CSLAB

ISEP - DEI - MESCC

Smart Lighting and Heating System

WEEK 2

<u>Group 4:</u>

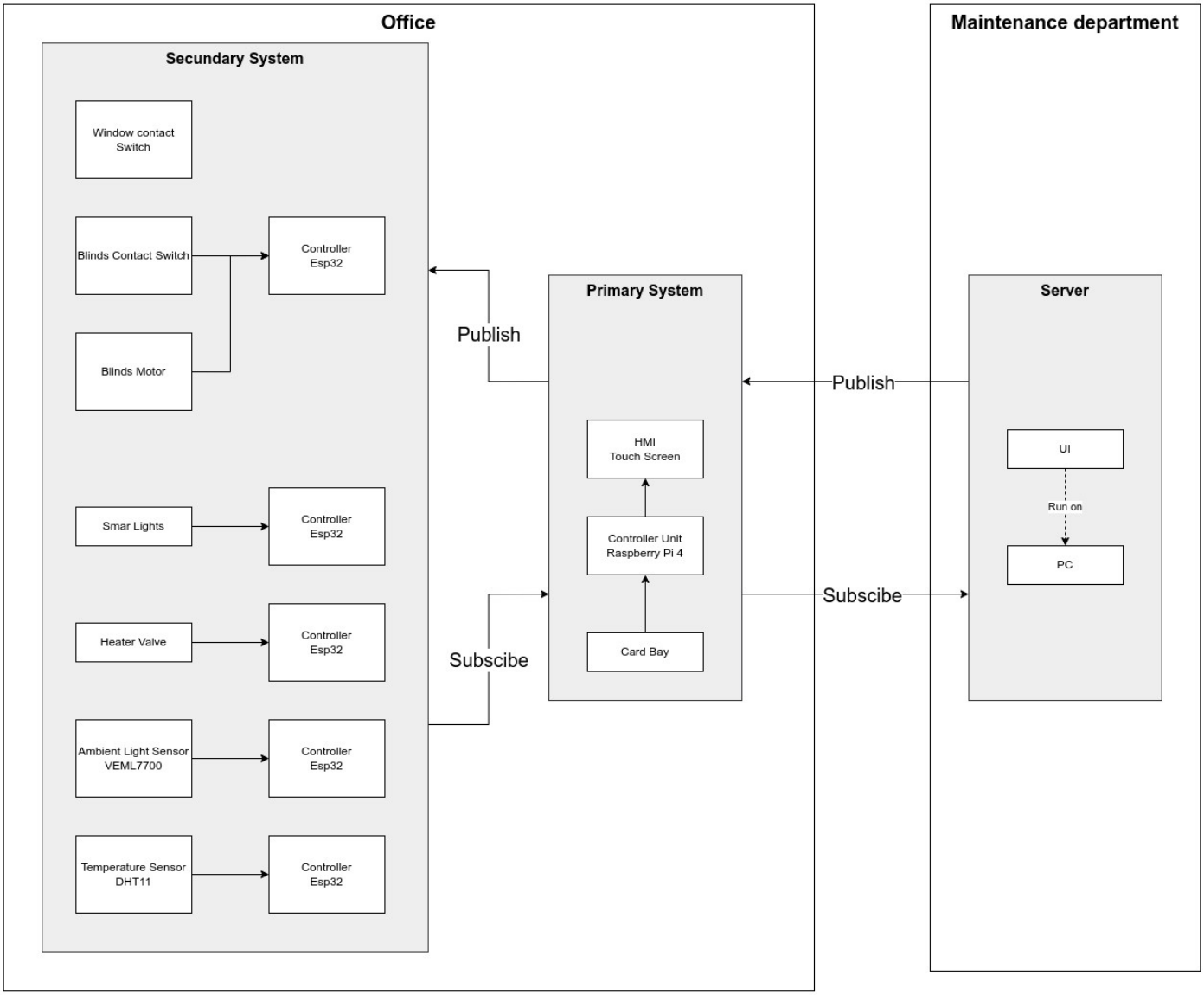
Carlos Rijo - 1101626

João Fernandes - 1221973

1. Introduction

In this document, we will discuss the technologies planned to be 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 will be used to support the success of our project. We hope that this document will serve as a useful reference for all stakeholders and help everyone to understand the technological decisions that have been 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 that we will implement, which is 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.

2. Development

In our communication system, we have decided to use the First-In-First-Out (FIFO) scheduler for task prioritization. This decision was based on the fact that our system prioritizes the importance of tasks for communication. By using the FIFO scheduler, we can ensure that the most important tasks are completed first, thus guaranteeing a more efficient communication system.

