U. PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Embedded Systems

Automatic Lights

Leonardo Araújo Freitas up202400832
Matilde Isabel da Silva Simões up202108782

May 2025

Contents

1 Introduction

This report presents the specification and implementation of the Smart Lighting System developed within the scope of the Embedded Systems course. The main objective of the project is to design and implement an automated lighting system that reacts to user presence and ambient light conditions.

The system is built upon the integration of three hardware platforms – Android, Raspberry Pi and Arduino.

This report provides a comprehensive overview of the system's functional and non-functional requirements, architectural design, hardware and software prototype development, integration strategy and evaluation. The project demonstrates the successful combination of multiple technologies and communication protocols, resulting in a realistic and extensible embedded system for smart automation.

2 Requirements Analysis

This section outlines the functional and nonfunctional requirements of the Smart Room Lighting System. The requirements ensure that the system responds appropriately to user presence, environmental conditions and remote commands, providing a responsive smart lighting experience.

2.1 Functional Requirements

- The system shall turn on the room light when an user is detected entering the room.
- The system shall turn off the light when all users have left the room.
- The system shall support the detection of multiple users.
- The system shall determine when each known user enters or exits the room.
- The Android phone shall display the status of the room light in real time.
- It shall be possible to turn the room light on or off from the Android phone.
- The system shall always accept and execute light control commands sent from the Android device, regardless of environmental conditions or sensor inputs.
- A personalized welcome message shall be displayed for a detected user upon entering the room.
- User names used for the welcome message shall be defined on the Android device.
- The light shall not be turned on if the room's ambient luminosity is above a defined threshold.
- Lights shall only be turned on after the door has been opened.
- Lights shall only be turned off after the door has been closed.
- Lights may be turned on before the user physically enters the room.
- Lights shall only be turned off after the user has completely exited the room.

2.2 Nonfunctional Requirements

- User detection shall occur within 5 seconds.
- The remote light control actions from the Android app shall be executed within 4 seconds of pressing the button.
- The Android app shall display the current light status within 4 seconds of any state change.

3 Modeling

3.1 Actor Model

The Smart Lighting System can be described through the actor model present in Fig.1. This model allows us to visualize the logical components of the system and the interactions between them through the exchange of messages, providing a high-level abstraction of the system architecture.

The core of the intelligent room control system is the Room Controller, which manages all system functionality and handles communication with two essential components: the Light Controller, responsible for controlling the physical lighting system; and the Mobile Interface, an Android application that allows users to check the status of the lights and send remote commands.

The Room Controller integrates internal logic for user detection, ambient light sensing and decision-making. When a known user enters the room, the Room Controller detects their presence, verifies the ambient brightness level, and, if it is below a defined threshold, sends a message to the Light Controller to turn on the lights. Simultaneously, it updates the Mobile Interface with the light status and displays a personalized welcome message for the user.

This actor-based architecture ensures modular communication between components through message passing, enabling efficient and responsive smart lighting behavior.

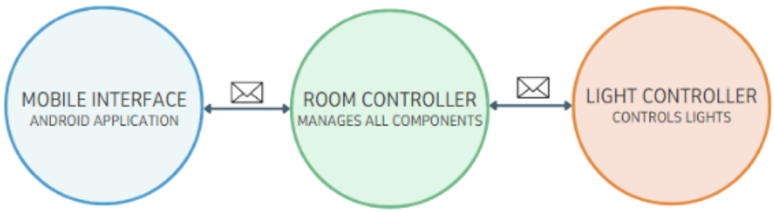


Figure 1: Actor Model Diagram

3.2 State Machine

This section presents the state machine diagram that models the behavior of the Smart Lighting System. The system transitions between two main states: `light_off` and `light_on`, based on a combination of inputs such as door state, user presence, luminosity and remote commands.

The definition follows the standard tuple `(States, Inputs, Outputs, UpdateFunction, InitialState)`, which is detailed below. The transition logic is determined by a priority rule, where the remote command always overrides any contextual or environmental conditions. That is, if the user manually turns the light on or off via the mobile application, the system immediately follows the instruction, regardless of whether someone is present in the room.

When no remote command is issued, the system reacts automatically based on contextual inputs. Among the inputs, those marked as pure — `door_state`, `user_detected` and `remote_command` — represent discrete signals whose value at a given instant fully defines their meaning and they have no intrinsic value beyond their current state.

This reactive behavior is defined as follows:

- The light is turned on when:
  - the door is open,
  - the luminosity is below a defined threshold,
  - and a known user is detected inside the room.
- The light is turned off when:
  - the door is closed,
  - and no known users are detected in the room.

The Smart Lighting System is designed as an event-triggered model, where the control logic reacts to discrete changes in environmental conditions or user interactions. This makes the system reactive and efficient, only processing updates when necessary.

The semantics are formally presented below and are used to derive the state diagram in Fig.13 in Subsec. 9.1.

4 System Architecture

This section describes the system architecture of the Smart Lighting System, encompassing both the hardware and software perspectives. The system architecture was designed with modularity, maintainability and scalability in mind. By separating sensing, decision-making and user interaction into distinct modules, the system facilitates development, debugging and potential future extensions.

4.1 Software Architecture

The Smart Lighting System follows a modular and service-oriented software architecture. The system is composed of three main software modules: the Light Controller, the Light Service and the Lightning App. Each module is responsible for a distinct set of functionalities and communicates with the others through message passing over the network.

