CSLAB

ISEP - DEI - MESCC

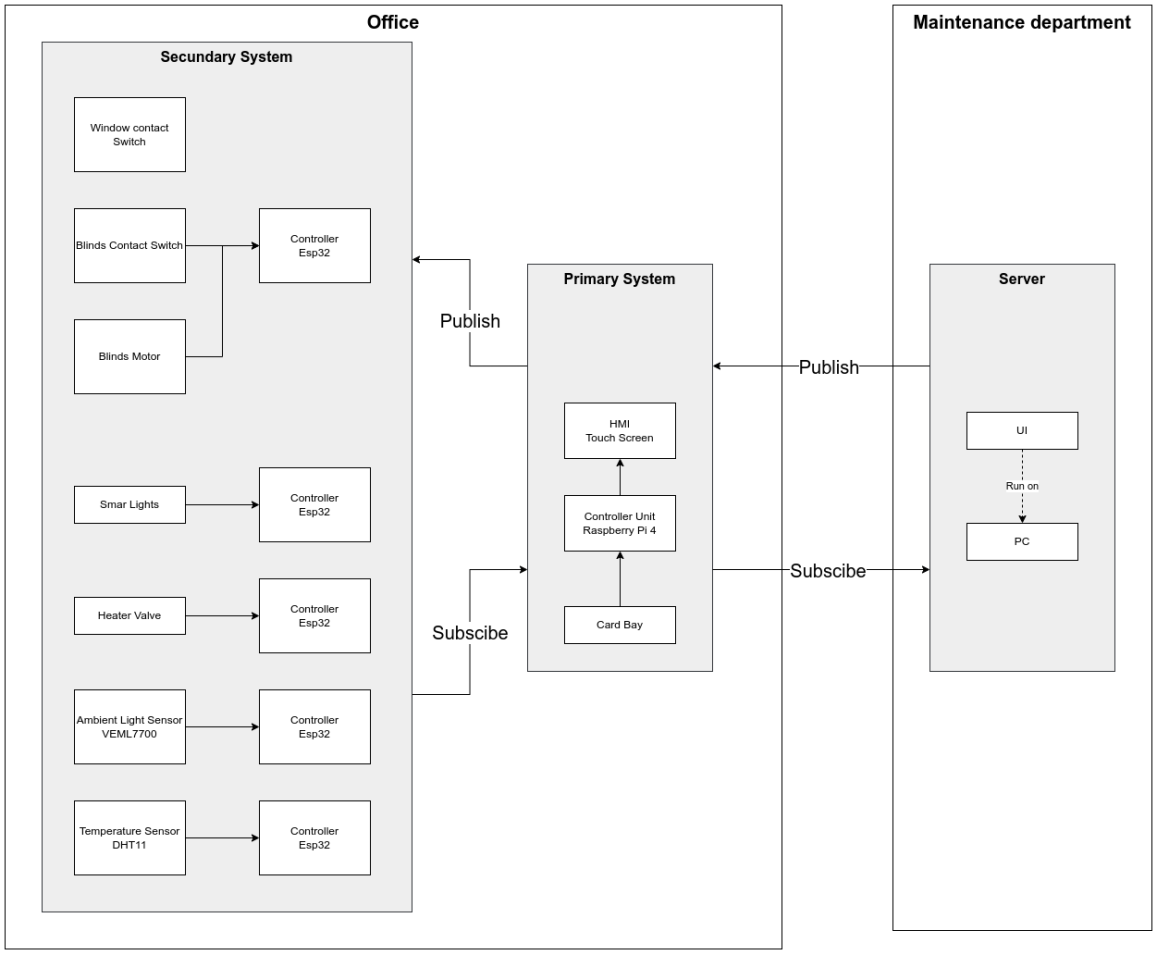
Smart Lighting and Heating System

Group 4:
Carlos Rijo - 1101626
João Fernandes – 1221973

1. Introduction

In this document, we will discuss the technologies we implemented in our project and the reasoning behind our choices. Our goal is to provide a clear and concise overview of the technologies we have selected and how they w ere used to support the success of our project. We hope this document will serve as a useful reference for all stakeholders and help everyone understand the technological decisions made.

