ALMA MATER STUDIORUM -
UNIVERSITY OF BOLOGNA

EMBEDDED SYSTEMS AND INTERNET OF THINGS

ASSIGNMENT #3

Smart Room

Author
Fabio Veroli

May 19, 2023

Contents

1 Introduction

The project represents the development of the third assignment for the Embedded Systems and IoT course at UNIBO. We want to realise an IoT system implementing a simplified version of a smart room, as a smart system monitoring and controlling the state of a room (e.g. in a Campus).

The system is composed of five subsystems:

1. **Room Sensor-Board (esp):** embedded system to monitor the state of the room by using a set of sensors. It interacts with the Room Service (via MQTT).
2. **Room Service (back-end - pc):** service functioning as the main unit governing the management of the room. It interacts: through the serial line with the Controller (Arduino), via MQTT with the Room Sensor-Board (esp) and via HTTP with the Dashboard (front-end/PC).
3. **Room Controller (Arduino):** embedded system controlling lighting and roller blinds. It interacts via serial line with the Room Service and via Bluetooth with the Room App.
4. **Room App (Android - smartphone):** mobile app that makes it possible to manually control lights and roller blinds. It interacts with the Room Controller via Bluetooth.
5. **Room Dashboard (front-end/web app on the PC):** front-end to visualise and track the state of the room. It interacts with the Room Service.

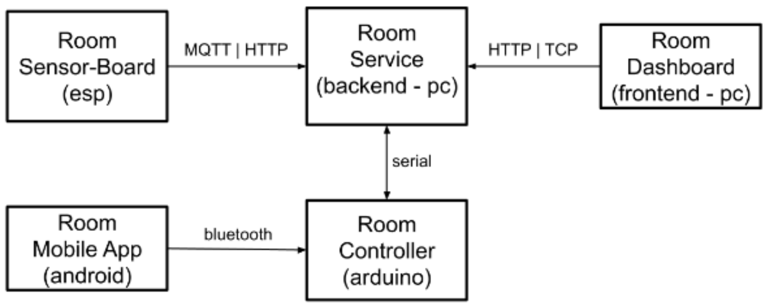
Room Sensor-Board (esp)
MQTT | HTTP
Room Service (backend - pc)
HTTP | TCP
Room Dashboard (frontend - pc)
serial
Room Mobile App (android)
bluetooth
Room Controller (arduino)

Figure 1: Scheme of components

2 Description

2.1 Hardware Components

Room Sensor-board :

- SoC ESP32 board
- 1 green led
- 1 PIR
- 1 photo-resistor analog sensor

Room Controller

- Micro-controller Arduino UNO board
- 1 green led simulating a light subsystem
- 1 servo motor simulating the roller blind subsystem
- 1 Bluetooth module HC-06

2.2 General behaviour of the system

The Smart Room system is meant to control the lighting system and roller blinds according to the following policy:

- If no one is in the room, the light (of the lighting subsystem) should be off.
- If someone enters in the the room and the room is dark, then the light should be turned on (if it was off).
- The roller blinds are fully rolled up automatically the first time someone enters in the room, from 8:00 (if someone enters).
- The roller blinds are fully unrolled at 19:00 (if they are up and no one is in the room), or as soon as someone who is still in the room at 19:00 leaves the room.

- Through the mobile app, a user can: turn on/off the light, roll up-/unroll – also partially (from 0 to 100%) the roller blinds.
- Through the dashboard a room manager can: track the state of the room, in particular in which hours and how long the lights where on, and fully control the light and roller blinds.

The light controlled by this policy representing the lighting system is represented by the green led in the Room Controller.
It can be assumed that the room is accessed from 8:00 to 19:00.

2.3 Further details

About the Room Sensor-board:

- The led should be on when someone is in the room and off when no one is the room.

About the Room Controller:

- The servo motor controls/simulates the roller blinds.
- 0° means roller blinds completely rolled-up.
- 180° means roller blinds completely unrolled.
- The green light simulates the lighting system: on/off.

No specific constraints/requirements are given for the Room Mobile App and the Room Dashboard.