- Light Controller: Handles sensor data acquisition and lighting actuation. It collects input from distance and luminosity sensors and controls the light relay based on received commands. Communication with the Light Service is established via the ESPHome API (TCP protocol) [3], which allows the service to read sensor values and issue light control instructions. It also manages connectivity and supports integration with higher-level automation logic.
- Light Service: Acts as the central coordination module of the system. It gathers environmental data from the Light Controller through the ESPHome API (TCP protocol), applies rule-based decision logic and determines whether the light should be turned on or off. Additionally, it exposes a REST API (HTTP protocol) [7] that enables external clients to interact with the system – including querying the current state, toggling the light, switching modes and accessing sensor readings.
- Lightning App: Serves as the user interface layer, implemented using the Home Assistant mobile application. It communicates with the Light Service through the REST API (HTTP protocol), enabling real-time monitoring, manual control of the light and reception of push notifications. The app also supports user authentication and provides a centralized dashboard for managing lighting preferences and automation behavior.

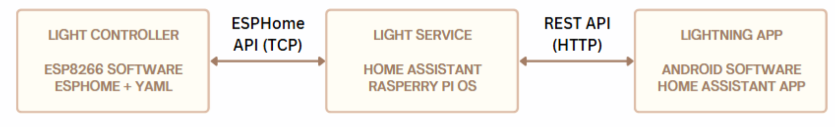


Figure 2: Software Architecture Communication

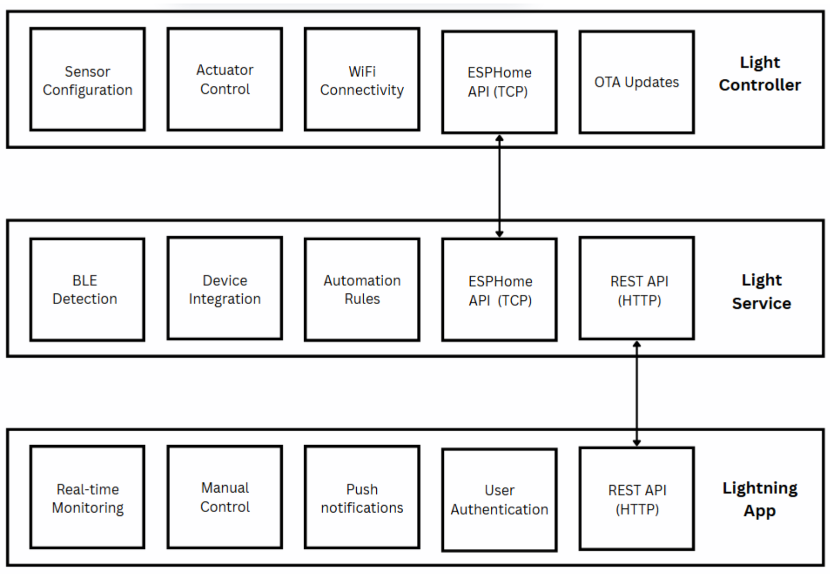


Figure 3: Software Architecture Diagram

The deployment diagram in Fig. 4 illustrates how software responsibilities are distributed across physical devices. It shows the allocation of core functionalities to each component in the system:

- The Arduino is responsible for interacting with the physical environment, acting as a Sensor Controller and Light Manager – Light Controller.
- The Raspberry Pi hosts the application and manages the system logic, serving as the central coordinator – Light Service.

- The Android device provides an user interface, allowing interaction with the system through commands and feedback – Lightning App.

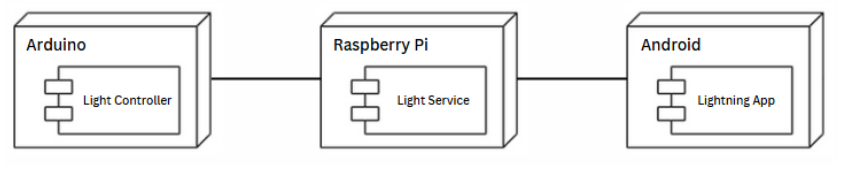


Figure 4: Deployment Diagram

4.2 Hardware Architecture

The hardware architecture diagram identifies three components: Arduino, Raspberry Pi and Android.

Communication between the Arduino and the Raspberry Pi is bidirectional and occurs exclusively over a Wi-Fi channel, with no physical connections between the two devices. Similarly, the Raspberry Pi and the Android device also communicate bidirectionally over the same wireless network.

This architecture is scalable, allowing the addition of multiple Arduinos and Android devices to the system. Each Arduino can be configured as an additional node for lighting monitoring and control in other rooms, while each Android device represents an user interface that can independently interact with the Raspberry Pi, enabling personalized control and status updates per user.

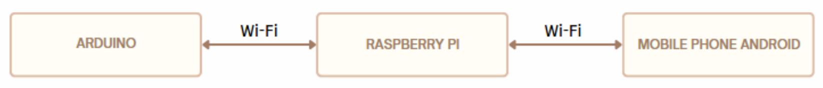


Figure 5: Hardware Architecture Communication

To complement the deployment diagram (Fig. 4), the following figure provides a more detailed view of the system's physical architecture. It illustrates the concrete hardware components involved in the Smart Lighting System, as well as some data flow between them. Each block corresponds to a physical device previously shown in the deployment diagram, now expanded to show its internal modules and how they interact.