To better understand our system, we need to describe what devices will need to communicate with each other and what data they will need to send to each other.
We made a high-level diagram where we described our system which we can see below:



In this diagram we describe the project we implemented, an automated climate and lighting control system. This system needs to be able to control three actuators: blinds, lights, and a heater for the user or maintenance team to set a desired temperature or luminosity for a target room.
In order to make this project possible we utilized a variety of hardware components which we will list below:

- Secondary System
  - 5x ESP-32
  - 2 VEML7700 Light Sensors (unable to make these redundant due to hardware shortages)
  - 2 DHT11
  - I2C multiplexer
  - Servo motor
  - 5 Push Buttons
  - LED's
  - Various resistors
- Primary System
  - Raspberry Pi 4 B
- Server
  - PC

2. Requirements (RAMDE)

In our project, we have created system requirements using the mbedder tool and implemented them on our project to ensure the smooth and efficient operation of our software.

We evaluated these system requirements throughout the development process to ensure that they were met and that the software is functioning as expected. This involved regular testing and monitoring of the system's resources and performance, and user feedback to ensure the software meets their needs and expectations.

The use of mbedder helped us have a clear and concise overview of the requirements and helps in testing and validation of them.

Bellow we will list the requirements we created in mbedder and will classify them as accomplished and not accomplished:

1 **Operational Requirements –** **Accomplished**

The system shall manage the ambient light and temperature inside the office by controlling the lights, blinds, and heater.

1.1 Sensors Requirements - **Accomplished**

The system shall be equipped with sensors to monitor the ambient light and temperature.

1.1.1 Sensor Communication Requirements – **Accomplished**

The sensors shall communicate wirelessly to a control unit@req(controlUnit).

1.1.2 Sensor Redundancy Requirement - **Partially Accomplished – not able to accomplish on the light due to global shortage of hardware.**

The sensors shall have redundancy.

1.1.2.1 Ambient Light Sensor Redundancy Requirements - **Not Accomplished -** due to global shortage of hardware.

The ambient light sensors@req(amblightSensor) shall have, at least, two sensors inside the office and two sensors outside the office.

1.1.2.2 Temperature Sensor Redundancy Requirements – **Accomplished**

The temperature sensors@req(tempSensor) shall have, at least, two sensors inside the office.

1.1.3 Sensor Message Requirements - **Accomplished**

When the sensors are not connected, they shall send to the control unit@req(controlUnit) an error message

1.1.3.1 Sensor Message Requirements - **Partially Accomplished**

The error message shall be composed by office ID, sensor type, average of readings and error code.

1.2 Parameters Requirements - **Accomplished**

The system shall have a determined range of max values and set points

1.2.1 Parameters Requirements - **Partially Accomplished**

The system parameters shall be:@req(tempMaxValue), @req(tempMinValue), @req(amblightMaxValue), @req(amblightMinValue), @req(tempSetPoint) and @req(amblightSetPoint).

1.3 Actuators Requirements - **Accomplished**

The system shall be equipped with a set of lights@req(lights), a blinds@req(blinds) and a heater@req(heater).

1.3.1 Actuators Communication Requirements - **Accomplished**

The actuators shall communicate wirelessly to a control unit@req(controlUnit).

1.3.2 Actuators Timeout Requirements - **Not Accomplished**

The actuators shall have an expected time (@req(actTimeoutValue)) to reach the target value/position

1.4 User Detection Requirements- **Accomplished**

The system shall have the capability to know if the@req(user) is in the office.

1.5 Control Unit Requirements – **Accomplished**

The system shall have a control unit@req(controlUnit) to control the environment of the office

1.5.1 Control Unit Communication Requirements - **Accomplished**

The control unit@req(controlUnit) shall communicate wirelessly with the sensors and actuators.

1.5.2 Control Unit Interface Requirements - **Accomplished**

The control unit@req(controlUnit) shall be equipped with a human-machine interface (@req(hmi)) with the overall status of the office and with buttons to control each actuator manually.