3 Development

3.1 Circuit Diagram

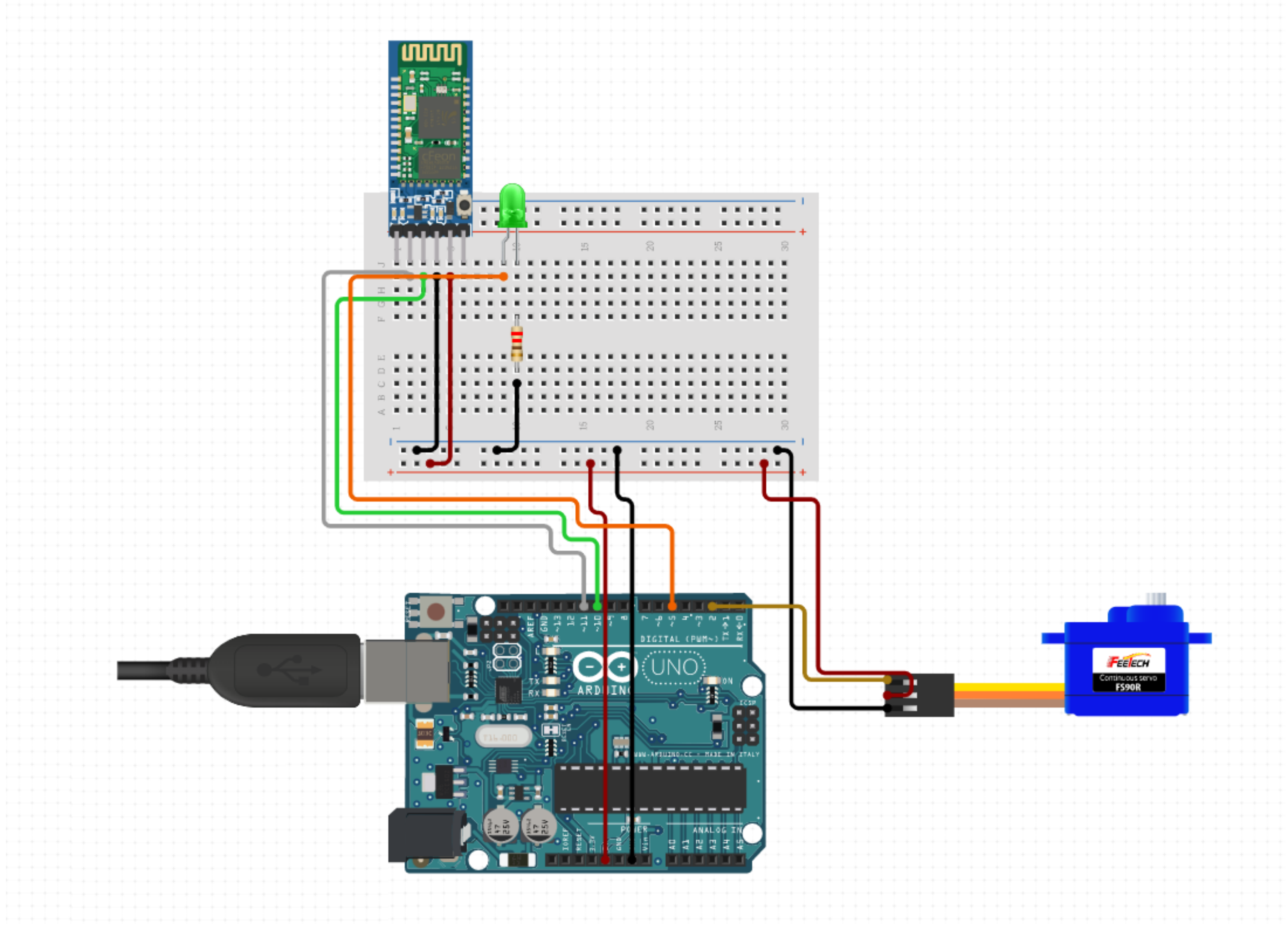

Figure 2: Room Controller Circuit Scheme

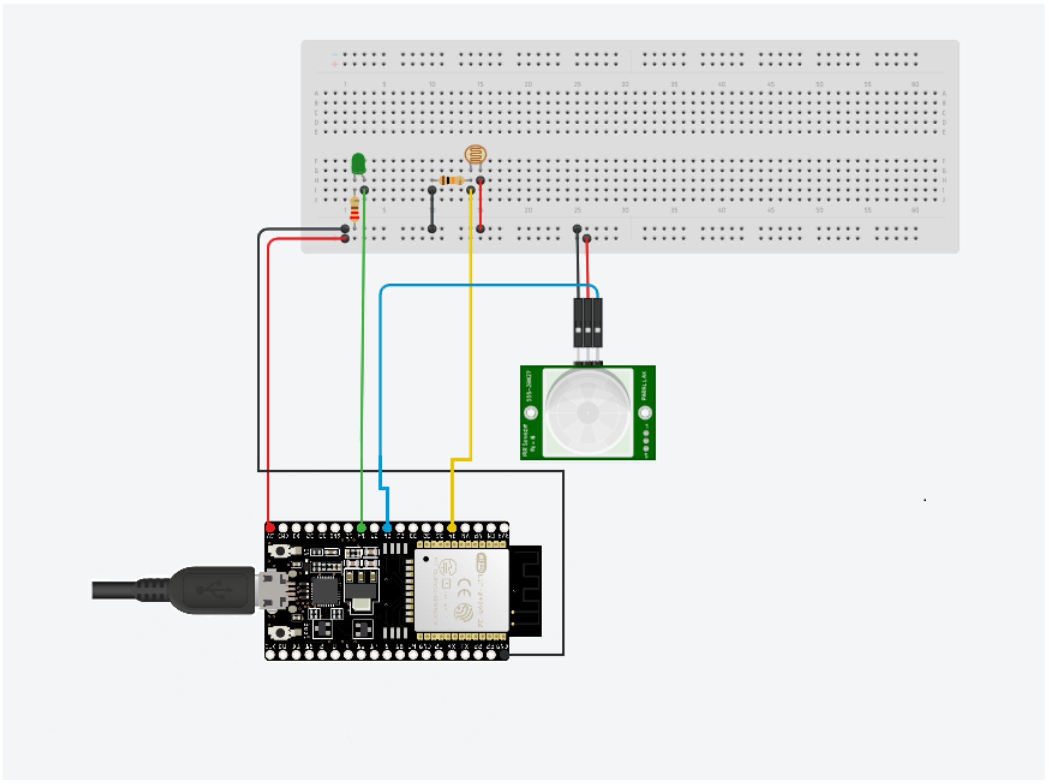

Figure 3: Room Sensor-Board Circuit Scheme

Note: Both the sensitivity potentiometer and the time-delay potentiometer, placed on the back of the PIR, were fully turned anticlockwise. This configuration ensures the minimum detection range, and it sets the output to remain HIGH for the minimum time after motion is detected.

Note: to ensure proper functionality, the photo-resistor of the Room Sensor-Board should be connected to an ADC (Analog to Digital Converter) pin of the ESP32 board. In this case, we utilize pin 3 (GPIO3/ADC1_2) for this purpose.

3.2 FSMs Diagram

The Room Sensor-Board system (fig. 4) is composed of three synchronous FSMs (or tasks), that work together:

- **Fetch Data Task:** consists of only one state, where motion sensor and light sensor are sampled whenever the period occurs. Additionally the current hour is obtained from the server. The obtained data are then used in the other tasks to determinate the current state.

- **Led Task:** responsible for switching the led of the Room-Sensor-Board on/off when the detection state changes.

- **Communication Task:** handles communication with the server based on specific conditions. An UNDEFINED state is used as an entry point to ensure that messages to the server are only sent when the FSM is in a "valid" state. The task is divided into two subsystem to enable independent message sending when each subsystem changes state.

All the tasks of the Room Sensor-Board have the same period, which correspond to the period used for sampling the data.

The Room Controller system (fig. 5), on the other hand, can be seen as a hybrid system. Periodically, the serial port and Bluetooth are sampled to check for the availability of any messages. This aspect makes it resemble a synchronous FSM. However, the state of the system is actually changed after the the occurrence of the received message event, and only if certain conditions are met. This characteristic allows the system to be considered a type of asynchronous FSM. Specifically, the system is divided into two subsystems that change state based on messages received from either the serial port or Bluetooth.