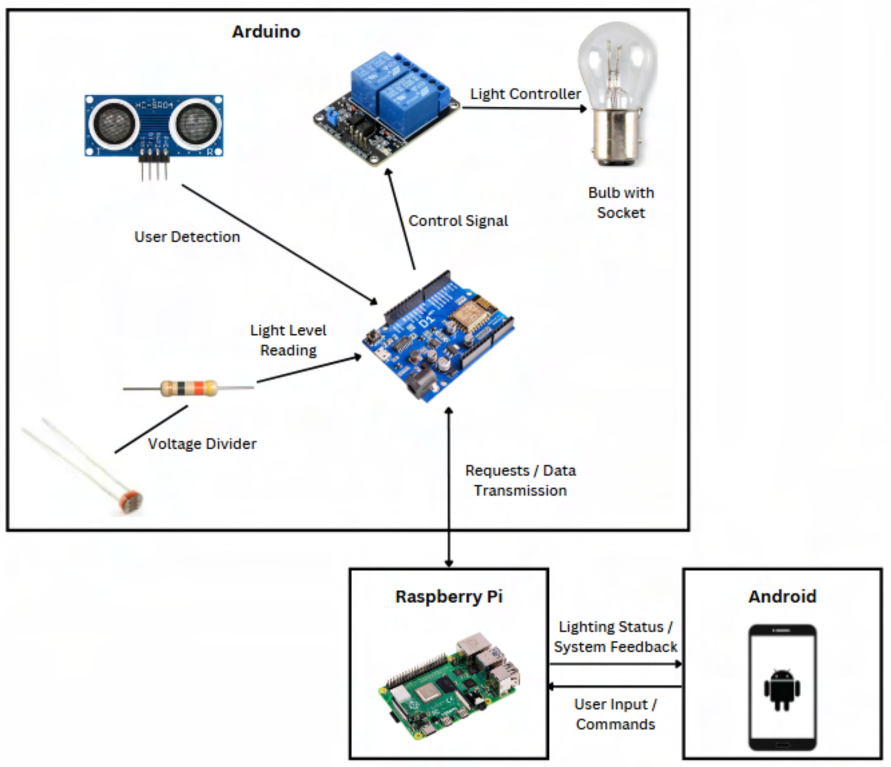


Figure 6: System-Level Hardware Block Diagram

5 Prototype

This section provides a comprehensive overview of the developed prototype, including both hardware and software components that compose the Smart Lighting System.

The hardware prototype includes all physical components required for sensing, control and processing, such as the ESP8266 microcontroller, sensors, relays and the Raspberry Pi. The software prototype integrates ESPHome, Home Assistant OS and the Home Assistant Companion App to manage automation logic, user interaction and device communication.

5.1 Hardware Prototype

The following subsection outlines the hardware configuration implemented in the Smart Lighting System. It details the selected hardware components, the structure of the prototype and the strategy used to support sensing, actuation and communication.

5.1.1 ESP8266-Based Design Decision

The materials provided for the project included an Arduino Uno (Fig. 15a) and an ESP8266 module (Fig. 15b), rather than an Arduino with a Wi-Fi board, as expected. After a detailed analysis of the components available to us, we concluded that the ESP8266 alone could fulfill all required roles, handling both sensing/actuation and wireless communication. Although the initial architecture included the Arduino Uno as the central microcontroller, the final prototype uses only the ESP8266 module. From an embedded system's perspective, this decision offers numerous advantages. The ESP8266 integrates both processing capabilities and Wi-Fi connectivity within a single compact chip, eliminating the need for external communication modules and significantly reducing hardware complexity, physical footprint and wiring. This not only simplifies the overall design but also lowers power consumption and implementation cost. Full compatibility with the ESPHome framework further enhances development efficiency by allowing rapid prototyping through declarative YAML configuration and seamless integration with Home Assistant. By consolidating functionality into a single programmable device, the system benefits from lower latency, higher reliability, simplified maintenance and better scalability – key attributes in the design of efficient and robust embedded systems. For this reason, the ESP takes on the role of the Arduino in our project.

5.1.2 Component Overview and Physical Integration

The list of hardware components used in the development of the Smart Lighting System is shown in Fig. 15 in Subsec. 9.3.

The core processing unit is the Raspberry Pi (15c), which communicates with the ESP8266 (15b).

A Raspberry Pi 4 (15c) acts as the central coordinator of the system, receiving sensor data and executing logic based on defined conditions. It also hosts the interface application that communicates with the Android device (15d), which serves as the user's control panel, allowing light status visualization and remote control actions.

To detect user presence, an ultrasonic sensor HC-SR04 (15e) is connected to the ESP8266. Ambient light levels are measured using a Light Diode Resistor (LDR) (15g) configured in a voltage divider circuit with a resistor (15i), allowing the ESP8266 to read analog luminosity values. These components are connected through a Breadboard (15k) using Jumper Wires (15j) to simplify prototyping and ensure flexible reconfiguration.

Lighting control is achieved via a Relay Shield Module (15f) that receives digital output signals from the ESP and switches the Light Bulb (15h) on or off. The bulb is installed in a Socket Light holder (15l) and powered externally.

Further implementation details of the physical connections, including exact pin mappings and wiring logic, are described in Subsec. 9.4, which illustrates how all components are integrated to form a fully functional system.