1.5.3 Control Unit Error Requirements- **Accomplished**

The control unit@req(controlUnit) shall receive errors from the sensors and actuators and send (wirelessly) the information to the maintenance department (@req(maintenanceDep)).

1.5.4 Control Unit Log Requirements - **Partially Accomplished**

The control unit@req(controlUnit) shall log each event of the system

1.6 Maintenance Department Control Requirements- **Accomplished**

The Maintenance department@req(maintenanceDep) shall always be able to override and block the settings defined in each office.

1.6.1 Maintenance Department Interface Requirements - **Accomplished**

The maintenance department@req(maintenanceDep) shall be equipped with a user interface (@req(ui)) with the overall status of each office.

2 Behaviour Requirements - **Accomplished**

The system shall behave according to@req(operationalReq)

2.1 Blinds Behaviour Requirements - **Accomplished**

The blinds@req(blinds) shall adjust according to the office ambient light defined set point.

2.1.1 Blinds Behaviour Requirements - **Accomplished**

The blinds@req(blinds) shall move in intervals of 25%.

2.1.2 Blinds Behaviour Requirements - **Accomplished**

The blinds@req(blinds) shall have sensors@req(blindsPosition) to detect when the blind is in the desired position

2.1.3 Blinds Behaviour Requirements - **Accomplished**

The blinds@req(blinds) shall be able to be controlled autonomously or manually (by the@req(user) or by the maintenance department@req(maintenanceDep))

2.2 Lights Behaviour Requirements - **Accomplished**

The lights@req(lights) shall have the capability to adjust its brightness according to the office ambient light defined set point.

2.2.1 Lights Behaviour Requirements - **Accomplished**

The lights@req(lights) shall be able to be controlled autonomously or manually (by the@req(user) or by the maintenance department@req(maintenanceDep)).

2.3 Heater Behaviour Requirements - **Accomplished**

The heater@req(heater) shall have the capability to adjust its temperature according to the office temperature defined set point.

2.3.1 Heater Behaviour Requirements - **Accomplished**

The heater@req(heater) shall be able to be controlled autonomously or manually (by the@req(user) or by the maintenance department@req(maintenanceDep)).

3 Safety Requirements-safetyReq

3.1 Sensor Error Requirements - **Accomplished**

When an error is detected, the sensor shall report an error. Hazards:@req(sensorHazard)

3.1.1 Sensor Error handling Requirements - **Accomplished**

The system shall have the capacity to handle sensor errors.

3.2 Actuators Error Requirements - **Accomplished**

When an error is detected, the actuator shall report an error. Hazard:@req(actuatorHazard)

3.2.1 Actuators Error handling Requirements - **Accomplished**

The system shall have the capacity to handle actuator errors.

3.3 Window Requirements - **Accomplished**

The windows shall have contact switches@req(blindsWindow) to check window status.Hazard:@req(windowSecHazard)

3. Development

To make it easier for the lecturers to correct and evaluate each topic, we have decided to split the course material into four distinct areas of focus. These areas are RTAES, COMCS, CCSYA and CSLAB. By breaking down the material in this way, lecturers will be able to focus on the specific topics more effectively at hand and provide more detailed feedback. Furthermore, this will allow us to better understand the material, as they can focus on one area at a time, rather than trying to take in all the information at once.

a) RTAES

Regarding our shortcomings last week regarding the real-time component of our project, we needed to change our development path to meet the expected goals.

Unfortunately, we could not complete the topic at hand in time. Since we were not able to implement in code what we planned, we will instead describe it properly in this document.

Our plan was to use the FreeRTOS operating system on our ESP32 microcontrollers to control our tasks in real time. These tasks were designed to take full advantage of the capabilities of the ESP32, and we had planned to implement a total of five different schedulers since we had 5 ESP's running 5 different codes.

Each of these tasks would have been designed to run concurrently and independently, allowing us to take full advantage of the multi-tasking capabilities of the ESP32. We had also planned to use the FreeRTOS scheduler to manage the execution of these tasks and ensure that they were running in a timely and efficient manner.

