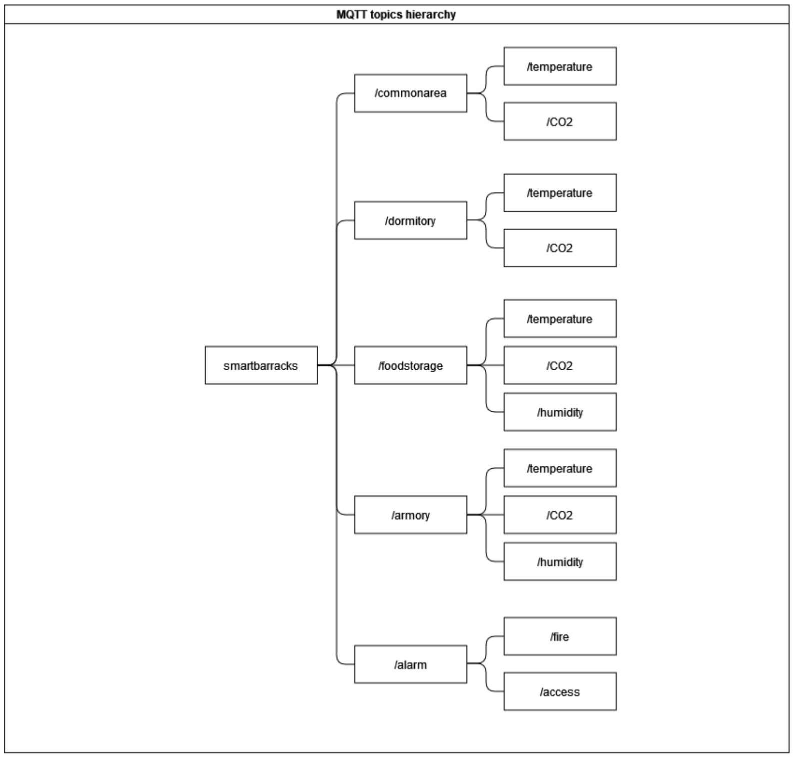


Figure 3.12: MQTT topics structure

3.1.7 Webform

The web form page is implemented in node-red using "http-in", "function", "template" and "http-response" nodes for the webform building part as shown in figure 3.13.

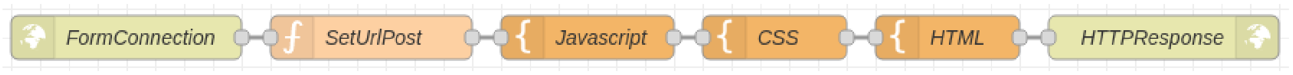


Figure 3.13: Webform flow

The flow works as follows:

- FormConnection sets the GET method and the url (/managebarracks) to access the webpage;
- SetUrlPost sets the url where the form retrieved data will be posted;
- Javascript, CSS, HTML nodes builds the webform page itself and the management of the data that will be passed in the form, along with the code that hide the form of the not selected room;
- HTTPResponse close the flow.

The form management data input is managed by this javascript snippet:

```
$(document).ready(function(e) {
    $("form[ajax=true]").submit(function(e) {

    e.preventDefault();

    var form_data = $(this).serialize();
    var form_url = $(this).attr("action");
    var form_method = $(this).attr("method").toUpperCase();

    $.ajax({
        url: form_url,
        type: form_method,
        data: form_data,
        cache: false,

        success: function(returnhtml){
            alert("Data successfully transmitted");
        }
    });
    });
});
```

This code basically sets up an AJAX form submission process that prevents the default form submission, serializes the form data, and sends it to the server via AJAX, displaying a success message upon completion.

The last part of the flow implements the data processing part, the basic pipeline follows these steps:

- <room>Post catches the incoming data:
  - json transforms those into json format;
  - Result alerts the correct transmission of data;
  - HTTPResponse close the flow.
- StorePayload copy the payload so it can be processed;
  - then the logic for each sensor slightly chenges, the basic workflows include setting the sensor's id / value and then send those data to the simulator, as in figure 3.14.