5.2 Software Prototype

The software prototype of the Smart Lighting System is developed using three primary platforms: the ESP8266 microcontroller, the Raspberry Pi with Home Assistant Operating System and an Android platform. The complete source code for all components is available in the zip file provided.

5.2.1 ESP8266 Software

Physical sensors and actuators are handled by an ESP8266 microcontroller [2] and ESPHome with YAML configuration for a simplified development experience that provides declarative definition of components in the system.

The system includes the configuration of two sensors: an ultrasonic sensor (HC-SR04) that measures the distance to detect presence at the door and a light-dependent resistor (LDR) that monitors the room's light level. Based on the data collected, the system controls a relay connected to a room light, defined as a binary light in ESPHome.

Additionally, a manual switch *Auto Light Control* allows enabling or disabling the automatic lighting behavior. This setup enables the light to turn on or off automatically, depending on presence and ambient light. A detailed description of the ESP configuration can be found in Subsec. 9.2.

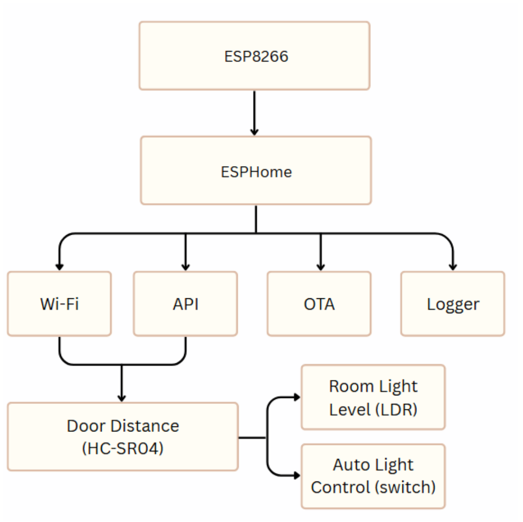


Figure 7: ESP8266 flowchart

The complete ESPHome configuration file can be found in the accompanying zip file under `esphome-web-0451d2.yaml`.

5.2.2 Raspberry Pi Software with Home Assistant OS

The Raspberry Pi runs Home Assistant OS as the central intelligence of the system. Home Assistant provides a comprehensive platform for integrating the ESP8266 sensors and implementing automation rules.

Key software components include:

- Home Assistant Integration: Automatic discovery and integration of the ESPHome device with entity configurations. Sensor data and actuator control are handled through support for ESPHome, allowing seamless communication and control.

- Automation Rules: Rule-based logic that controls the light based on door distance, ambient light level and operational mode. These rules are defined in YAML within the Home Assistant interface and can respond to changes in any registered entity.

  Rule: Light off to on

  This automation (`automation_rule_OFF2ON`) automatically turns on the room light and sends a welcome message when a user enters the room. It is triggered when a sensor value drops below 0.3 (door distance). The conditions check if at least one of the Bluetooth signal strengths is stronger than -85 dBm, meaning a user is nearby.

  If the system voltage is also below 0.05 (luminosity of the room), the light is turned on. Additionally, when a known user is detected entering the room, the system identifies which user has been recognized and sends a personalized "Welcome Home!" notification with the user's name. Even if the ambient luminosity is above the defined threshold — meaning that the room already has sufficient light — the notification is still sent to acknowledge the user's presence upon entry.

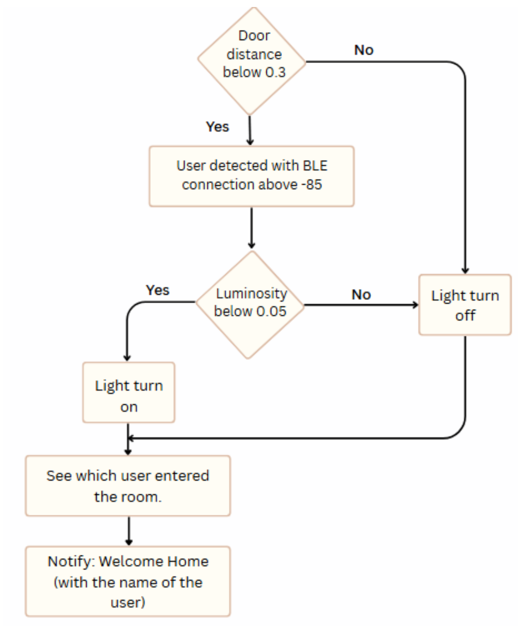

Door distance below 0.3
No
Yes
User detected with BLE connection above -85
Yes
Luminosity below 0.05
No
Light turn off
Light turn on
See which user entered the room.
Notify: Welcome Home (with the name of the user)

Figure 8: Flowchart for the Automation Rule to turn on the light.

Rule: Light on to off

This automation (`automation_rule_ON2OFF`) automatically turns off the room light when no users are detected in the room. It is triggered when a sensor value rises above 0.9 (door distance). The action only proceeds if the Bluetooth signal strength from both devices is below -85 dBm, suggesting the users have left the area. Once these conditions are met, the configured light is turned off.