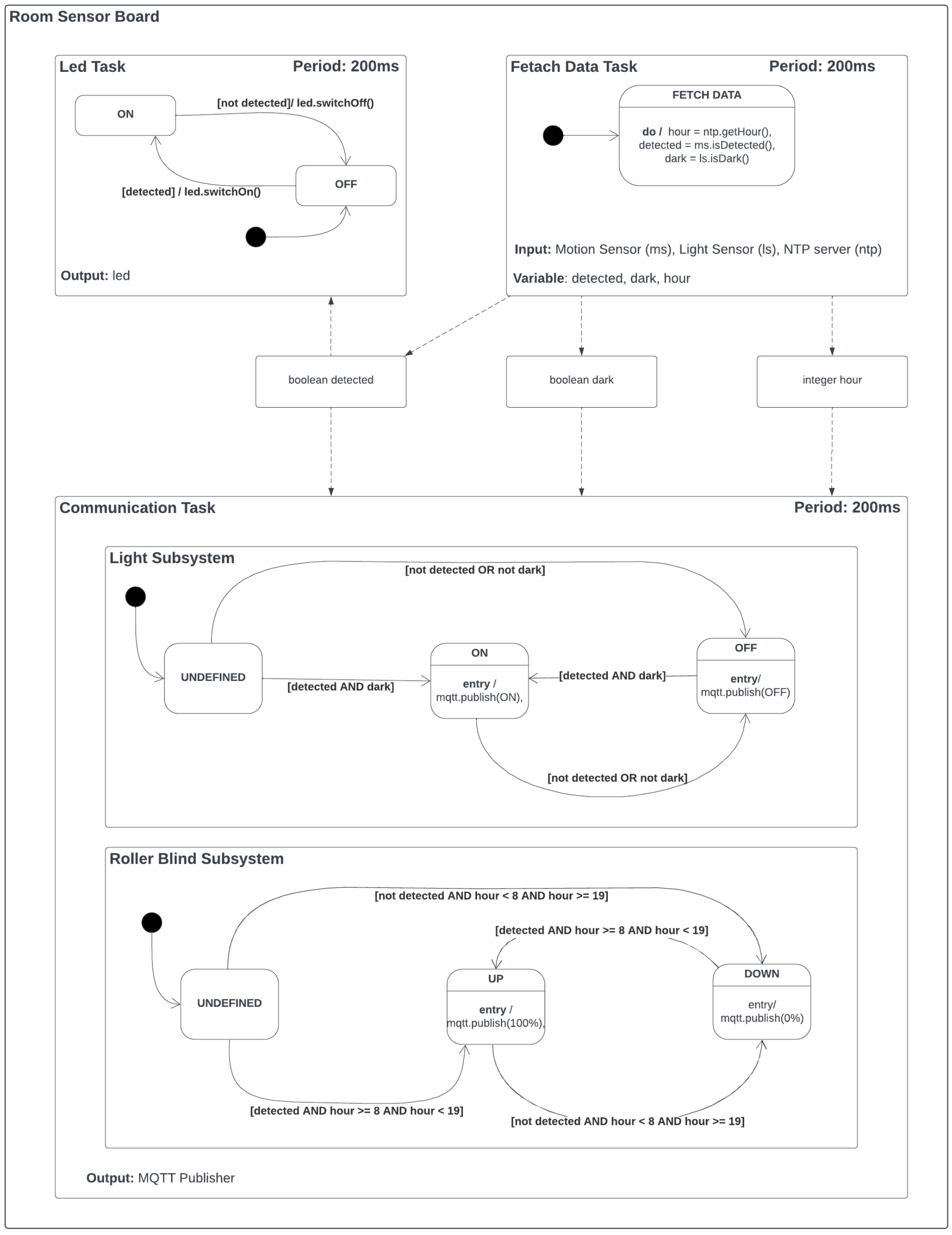
Room Sensor Board
Led Task
Period: 200ms
ON
[not detected]/ led.switchOff()
OFF
[detected] / led.switchOn()
Output: led
Fetach Data Task
Period: 200ms
FETCH DATA
do / hour = ntp.getHour(), detected = ms.isDetected(), dark = ls.isDark()
Input: Motion Sensor (ms), Light Sensor (ls), NTP server (ntp)
Variable: detected, dark, hour
boolean detected
boolean dark
integer hour
Communication Task
Period: 200ms
Light Subsystem
[not detected OR not dark]
UNDEFINED
[detected AND dark]
ON
entry / mqtt.publish(ON),
[detected AND dark]
OFF
entry/ mqtt.publish(OFF)
[not detected OR not dark]
Roller Blind Subsystem
[not detected AND hour < 8 AND hour >= 19]
[detected AND hour >= 8 AND hour < 19]
UNDEFINED
UP
entry / mqtt.publish(100%),
DOWN
entry/ mqtt.publish(0%)
[detected AND hour >= 8 AND hour < 19]
[not detected AND hour < 8 AND hour >= 19]
Output: MQTT Publisher