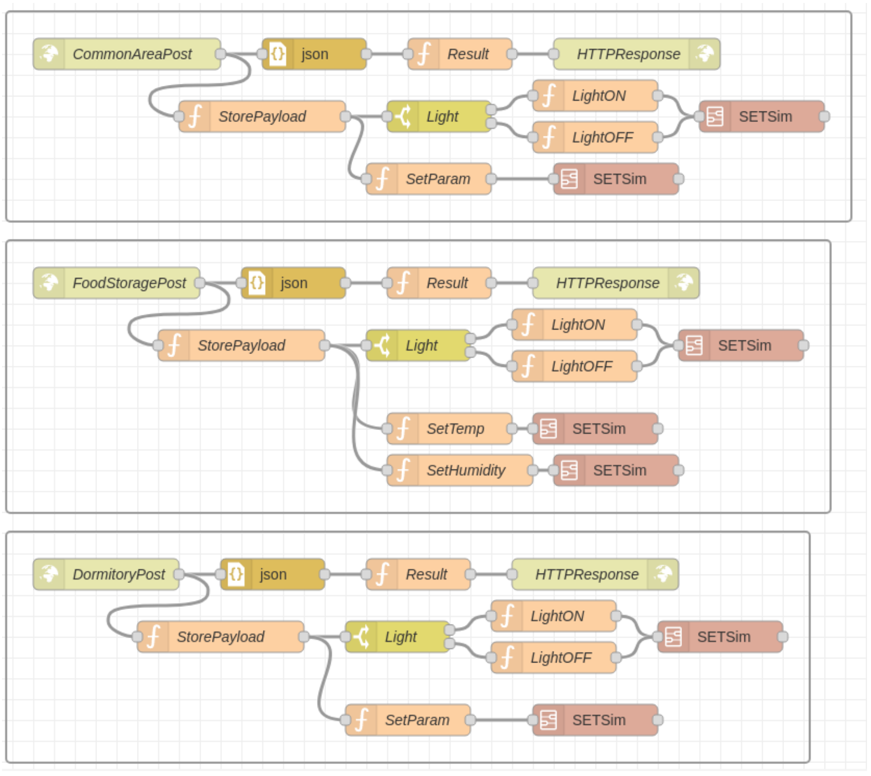


Figure 3.14: Catching data flows

3.1.8 Armory Access Logs

The data are stored into a mongodb collection, after the randomization of the motion detection sensor another randomization is done in order to generate credentials that simulates an access from a user approaching the biometric sensor.
The complete flow with also the MQTT implementation is shown in figure 3.15.

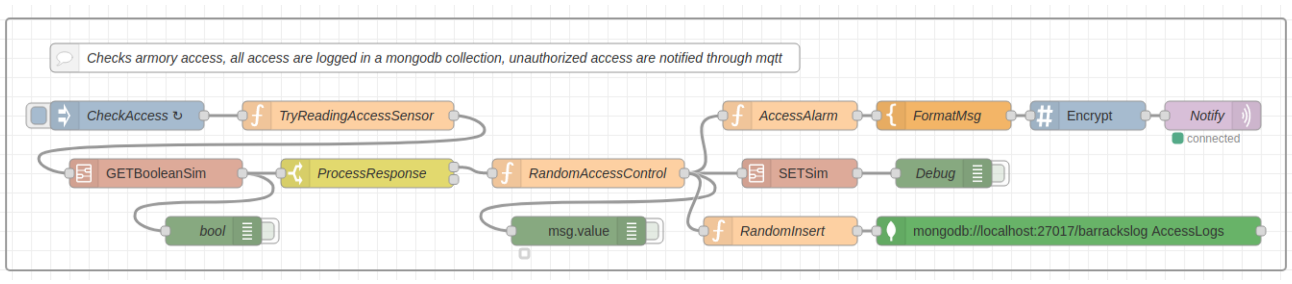

Checks armory access, all access are logged in a mongodb collection, unauthorized access are notified through mqtt
CheckAccess ↻
TryReadingAccessSensor
AccessAlarm
FormatMsg
Encrypt
Notify
connected
GETBooleanSim
ProcessResponse
RandomAccessControl
SETSim
Debug
bool
msg.value
RandomInsert
mongodb://localhost:27017/barrackslog AccessLogs

Figure 3.15: Storing access flow

The web page that shows the logs is structured into two sub-parts:

- first, a basic login page is built with the same logic as the webform shown in section 3.1.7,
- then, after the login the database is queried and the information are displyed.

The "Transform Payload" node simply process the query result so it is simpler to elaborate for the "ArmoryAccessLogTable" which builds the webpage itself and the table with the logs using the library Grid.js.
The complete flow is shown in figure 3.16.

Note that if a user tries to access the logs page directly typing 127.0.0.1:1880/armorylogs in the search bar of the browser, it will be automatically redirected to the login page, thanks to this line of code:

```
window.onload = function() {
    if(document.referrer == "") window.location.href = "http://localhost:1880/login";
}
```

It is important to underline that this is a proof of concept login systems so the only authorized user is admin with "admin","admin" credentials which are not stored in a database, obviously in a real implementation the authorized users' credentials has to be stored in any kind of database, retrieved and then checked.