Below we will describe the tasks we are running on our ESP's and how we would run the priorities on them. The higher the value the bigger the priority(0-255).

- ESP32 – Blinds
    - Communication with MQTT (Publish and Subscribe) – Priority 255
    - Reading of values – Priority 253
    - Actuators - Priority 254
- ESP32 – Heater
    - Communication with MQTT – Priority 255
    - Actuators – Priority 254
- ESP32 – Lights
    - Communication with MQTT – Priority 255
    - Actuators – Priority 254
- ESP32 – Light Sensors
    - Communication with MQTT – Priority 255
    - Reading of values – Priority 254
- ESP32 – Temperature Sensors
    - Communication with MQTT – Priority 255
    - Reading of values – Priority 254

Our plan was to use the FreeRTOS operating system on our ESP32 microcontrollers to control our tasks in real time. One of the main challenges we faced was the need to run more than one task on each ESP32, as each one was responsible for a different aspect of our project.

One important aspect of our project was the communication between the ESP32s and our sensors and actuators. This required the use of the MQTT protocol, which needed high priority to ensure reliable and timely communication.

Another important aspect of our project was the need to manage shared resources, such as memory and I/O ports. We had planned to use the FreeRTOS resource management functions to ensure that these resources were used efficiently and without conflicts.

To conclude we could also implement core affinity since the ESP-WROOM-32D has two cores that can be manipulated individually.

In summary, our plan was to use FreeRTOS on the ESP32s to control tasks in real time, unfortunately, due to lack of time, we were not able to complete the implementation of these tasks, but we have thoroughly described them and their functionalities in this document for future reference.

Since we did not code our control unit in python and instead coded everything in node red, we decided to run both our instances of node red on the server and RPI with maximum priority.

b) COMCS

Regarding the communication part of our project, we used TCP (Transmission Control Protocol) connections between our ESPs, our Raspberry Pi (RPI) and our server. To facilitate the communication between the devices within the TCP connections, we used the MQTT protocol, which allowed us to send and receive data in a reliable and efficient manner.

Our ESPs were responsible for reading the values from our sensors and sending them to our Raspberry Pi (RPI) via MQTT over the TCP connection. The RPI would then process these values and use them to control the actuators on our ESPs. The RPI also used MQTT to send the control commands to the ESPs. This allowed for real-time monitoring and control of our system, as the ESPs could send sensor readings and receive commands from the RPI in near real-time.

We could also control and monitor our system from the touch screen of our Raspberry Pi (RPI) with limited control, while we had full control from our server GUI. This allowed for flexibility in terms of access and control of the system, as the touch screen provided a convenient local interface for basic control, while the server GUI allowed for more advanced control and monitoring capabilities. The GUI was accessible from the network, which gave us the ability to control and monitor the system from anywhere.

Furthermore, the MQTT protocol allowed us to easily scale our system, as new devices could be added and integrated into the system by connecting them to the MQTT broker. The MQTT protocol also provided a lightweight and efficient way to send and receive data, which was important as we had multiple devices communicating with each other. MQTT also provided a publish-subscribe model, which allowed for decoupled communication between devices, making the system more robust and reliable.

In our project, we had two MQTT brokers, one running on the Raspberry Pi (RPI) and one running on the server. The RPI broker was responsible for managing the communication between the ESPs and the RPI, while the server broker was responsible for managing the communication between the RPI and the server. This allowed for more efficient and secure communication within our system. The RPI broker also acted as a local gateway, allowing the ESPs to communicate with the server broker.

Overall, we feel we successfully accomplished our goal of implementing a robust and efficient communication system within our project. We used the MQTT protocol to facilitate communication between our ESPs, RPI and server and provide a way to easily scale and add new devices to the system. Our use of two MQTT brokers also added an extra layer of security and reliability to our system. We were also able to provide various levels of control and monitoring of our system using the touch screen on the RPI and the server GUI. We are confident that this communication system is performing well in our final project.