Figure 4: Final State Machine Diagram of Room Sensor-Board

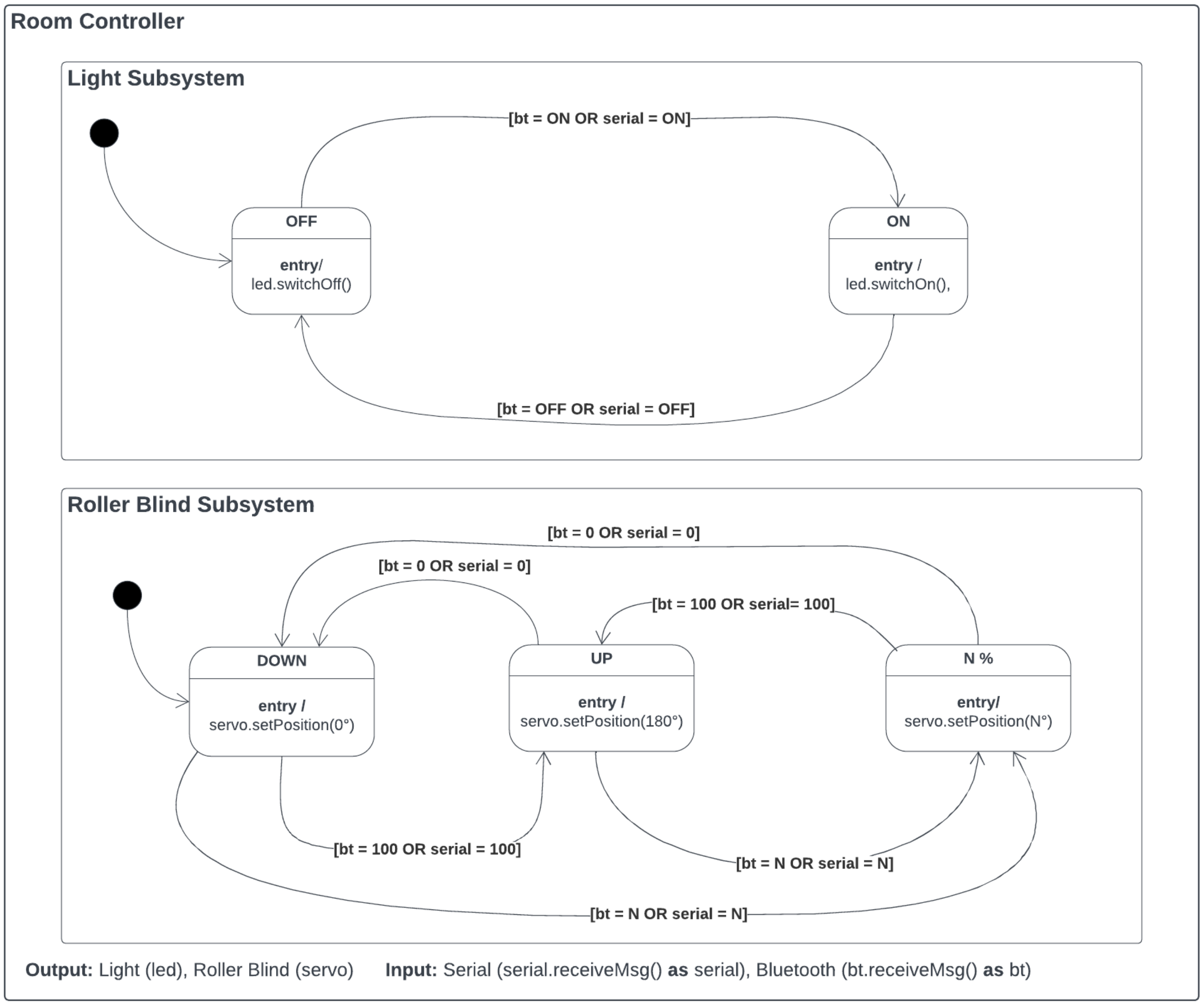

Figure 5: Final State Machine Diagram of Room Controller

3.3 Room Sensor-Board

The embedded software of the Room Sensor-Board was implemented on ESP32 using C++ in the PlatformIO environment. Each input/output module (PIR, photo-resistor, led) was mapped to a C++ class that defines its behaviour. The system consists of three tasks, as described in figure 4, managed by FreeRTOS.
The Fetch Data Task connects to an NTP (Network Time Protocol) server to obtain the current hour. This allows the system to be fully autonomous and send messages only when specific hour conditions are met.
The Communication Task publishes messages on two different MQTT topic, one for the light and one for the roller blind, but only when specific condition are met. The messages are sent in JSON format, using ArduinoJSON library.
The system operates independently and only communicates with the server in one direction. Therefore, it is unaware of the current state of the Room Controller, which may have been changed from the dashboard or the mobile app. The system only maintains its previous state. This approach was chosen to minimize the number of messages sent to the server, ensuring that messages are sent only when the sensor-board changes its previous state.

3.4 Room Service

The Room Service was implemented using the Node.js environment. The system receives messages through MQTT from the Room Sensor Board, and sends/receives messages through the serial port from the Room Controller and via HTTP from the Room Dashboard.
The server uses the JSON format for all the messages sent or received. This ensures a uniform format that can be used by all the systems within the Smart Room for communication.
The server collects all the received data regarding state changes of the light or the roller blind. Each time a message is received, it is assigned a date object and added to a separate array based on the message type. In a real system, the server would write the received messages to a database to maintain a record of the room's previous states.