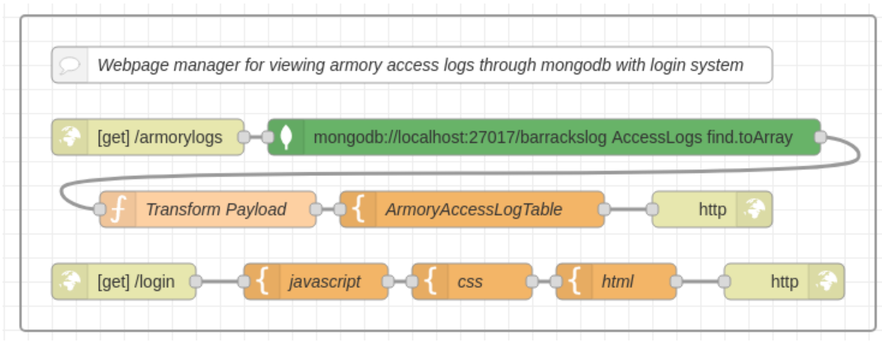

Webpage manager for viewing armory access logs through mongodb with login system
[get] /armorylogs
mongodb://localhost:27017/barrackslog AccessLogs find.toArray
Transform Payload
ArmoryAccessLogTable
http
[get] /login
javascript
css
html
http