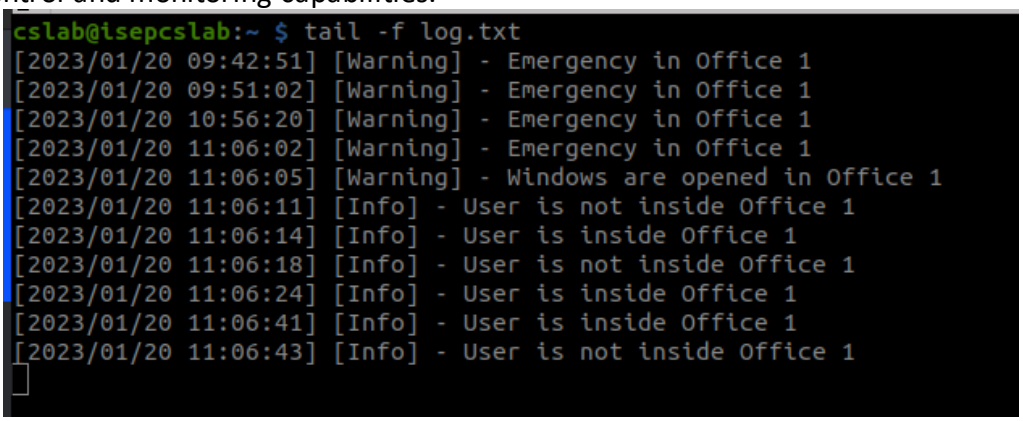
c) CCSYA

Despite facing numerous challenges, we are proud to announce that we have successfully implemented the assembly code in our project. One of the main obstacles we encountered was figuring out how to manipulate the register I/Os on the ARM architecture of the Raspberry Pi 4.

The process of understanding and manipulating the register I/Os took a significant amount of time and effort, which slowed the implementation of other features. As a result, we have implemented an assembly code that, when the emergency button is pressed, creates a log file with information about the room in which the emergency button was pressed as well as the time it was pressed. This is a crucial feature for ensuring safety and security in our system. The code we reference will be sent as an attachment with the name emergency.s. In addition to the assembly code for the emergency button, we also created three other scripts that further enhanced the safety and security of our system. The first and second script was added to the log file, monitoring when the user entered and exited the office, and the time of this event. This feature allows us to have a record of who is in the office at any given time and helps manage office occupancy.

The third script logged when the window was opened and closed, providing an additional layer of security, ensuring that the office is not left open or vulnerable when no one is present. This script also allows for managing the office's energy consumption by monitoring when the window is opened or closed. The other assembly scripts we used are the following userIN.s (logs the entrance), userOUT.s (logs the exit) and finally the window.s (logs the opening and closing of the windows)

Overall, implementing these assembly scripts enhanced the safety and security of our system and provided high control and monitoring capabilities.



d) CSLAB

Our project, which was developed in the CSLAB, aims to create a system that will control and monitor the temperature and ambient light of the offices in the Cister building. This system will use a heater, blinds, and lamps as actuators to adjust the temperature and light levels in the offices. The project's goal is to create a comfortable and energy-efficient environment for the occupants of the Cister building.

The system will be able to automatically adjust the temperature and light levels based on the occupancy and desired settings, as well as allow manual adjustments through a user interface. The project is expected to have a positive impact on energy consumption and overall comfort of the building.

This week, we were able to make noteworthy progress on our project in the CSLab. One of the key accomplishments was the completion of both user interfaces using Node-RED. These interfaces will allow users to manually adjust the temperature and light levels in the offices as well as view the current settings.

Another important aspect of the project is the communication between the system and the actuators, which we successfully implemented using MQTT protocol. With these key components in place, we were able to test and refine the system's functionality to meet our requirements.