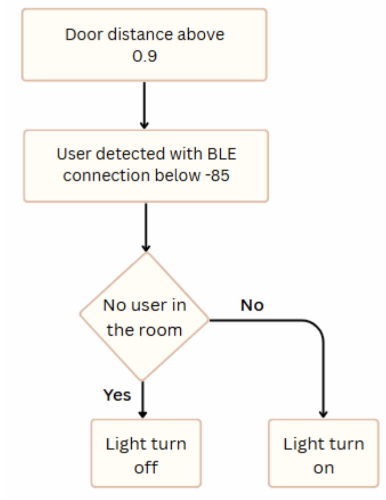
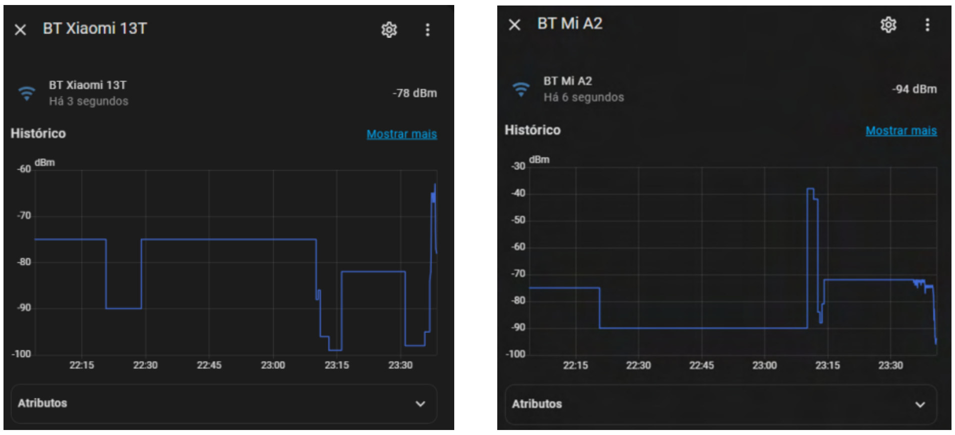


Figure 9: Flowchart for the Automation Rule to turn off the light.

- Bluetooth Integration: A Python (v.3.10.11) script performs continuous Bluetooth Low Energy (BLE) scanning using the *bleak* library [1]. The scanner identifies nearby devices by parsing manufacturer-specific advertising packets and extracting their UUIDs. Each device's signal strength (RSSI) is classified (Excellent, Fair, Weak) and matched against a list of known UUIDs. If a known user is detected, the script sends this data to Home Assistant via its REST API, dynamically updating virtual sensor entities with attributes such as device name, UUID, MAC address, signal strength and quality. These entities are then used in automation rules to trigger context-aware actions based on user presence. The scanner supports both one-time and continuous detection modes and logs all detections locally for monitoring and debugging. The complete script is included in the zip file under `/BLUESCAN/uuid_scanner.py`.



Reasonable signal with variations. In the last minutes, it approached **-78 dBm**, indicating *close presence.*

More unstable signal. It had a strong peak, but is currently at **-94 dBm**, indicating *absence or distance.*

Figure 10: RSSI signal strength history of the two Bluetooth devices monitored via Home Assistant.

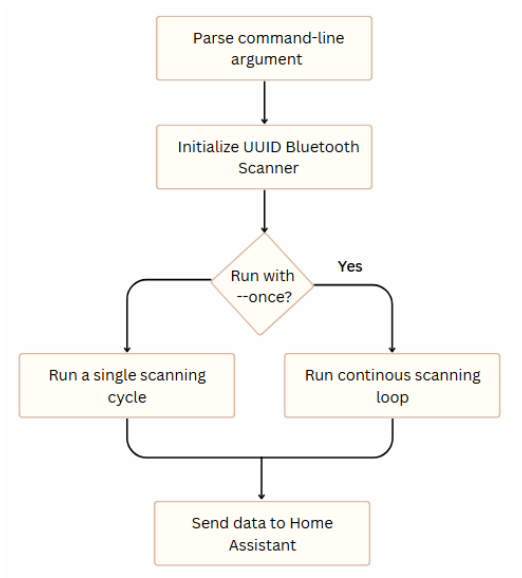
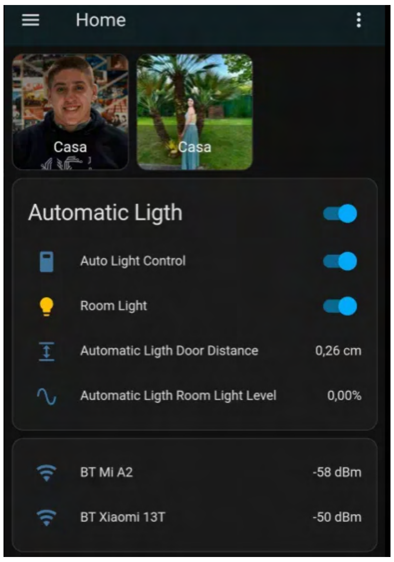

Parse command-line argument
Initialize UUID Bluetooth Scanner
Run with --once?
Yes
Run a single scanning cycle
Run continous scanning loop
Send data to Home Assistant

Figure 11: Flowchart of the Bluetooth Scanning Script Running on the Raspberry Pi

5.2.3 Android Software

The Android component utilizes the official Home Assistant Companion App, which provides a comprehensive mobile interface for monitoring and controlling the system. Through this application, users can access a customized dashboard – built using the Home Assistant interface – that includes buttons for light control, sensor data visualization and activity monitoring. The app allows users to obtain real-time information about room occupancy and light status, and to manually override lighting controls when necessary. Additionally, it supports push notifications to alert users about relevant system events and includes built-in user authentication mechanisms to ensure secure and personalized access to the home automation environment.