We have chosen to use FIFO over other schedulers such as Rate Monotonic and Earliest Deadline First because it aligns better

with our system's requirements. Rate Monotonic prioritizes tasks based on their periodicity, while Earliest Deadline First prioritizes tasks based on their deadline. However, in our system, the importance of tasks is determined by their relevance to communication, not their periodicity or deadline.

Additionally, using the FIFO scheduler will allow us to easily implement task prioritization on both the Raspberry Pi and the server, as we will be programming both in Python. By setting the priority values for each task, we can ensure consistency in task prioritization across both devices. This will greatly improve the overall performance and reliability of our communication system.

However, it is important to note that the FIFO scheduler does not use priority values, unlike other schedulers such as the Priority Scheduler or the Completely Fair Scheduler. In the FIFO scheduler, tasks are simply added to the queue in the order they are created and are given CPU time in the same order. This means that we cannot set specific priority levels for tasks in the FIFO scheduler. Instead, we must rely on the order in which the tasks are added to the queue to determine their priority. Despite this limitation, we have determined that the FIFO scheduler is the best fit for our system's requirements as it prioritizes the importance of tasks for communication.
We have decided to use FIFO scheduler for task prioritization and to improve performance, we are considering using CPU Affinity to assign tasks to specific cores of the CPU. This will help in preventing task contention and improve responsiveness of the system. However, we need to evaluate the impact of CPU Affinity on our system before implementation.

We plan to use threads to improve performance by executing multiple tasks simultaneously, in conjunction with FIFO scheduler and CPU Affinity. This will result in a more efficient and effective communication system. However, careful design and implementation will be needed to avoid bugs and errors due to the added complexity.

Furthermore, regarding the communication between our ESP32 and Raspberry Pi, we have decided to switch from Bluetooth Low Energy (BLE) to Transmission Control Protocol (TCP). This decision was made for several reasons, one being that both the ESP32 and Raspberry Pi will always be connected to a power source, eliminating the need for battery-saving features provided by BLE.

Another reason for this switch is to simplify the project in order to meet the deadlines set. BLE requires additional configuration and management compared to TCP, which can add complexity and development time to the project. By using TCP, we can establish a reliable and straightforward connection between the ESP32 and Raspberry Pi without the added complexity of BLE.

Additionally, TCP allows for more flexibility in terms of data transfer, as it can handle larger amounts of data and has built-in error-checking mechanisms. This ensures that the communication between the ESP32 and Raspberry Pi is stable and reliable, which is crucial for the success of our system.

Additionally, we have decided to implement the MQTT protocol in our communication system. MQTT, or Message Queuing Telemetry Transport, is a lightweight publish-subscribe based messaging protocol that is ideal for use in IoT and M2M communication. It allows for efficient and low-overhead communication between devices, making it well-suited for connecting small devices and low-power sensors over low-bandwidth networks. By using MQTT, we can further improve the performance and reliability of our system by allowing for real-time monitoring and control of the sensors and actuators on the ESP32 devices. The Raspberry Pi will act as the MQTT broker, receiving and distributing messages from the ESP32 devices to the main server. This allows us to easily collect and analyze the monitoring data from the sensors, and control the actuators on the ESP32 devices in real-time. Furthermore, MQTT's publish-subscribe model allows for easy scalability as we can add new devices and sensors to the system without having to make changes to the existing infrastructure. It also provides built-in security features such as user authentication and data encryption, ensuring that the communication between devices is secure and protected. In summary, by implementing MQTT in our communication system, we are able to improve the performance, reliability, scalability and security of our system, and to collect and analyze the monitoring data from the sensors and control the actuators on the ESP32 devices in real-time. The broker will be hosted on the Raspberry Pi and the main server will be hosted on a PC.

Our decision to switch from Bluetooth Low Energy (BLE) to Transmission Control Protocol (TCP) and implement the MQTT protocol in our communication system was driven by the need to improve the performance, reliability, scalability and security of our system. Switching from BLE to TCP simplifies the project and improves the stability and reliability of the communication between the ESP32 and Raspberry Pi. By using MQTT, we are able to improve the performance and reliability of our system by allowing for real-time monitoring and control of the sensors and actuators on the ESP32 devices, as well as providing built-in security features. The Raspberry Pi acts as the MQTT broker, receiving and distributing messages from the ESP32 devices to the main server, while the main server is hosted on a PC. These changes will improve the overall functionality of our communication system and meet our project deadlines.

The hardware we will use in our project will be the following

- Server
  - PC
- Primary System
  - Raspberry PI 4
  - Touch Screen
- Secondary System
  - Esp32-wroom-32d
  - Dht11 (temperature sensor)
  - VEML7700 (light sensor)
  - Contact Switches (simulated by DipSwitches)
  - Blinds (simulated by 2 LEDs)
  - Lights (simulated by LEDs)
  - Heater (simulated by 2 LEDs)