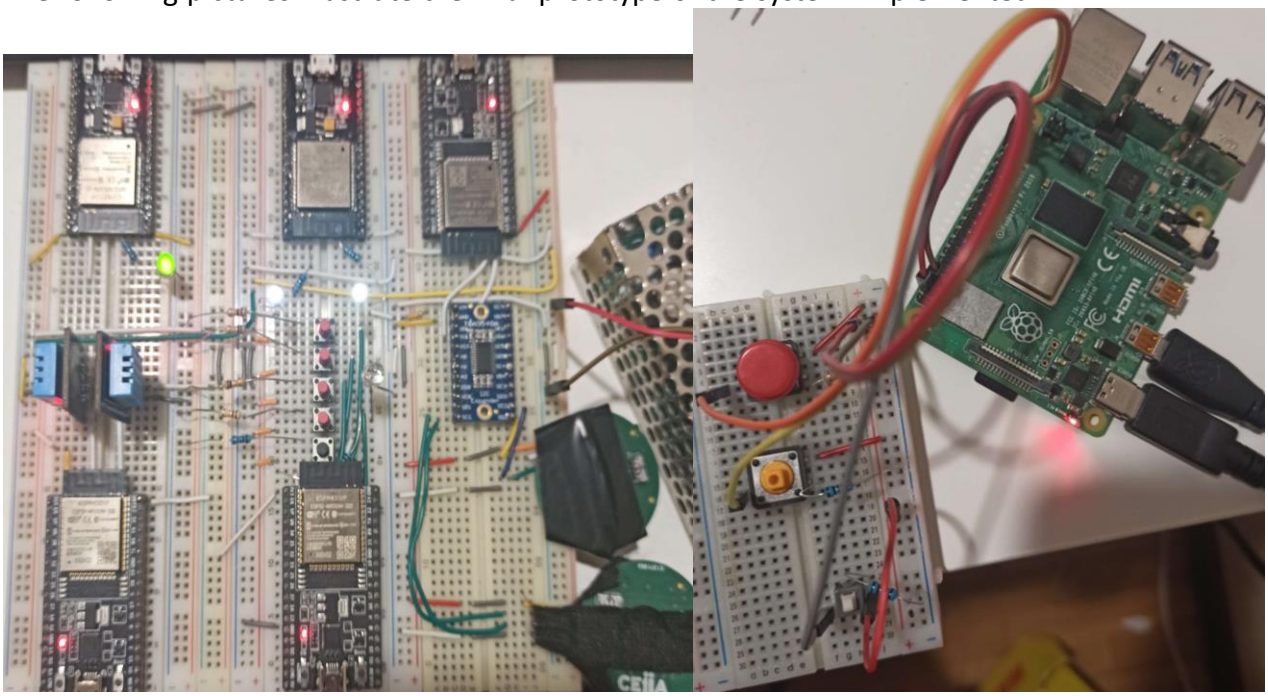
Another important milestone we reached this week is the completion and improvement of our ESP32 code. The ESP32 is the microcontroller that will be responsible for sending the values to the actuators and gathering sensor data in the system. We were able to fine-tune the code to ensure that it is fully optimized for our specific needs, and it is now able to communicate seamlessly with the Node-RED user interface and the MQTT protocol. Additionally, we have also added some extra features to the code such as the ability to handle unexpected situations, to detect and handle errors, and to improve the overall performance of the system. The ESP32 code is a crucial part of the project, and its successful implementation was a major step towards completing the project and achieving our goals.

As we mentioned previously, for our project we implemented 5 ESP32's, where each one had a specific task to handle. These were:

- ESP32 – Blinds (blinds.ino)
    - Communication with MQTT (Publish and Subscribe)
    - Reading of the values of our push buttons that simulated the blinds contact switches
    - Actuators - Simulated with and RGB led that changed color for each movement
        - Red – Open
        - Green - Close
- ESP32 – Heater (heater.ino)
    - Communication with MQTT (Publish and Subscribe)
    - Actuators – Simulated with a servo motor
- ESP32 – Lights (lights.ino)
    - Communication with MQTT (Publish and Subscribe)
    - Actuators – Simulated with white LED's
- ESP32 – Light Sensors (lightSensor.ino)
    - Communication with MQTT (Publish and Subscribe)
    - Reading of values from the VEML770 sensors
- ESP32 – Temperature Sensors (temperatureSensor.ino)
    - Communication with MQTT (Publish and Subscribe)
    - Reading of values from the DHT11 sensors

The code mentioned above will be sent as an attachment.