Automatic Ligth

Auto Light Control	
Room Light	
Automatic Ligth Door Distance	0,26 cm
Automatic Ligth Room Light Level	0,00%
BT Mi A2	-58 dBm
BT Xiaomi 13T	-50 dBm

Figure 12: Dashboard of the application

6 Integration

A demonstration video has been recorded to complement this report. It illustrates the assembly of the hardware components as well as the preparation and execution of the test scenarios. The video can be accessed at the following link: `https://www.youtube.com/watch?v=Tu863c25fHs`.

6.1 Installation

The initial step involved setting up ESPHome and preparing the development environment for the ESP8266, following the official documentation [2] [4]. The configuration was uploaded directly to the ESP8266 using the ESPHome web interface.

Subsequently, the operating system for the Raspberry Pi was installed with the help of the Raspberry Pi Imager tool [9]. This software facilitates the selection of the target SD card and streamlines the installation process. It also enables Wi-Fi configuration, which is crucial for establishing remote communication. Further details regarding this setup can be found in [8].

Home Assistant was deployed on the Raspberry Pi via Docker, using the provided `docker-compose.yml` file to define the container's configuration [5]. Following this, the official Home Assistant mobile application [6] was installed on an Android device.

7 Evaluation

This section presents the validation process of the Smart Lighting System. The evaluation was designed to verify compliance with both functional and non-functional requirements defined in Sec.2. A combination of automated tests, manual observations and Home Assistant log analyses was used to assess the system's behavior in realistic scenarios.

7.1 Functional Testing

To validate the functional behavior of the system, a test plan was executed covering the 14 functional requirements. Each test involved reproducing the triggering conditions and observing whether the system responded appropriately.

Test Description	Passed
Light turns on when user enters	Yes
Light turns off when no users are present	Yes
Multiple user detection supported	Yes
Entry and exit of known users detected	No (*)
Android shows light status in real-time	Yes
Light can be controlled from Android	Yes
Remote command overrides sensors	Yes
Welcome message shown on user entry	Yes
Usernames configurable on Android	Yes
Light does not turn on in high luminosity	Yes
Light only turns on after door opens	Yes
Light only turns off after door closes	Yes
Light may turn on before user fully enters	Yes
Light only turns off after user fully exits	Yes

Table 1: Functional Requirement Validation Summary

Note (*): The system's detection of user entry and exit was partially unreliable due to limitations in Bluetooth-based indoor localization. Although BLE scanning with UUID matching was employed, the initial detection latency varied between 2–10 seconds – exceeding the ideal reaction time. However, once a user was initially detected, subsequent interactions and system responses occurred within the expected time frame. Therefore, this requirement is marked as partially fulfilled, depending on the detection conditions.

7.2 User Presence Detection Limitations

We experimented with several strategies to improve user detection accuracy:

- BLE UUID Matching (Implemented): Using a Python scanner script with the `bleak` library to match user UUIDs and RSSI signal strength. Detection worked reliably but suffered from occasional delay due to BLE advertisement latency and background scanning policies on mobile devices.
- Wi-Fi Signal Strength (Tested): Using the device's MAC address to estimate proximity via RSSI over Wi-Fi. However, this method lacked precision and was subject to interference from other devices.
- GPS-based Detection (Rejected): Considered infeasible for indoor accuracy on mobile devices.

To further improve the accuracy of user detection, we conducted a market search for alternative presence sensors that could enhance our system's performance. One of the most promising options identified was the Hi-Link HLK-LD2450, a 24 GHz millimeter wave radar sensor capable of detecting the position and trajectory of moving persons in real time. Unlike BLE or Wi-Fi-based methods, which rely on the user carrying a mobile device, the HLK-LD2450 allows for non-intrusive presence detection by emitting and interpreting radio frequency signals. Its radar technology enables more accurate position tracking, supporting multiple target detection within a defined area.

The user detection component proved to be the most challenging aspect of the entire project. Beyond its technical complexity, it became a significant learning experience that exposed us to the real-world limitations of indoor localization technologies. Throughout the development process, we tested multiple strategies and explored alternative approaches, ultimately becoming aware of how difficult it is to achieve accurate, real-time presence detection in indoor environments. This is a well-known and actively researched problem, with ongoing debates in both academic and industrial contexts. As such, the experience highlighted not only the technical depth of the challenge but also the importance of balancing feasibility, cost, privacy and precision in embedded system design.

7.3 Non-Functional Testing

The performance aspects of the system were assessed via manual timing and system logs. The results are summarized in table below.

Test Description	Result
User detection occurs within 5 seconds	Average: 2–10 seconds
Light control from Android within 4 seconds	2.1–2.7 seconds
Light status shown on Android within 4 seconds	1.9–2.3 seconds

Table 2: Non-Functional Requirement Validation

While remote control responsiveness met expectations, BLE detection delay exceeded the target. Nonetheless, user experience remained acceptable for non-critical lighting automation.

7.4 System Logs and Observations

The system logs recorded by Home Assistant and the BLE scanner script provided further evidence of correct operation. These logs were essential in confirming rule-based transitions, such as light on/off conditions, measuring the response time between commands issued by the Android application, the resulting status updates and analyzing the classification of RSSI signal strength as well as the rates of device detection.

