CSLAB

ISEP - DEI - MESCC

Smart Lighting and Heating System

Group 5:

Carlos Rijo - 1101626

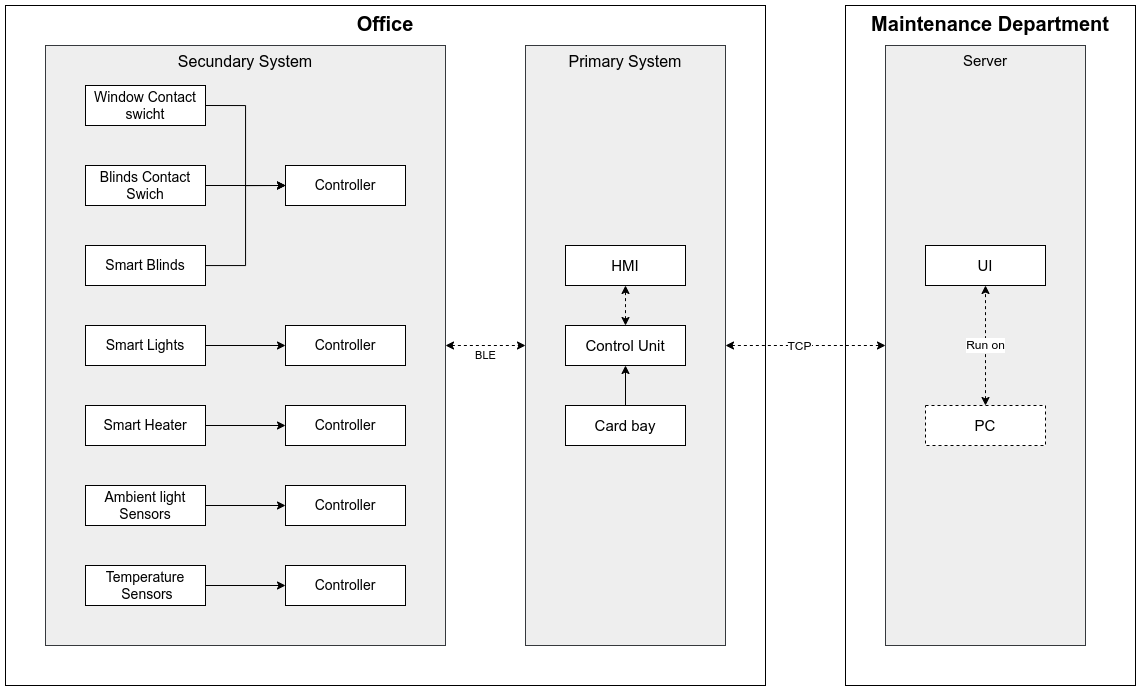
João Fernandes - 1221973

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 were 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:



MUDAR A FOTO DIAGRAM NAO E BLE

In this diagram we describe the project we will implement, 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

1. 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. These requirements include specifications such (...).

We will be 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.

Adicionar aqui os requirements de mbedder e dizer se foram met ou nao

1. 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.

* 1. 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.

* ESP32 – Blinds
  + Communication with MQTT – Priority
  + Reading of values – Priority
  + Actuators
  + (VER AS TAKS QUE CORREM NAS ESPS)
* ESP32 – Heater
  + Communication with MQTT – Priority
  + Reading of values – Priority
* ESP32 – Lights
  + Communication with MQTT – Priority
  + Reading of values – Priority
* ESP32 – Light Sensors
  + Communication with MQTT – Priority
  + Reading of values – Priority
* ESP32 – Temperature Sensors
  + Communication with MQTT – Priority
  + Reading of values – Priority

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.

* 1. 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.

n 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.

* 1. 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 RISC-V 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. This is a crucial feature for ensuring safety and security in our system. (ACABAR)

* 1. CSLAB

Our project, which is currently being 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 goal of the project 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.

Falar do codigo do node-red e completar o codigo da esp

1. Management and review of project

1. Conclusion

In conclusion, the technologies being implemented in this project are designed to support the success of an automated climate and lighting control system in a target room.

In this project, we will use sensors to monitor temperature and ambient light levels, and actuators such as a smart heater, blinds with contact switches, and LED lights. The control unit will receive input from the sensors and send commands to the actuators, and the system will have a redundancy factor in the temperature and ambient light sensors. The system can be controlled through an HMI (Human Machine Interface) touch screen and the server UI can control multiple offices. These technologies will create an automated climate and lighting control system that meets desired set points for the target room.