The following pictures illustrate the final prototype of the system implemented:



Node-RED is a powerful programming tool that allows developers to create sophisticated applications quickly and easily for the Internet of Things (IoT) and other connected systems. The platform provides a visual, drag-and-drop interface that makes it easy to wire together various devices, services, and protocols, allowing users to create complex and customized solutions with minimal coding.

In our project, we utilized Node-RED to create an HMI (Human Machine Interface) and UI (User Interface) that controlled various aspects of our system. This included the ability to monitor and control various sensors, actuators, and other devices, and to visualize and analyze data in real-time.

One of the key features of our implementation was the use of MQTT, a lightweight messaging protocol that is particularly well-suited for IoT applications. We integrated Node-RED with MQTT, which allowed for efficient and secure communication between the devices and the server. This was critical for our project, as we needed to ensure that the data being transmitted was accurate and reliable.

We also created two UI's, one of which was an HMI that was hosted on a Raspberry Pi and communicated with our ESP32 microcontroller and server. The other one was hosted on the server and communicated with the one hosted on the Raspberry Pi. This allowed us to have seamless and centralized control of our system and the ability to access it remotely.

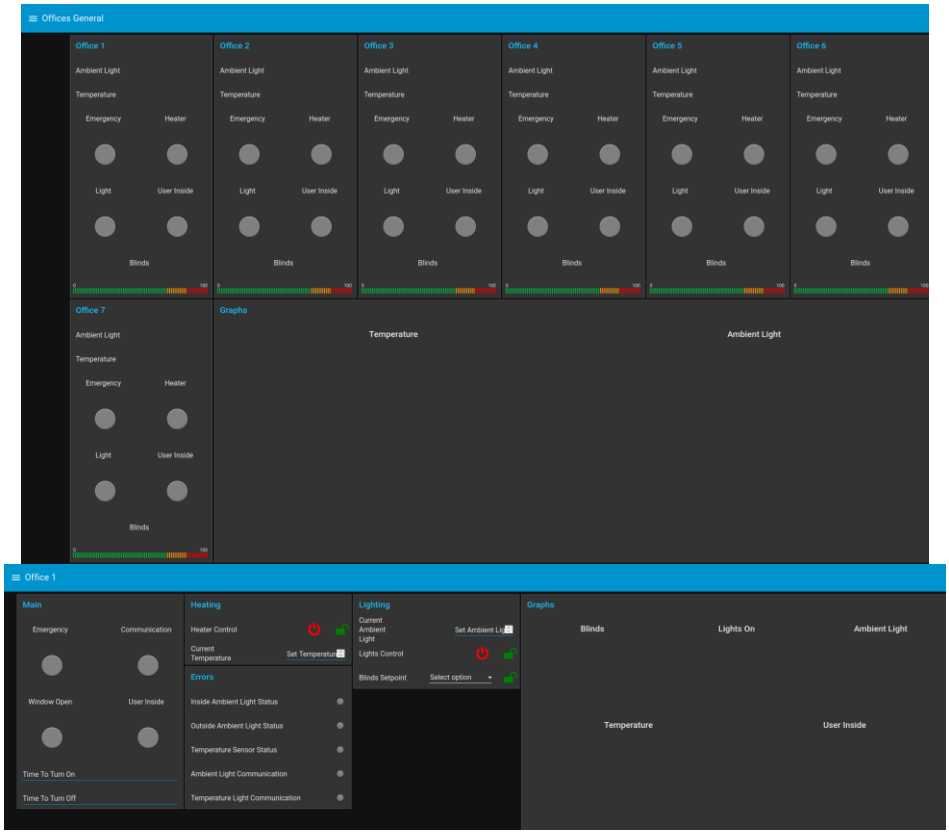
Overall, Node-RED was a key component of our project and allowed us to quickly and easily create a sophisticated and highly customizable solution that met all our requirements.

It is a great tool for prototyping and quickly tests innovative ideas without having to write a lot of code.