Whenever the server receives a message from the Room Sensor-Board or the Room Dashboard, it immediately sends a message to the Room Con-

troller to change its state.
To run the server, execute the following command within the `room-service` folder: `npm start`

3.5 Room Controller

The embedded software of the Room Controller was implemented on Arduino UNO using C++ in the PlatformIO environment. Each input/output module (Bluetooth, led, servo) was mapped to a C++ class that defines its behaviour. The system consists of two communicators: one used for Bluetooth communication with the Room App and the other for serial port communication with the Room Service. The system's behaviour, based on FSMs, is described in figure 5.
During each iteration of the `loop()` cycle, the system check if any messages are available, both from Bluetooth or the serial port. If a message is available, it is decoded using the ArduinoJSON library, and the room state is updated if specific conditions are met. Whenever a message is received via Bluetooth, the systems also sends the message to the serial port, ensuring that the server is updated with the new room state. Similarly, when a message is received through the serial port, a message is sent via Bluetooth to update the Room App's state.

3.6 Room App

The Room App was implemented using the open-source framework Flutter. The app utilizes the `flutter_bluetooth_serial` library to establish a Bluetooth connection and exchange messages with the Room Controller. The app consists of two pages:

- **Main Page:** (fig. 6) it allows the user to view the current state of the room and make changes to the light state (using a switch) or the roller blind state (using a slider).

- **Discovery Page:** (fig. 7) it enables the user to select the Bluetooth device to connect to after pairing. The app is specifically designed to work with the Bluetooth module of the Room Controller, identified as `room_controller`.