The Smart Lighting System successfully satisfied the majority of its functional and non-functional goals. The main limitation lies in indoor user detection latency, which remains a technical bottleneck. Nevertheless, the system is functional, extensible and aligns well with embedded systems design principles.

8 Conclusion

The Smart Lighting System successfully met the defined objectives and fulfilled the functional and non-functional requirements outlined at the beginning of the project. The integration of the ESP8266, Raspberry Pi and Home Assistant enabled the implementation of a modular, scalable and responsive automation solution. Despite the overall success, the project faced challenges, particularly in achieving reliable and real-time user presence detection using Bluetooth Low Energy (BLE). In response, alternative technologies were explored and suggested as potential improvements to enhance detection accuracy and robustness. These insights not only addressed the limitations encountered but also opened new perspectives for future enhancements, reinforcing the system's adaptability and alignment with embedded systems design principles.

References

[1] Bleak Developers. *Bleak: Bluetooth Low Energy platform Agnostic Client Framework.* `https://bleak.readthedocs.io/en/latest/`.

[2] ESP8266 Arduino Core Developers. *ESP8266 Arduino Core Documentation.* `https://arduino-esp8266.readthedocs.io/en/latest/`.

[3] ESPHome. *ESPHome API Documentation.* URL: `https://esphome.io/components/api.html`.

[4] Home Assistant Developers. *ESPHome Integration Documentation.* `https://www.home-assistant.io/integrations/esphome`.

[5] Home Assistant Developers. *Home Assistant Installation Guide for Other Linux Systems.* `https://www.home-assistant.io/installation/raspberrypi-other/`. 2025.

[6] Home Assistant Developers. *Home Assistant Mobile App Integration.* `https://www.home-assistant.io/integrations/mobile_app/`.

[7] Home Assistant Developers. *Home Assistant REST API Documentation.* `https://developers.home-assistant.io/docs/api/rest/`.

[8] Raspberry Pi Foundation. *Raspberry Pi Configuration Guide.* `https://www.raspberrypi.com/documentation/computers/configuration.html`.

[9] Raspberry Pi Foundation. *Raspberry Pi Imager.* `https://www.raspberrypi.com/software/`.

9 Appendix

9.1 Formal State Machine Model

- States:

$$States = \{light\_off,\ light\_on\}$$

- Inputs:
$$Inputs = \{door\_state, user\_detected, luminosity, remote\_command\}$$

$$\begin{aligned}
door\_state &\in \{\text{present}, \ \text{absent}\} : pure \\
user\_detected &\in \{\text{present}, \ \text{absent}\} : pure \\
luminosity &\in \mathbb{N} \\
remote\_command &\in \{\text{present}, \ \text{absent}\} : pure
\end{aligned}$$

- Outputs:
$$Outputs = \{present, \ absent\} : pure$$

- Update Function:
$$Update : States \times Inputs \to States \times Outputs$$

- Initial State:
$$InitialState = light\_off$$

Update Function Rules:

$$\text{update}(s, i) = \begin{cases} (light\_on, \ present) & \text{if } i(door\_state) = \text{present} \\ & \quad \wedge \ i(luminosity) < \text{threshold} \wedge i(user\_detected) = \text{present} \\ (light\_on, \ present) & \text{if } i(remote\_command) = \text{present} \\ (light\_off, \ absent) & \text{if } i(remote\_command) = \text{absent} \\ (light\_off, \ absent) & \text{if } i(door\_state) = \text{absent} \wedge i(user\_detected) = \text{absent} \\ (s, \ s(i)) & \text{otherwise} \end{cases}$$

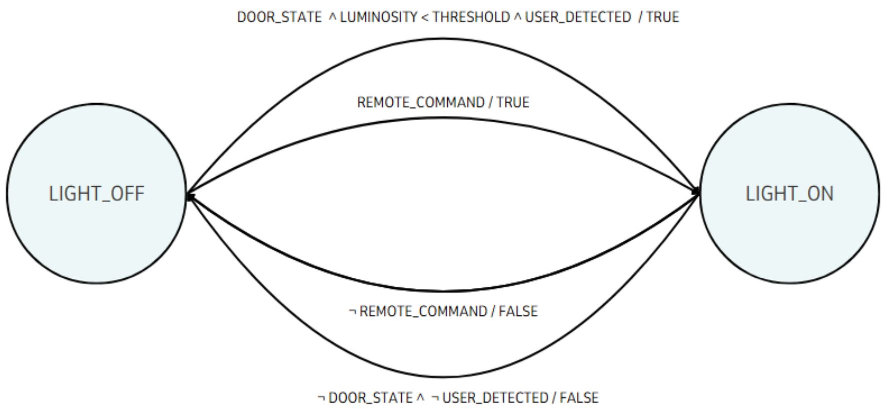

DOOR_STATE ^ LUMINOSITY < THRESHOLD ^ USER_DETECTED / TRUE
REMOTE_COMMAND / TRUE
LIGHT_OFF
LIGHT_ON
¬ REMOTE_COMMAND / FALSE
¬ DOOR_STATE ^ ¬ USER_DETECTED / FALSE

Figure 13: State Machine Diagram

9.2 ESPHome Configuration in Detail

The ESPHome configuration defines three main aspects of the system:

- Basic Configuration: The ESP8266 microcontroller was configured and integrated into the network with support for both standard Wi-Fi connection and fallback Access Point (AP) mode. The YAML configuration includes the definition of a device name, as well as activation of essential services such as the `logger` for diagnostics, the `api` for communication with Home Assistant and `ota` to allow Over-The-Air firmware updates. These components enable seamless interaction with the Home Assistant platform and allow remote updates and debugging without the need for physical access.