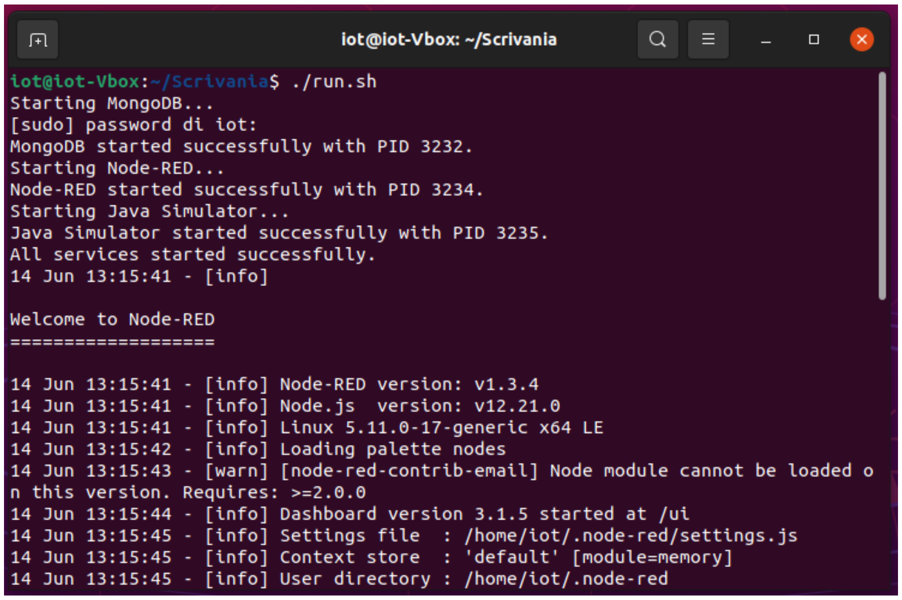
Figure 3.16: Accesses webpage flow

For reference here follows a partial dump of the database:

```
[
    {
        "_id" : ObjectId("66587b557528f40a4a03690d"),
        "timestamp" : "21/04/2024 - 02:47",
        "name" : "John Davis",
        "access" : true
    },
    {
        "_id" : ObjectId("66587b647528f40a4a03690e"),
        "timestamp" : "13/08/2024 - 19:11",
        "name" : "Alice Taylor",
        "access" : false
    },
    {
        "_id" : ObjectId("66587b6b7528f40a4a03690f"),
        "timestamp" : "06/08/2024 - 12:44",
        "name" : "Bob Moore",
        "access" : false
    },
    {
        "_id" : ObjectId("66587b727528f40a4a036910"),
        "timestamp" : "01/04/2024 - 23:28",
        "name" : "Grace Johnson",
        "access" : false
    },
    {
        "_id" : ObjectId("66587b797528f40a4a036911"),
        "timestamp" : "05/08/2024 - 03:28",
        "name" : "Hank Miller",
        "access" : false
    },
    {
        "_id" : ObjectId("66587b897528f40a4a036912"),
        "timestamp" : "07/05/2024 - 09:52",
        "name" : "Hank Jones",
        "access" : true
    },
    {
        "_id" : ObjectId("66587b8f7528f40a4a036913"),
        "timestamp" : "25/07/2024 - 05:08",
        "name" : "John Wilson",
        "access" : false
    },
    {
        "_id" : ObjectId("66587ba37528f40a4a036914"),
        "timestamp" : "12/05/2024 - 09:29",
        "name" : "Jane Miller",
        "access" : false
    },
    {
        "_id" : ObjectId("66587bc17528f40a4a036915"),
        "timestamp" : "20/09/2024 - 04:49",
        "name" : "Alice Johnson",
        "access" : true
    },
    {
        "_id" : ObjectId("66587bc67528f40a4a036916"),
        "timestamp" : "07/12/2024 - 04:34",
        "name" : "Frank Williams",
        "access" : true
    },
    {
        "_id" : ObjectId("6666aebe6cc1831bd7c2a36e"),
        "timestamp" : "29/09/2024 - 02:55",
        "name" : "Hank Miller",
        "access" : false
    },
    {
        "_id" : ObjectId("6666aeee6cc1831bd7c2a36f"),
        "timestamp" : "17/06/2024 - 17:40",
        "name" : "John Williams",
        "access" : false
    },
    {
        "_id" : ObjectId("6666aefe6cc1831bd7c2a370"),
        "timestamp" : "30/08/2024 - 21:38",
        "name" : "Eve Davis",
        "access" : false
    }
]
```

3.1.9 Autorun script

In order to make the setup process simpler, a run script is implemented so that it automatically starts mongodb, node-red, and the interface that simulates the sensors.
Simply typing './run.sh' in the terminal where the file is, all the required software will start.
When asked for sudo password, type "iot" without quotes, as in figure 3.17.
If the script gives an error, type "chmod +x run.sh" without quotes to make the script executable.



```
iot@iot-Vbox:~/Scrivania$ ./run.sh
Starting MongoDB...
[sudo] password di iot:
MongoDB started successfully with PID 3232.
Starting Node-RED...
Node-RED started successfully with PID 3234.
Starting Java Simulator...
Java Simulator started successfully with PID 3235.
All services started successfully.
14 Jun 13:15:41 - [info]

Welcome to Node-RED
===================

14 Jun 13:15:41 - [info] Node-RED version: v1.3.4
14 Jun 13:15:41 - [info] Node.js  version: v12.21.0
14 Jun 13:15:41 - [info] Linux 5.11.0-17-generic x64 LE
14 Jun 13:15:42 - [info] Loading palette nodes
14 Jun 13:15:43 - [warn] [node-red-contrib-email] Node module cannot be loaded o
n this version. Requires: >=2.0.0
14 Jun 13:15:44 - [info] Dashboard version 3.1.5 started at /ui
14 Jun 13:15:45 - [info] Settings file  : /home/iot/.node-red/settings.js
14 Jun 13:15:45 - [info] Context store  : 'default' [module=memory]
14 Jun 13:15:45 - [info] User directory : /home/iot/.node-red
```

Figure 3.17: Run script output

Here the most relevant script's lines:

```bash
#!/bin/bash

# Function to start MongoDB
start_mongodb() {
    echo "Starting MongoDB..."
    sudo systemctl start mongod
    MONGODB_PID=$(pgrep mongod)

    if [ $? -eq 0 ]; then
        echo "MongoDB started successfully with PID $MONGODB_PID."
    else
        echo "Failed to start MongoDB."
        exit 1
    fi
}

# Function to start Node-RED
start_nodered() {
    echo "Starting Node-RED..."
    node-red &
    NODERED_PID=$!
    if [ $? -eq 0 ]; then
        echo "Node-RED started successfully with PID $NODERED_PID."
    else
        echo "Failed to start Node-RED."
        exit 1
    fi
}

# Function to start Java Simulator
start_simulator() {
    echo "Starting Java Simulator..."
    java -cp SmartBarracks/target/classes simulator.SimulatorGUI &
    JAVA_APP_PID=$!

    if [ $? -eq 0 ]; then
        echo "Java Simulator started successfully with PID $JAVA_APP_PID."
    else
        echo "Failed to start Java Simulator."
        exit 1
    fi
}

# more code

# Trap SIGINT (Ctrl+C) to run cleanup
trap cleanup SIGINT

# Starting the services
start_mongodb
start_nodered
start_simulator

echo "All services started successfully."
```