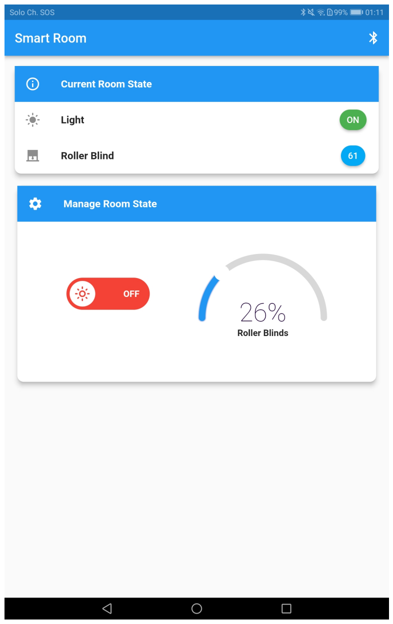

Figure 6: Main Page of the Room App

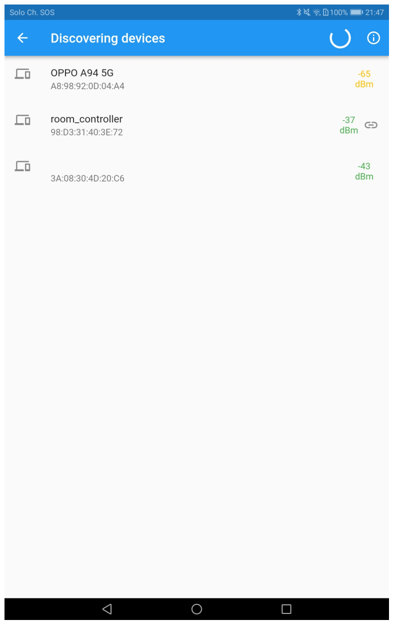

Figure 7: Discovery Page of the Room App

3.7 Room Dashboard

The Room Dashboard was implemented using React.js. To run the dashboard, execute the following command within the `room-dashboard` folder: `npm start` or `yarn start`. The dashboard is a simple web page that allows the user to monitor and change the room state.
The dashboard provides the user with the ability to view the current state of the room, including the light and roller blind. It also displays charts showing all the previous reported states, with separate charts for the light and roller blind. The current state of the room is updated every second, providing near real-time information. The charts, on the other hand, are updated only when the user requests the data by clicking a refresh button. This approach minimizes unnecessary component re-rendering.
To create the charts, the `react-chartjs-2` library was used. This library serves as a wrapper for utilizing the Chart.js library within a React application.

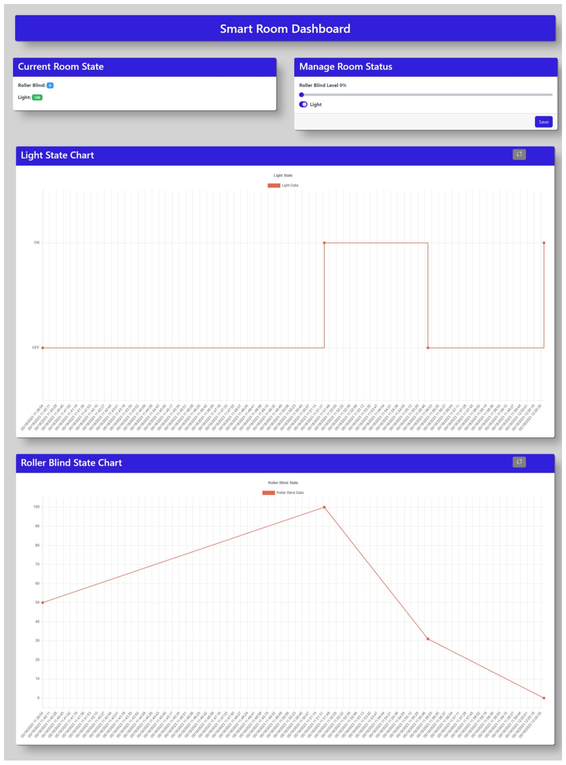

Figure 8: Room Dashboard

4 Documentation

A brief video demonstrating how the system works can be found both on my OneDrive archive and on the project GitHub repository.

The complete code of the project is available on the project's GitHub repository.