It is also great for small to medium size projects, as it can be easily deployed in a small device such as a Raspberry Pi and scale as the project grows.

In our UI, we could control every aspect we could on the HMI, but we also could block the editing of certain aspects, providing a secure and controlled environment for our system. This feature was particularly useful when it came to controlling our system.

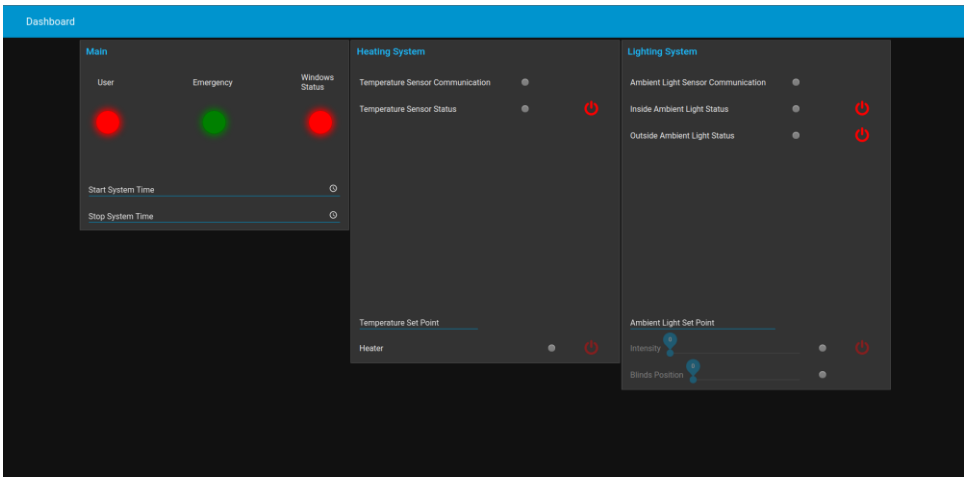
Additionally, the UI was designed with a user-friendly interface and provided real-time updates and notifications, allowing us to stay informed and make quick decisions.



In our HMI, we could control the actuators and monitor the light and temperature of our system. This included the ability to set the setpoint for temperature and light, as well as the ability to set a turn on and off time for the system. This feature allowed us to easily control the environment in our office and ensure that it was always at a comfortable level.

Furthermore, by having the HMI hosted on a Raspberry Pi, it enabled us to have real-time monitoring and control of our office's environment even if we were not physically present in the office.

The combination of the HMI and UI allowed us to have complete control and monitoring of our system, from the ability to control and monitor the environment in our office to the ability to access the system remotely and adjust as needed. Our HMI was responsible for the control of our system and was running the logic to control it on node red.



Overall, the use of Node-RED, MQTT and the HMI and UI allowed us to create a powerful and flexible system that met all our requirements and provided an elevated level of control and monitoring capabilities both for the maintenance team and the user.

We will send the UI's we created for node red attached with the name hmi.json and ui.json. To open these flows, it will be needed to install the following add-ons on the palette:

- node-red-contrib-simpletime
- node-red-contrib-ui-artless-gauge
- node-red-contrib-ui-led
- node-red-dashboard

- node-red-contrib-ui-level

4. Management and review of project

The project management for this project has been going well so far. During the first and third weeks, we split the tasks and responsibilities effectively among the team members.

Despite our best efforts, we feel that we could have benefited from an additional week to complete the project. We lost a significant amount of time on the assembly part of the project, which affected our progress. In retrospect, we feel that we aimed a bit too high and demanded too much of ourselves in such a brief time. Despite this, we have worked hard to deliver the best possible result within the given constraints, and we believe we have met most of the desired project requirements.

5. Conclusion

In conclusion, this project was a challenging one due to the tight time frame. However, the team enjoyed the experience and found it to be extremely rewarding. Despite the challenges, we were able to put in a lot of effort and dedication, resulting in a well-functioning prototype. The feeling is that we gave it our all and are proud of the work we have accomplished. Overall, it was a great learning experience, and we look forward to continuing improving our skills with projects like these.