- Sensor Configuration: Two sensors were defined to provide environmental context. First, an HC-SR04 ultrasonic sensor connected to GPIO5 (trigger) and GPIO4 (echo) continuously measures the distance to the door, with a 1-second update interval. This enables real-time detection of door states and user entry events. Second, an analog Light Dependent Resistor (LDR) connected to pin A0 measures ambient luminosity. The raw ADC reading is normalized to a 0–100% scale via a custom lambda function, with data refreshed every 5 seconds. This sensor allows the system to make informed decisions about whether artificial lighting is necessary based on current light levels.

- Actuator Configuration: The lighting control system is implemented using a relay module driven by GPIO14. This output is abstracted through a binary light component named `Room Light`, which can be toggled on or off through Home Assistant or automation rules. Additionally, a template switch named `Auto Light Control` is defined, allowing users to enable or disable automatic light behavior. This switch is marked as `optimistic` and set to `RESTORE_DEFAULT_OFF`, ensuring predictable behavior after a power cycle. These configurations allow both manual and automated control of the lighting system.

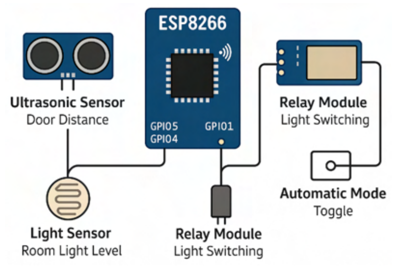


Figure 14: Pinout and software mapping of the ESP8266 configuration as defined in the ESPHome YAML file.

9.3 Hardware Components

(a) Arduino Uno Rev 3

(b) ESP8266 module

(c) Raspberry Pi 4

(d) Android

(e) Ultrasonic sensor HC-SR04

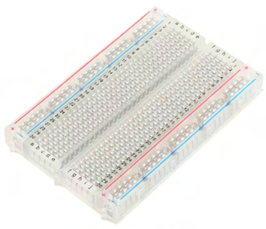
(f) Relay Shield Module

(g) Light Diode Resistor

(h) Light Bulb

(i) Resistor

(j) Jumper Wires


(k) Breadboard

(l) Socket Light

Figure 15: Hardware Components Used

9.4 Physical Connections

The following is how the physical connections between components are put into practice.

The main microcontroller is an Arduino Uno Rev 3, to which an ESP8266 WiFi-Board is linked to allow wireless contact with the Raspberry Pi 4. The WiFi Bridge code specifies that the ESP8266 module operates at a 9600 baud rate and is connected to the Arduino via serial communication (TX/RX pins). Four pins on the Arduino are used to connect the HC-SR04 Ultrasonic Sensor in order to detect the presence of an user:

- VCC is connected to the 5V power source of the Arduino.
- The Arduino's ground (GND) is connected to GND.
- TRIG sends the ultrasonic pulse by connecting to digital pin 9.
- To receive the echo signal, the ECHO is connected to digital pin 10.

A Light Dependent Resistor (LDR) in a voltage divider design makes up the ambient light sensor circuit:

- The Arduino's 5V power supply is connected to one of the LDR's terminals.
- Concurrently, the other terminal is connected to:
  - Analog pin A0 on the Arduino is used to read the light level.
  - GND is connected to a 10kΩ pull-down resistor.

The Relay Shield Module links to the Arduino and regulates the actual lighting using:

- VCC connected to Arduino's 5V power.
- GND connected to Arduino's ground.
- IN1 signal pin connected to Arduino's digital pin 7 for lamp control.
- The lamp circuit connects to the relay's NO (Normally Open) and COM (Common) terminals.

The bridge architecture used by the WiFi communication flow is where:

1. The Arduino processes sensor data and sends control commands via serial to the ESP8266.
2. The ESP8266 WiFi Bridge forwards these messages to the relay server running on the Raspberry Pi as HTTP POST requests.
3. The Raspberry Pi processes the data and returns commands that are relayed back to the Arduino.
4. The Android device communicates exclusively with the Raspberry Pi server over the WiFi network.

With the Arduino handling physical I/O, the ESP8266 handling network connectivity and the Raspberry Pi managing the business logic of the Smart Lighting System, this modular connection approach allows the system to function with a clear separation of responsibilities.

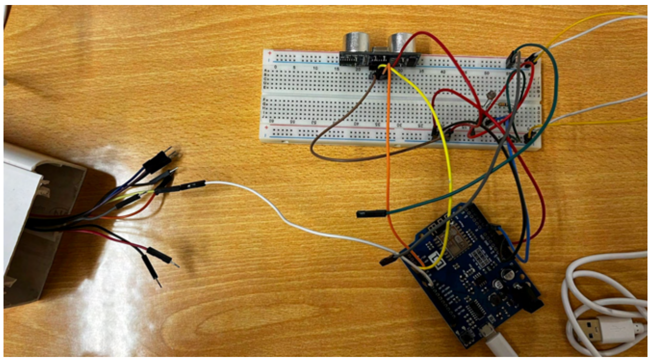


Figure 16: Illustration of the assembly and physical connections