10661018 - Bendazzoli Luca
10629553 - Gualla Pietro

PROJECT REPORT

INTRODUCTION

The key idea of our project was to create a domotic system which autonomously manages common IOT smart devices allowing also user interaction thanks to a Telegram-Bot.
More specifically the objective was to automatically and remotely control the position of a roller shutter and the status of lights in our livingroom according to the values of temperature, humidity and brightness measured by an Arduino Nano board provided with the needed sensors.

The actuators we used to control both the roller shutters and the lights are *Shelly 2.5* by *Allterco Robotics* because it is possible to enable them in order to exploit IOT communication protocols such as COAP and MQTT.
As for COAP, they actually use a slightly modified, not open source, variation of the protocol called CoIOT therefore we decided to choose MQTT communication.
Shellies are so MQTT clients that publish on “status” topics and are subscribed to “commands” topics: this means that to change a Shelly status we need another client to publish on the “commands” topics.

For what concerns the sensors part we used an Arduino Nano IOT 33 board, a DHT11 “humiditure” sensor (to measure humidity and temperature), an analogic photocell to sense brightness level and a Passive Infrared Sensor. The Arduino board is a MQTT client too and it only publishes on the topics related to its measurements.

To group all the devices, we used the MQTT broker *Mosquitto*, we installed it on our *Raspberry Pi* that also hosts *NodeRed*, from which we managed all the control logic and also the *Telegram Bot*. *NodeRed* is the real core of our work as, besides managing all the Telegram part, it is the Master MQTT client which reads (as a subscriber) all the sensed data and all the Shellies statuses and, based on these values, provides a “control law” (as a publisher) in form of setpoints for the actuators acting as a centralized regulator for the distributed system.

As said above the system is not only autonomous but it can also be controlled remotely through our Telegram Bot. There are several functions available that, in a nutshell, allow the users to read the sensors measurements (there is also a ThingSpeak channel) and to send commands to the actuators.

Moreover, we discovered during the project that also our Samsung air conditioners were Smart devices, so we tried to include them in the system, which we were able to do only partially as it is possible to control these ACs only through HomeAssistant.
Even though this is not really an IOT application, as it works on http, it provides a cool (:p) feature because we added the automatic control of the ACs according to the house temperature guaranteeing energy wasting prevention.

Finally, we also managed to connect the devices to the Amazon Alexa assistant thanks to a NodeRed palette we found online that mocks Alexa and makes it recognize all the devices as a Philips Hue smart lightbulb. After setting up some routines to make the commands more intuitive (as Alexa thought the devices were lights) we were able to control the entire system with vocal commands.

HARDWARE

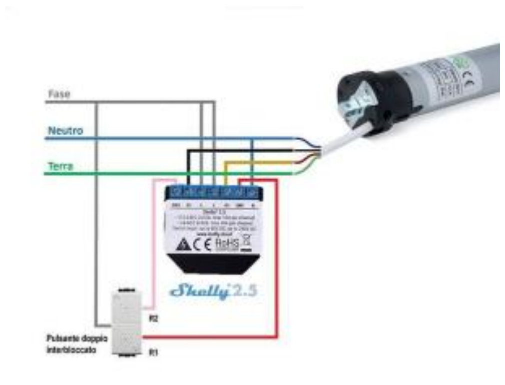
ACTUATORS: Shellies Circuits and System Physical Layout

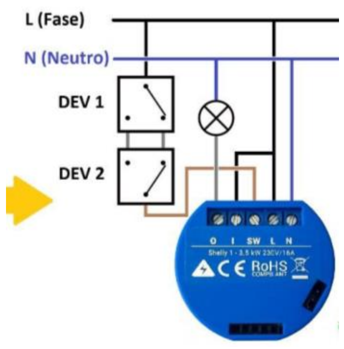
Our house is composed by three rooms: the kitchen/livingroom and two bedrooms.

In the livingroom there are two roller shutters which are referred to as *yard* and *road* (due to their position), they both are provided with a Shelly actuator.
The lighting of the room is composed by two different diverted circuits, one commanding LED spotlights called *mainlight* and the other related to a LED strip positioned in the false ceiling, which delivers more of a soft light, and so called *ambientlight* (schemes below). Connecting one circuit per channel to a second Shelly (schemes below) we were able to control the room lighting.
Also the Arduino board is located in the kitchen where it measures humidity, temperature, brightness and human presence. Finally, in the room there is also a smart air conditioner.

The main control idea is to open/close the roller shutters and turn on/off the lights in order to maximize the room brightness and minimize the energy consumption due to the lighting system (e.g.: it is a sunny afternoon, you are in the kitchen with lights on and roller shutters closed: the controller opens the roller shutters and turns off the lights).
Another control feature concerns the AC as we set a maximum and minimum temperature threshold to control the cooling systems, creating a sort of saturation loop that prevents the house to heat up too much but, contextually, avoids too low temperatures which of course mean pointless energy consumption and pollution production. In a similar way, thanks to the humidity measure, the system can automatically select the mode of the AC between cooler and dehumidifier. This automated control is active only if the presence sensor detects people in the room.

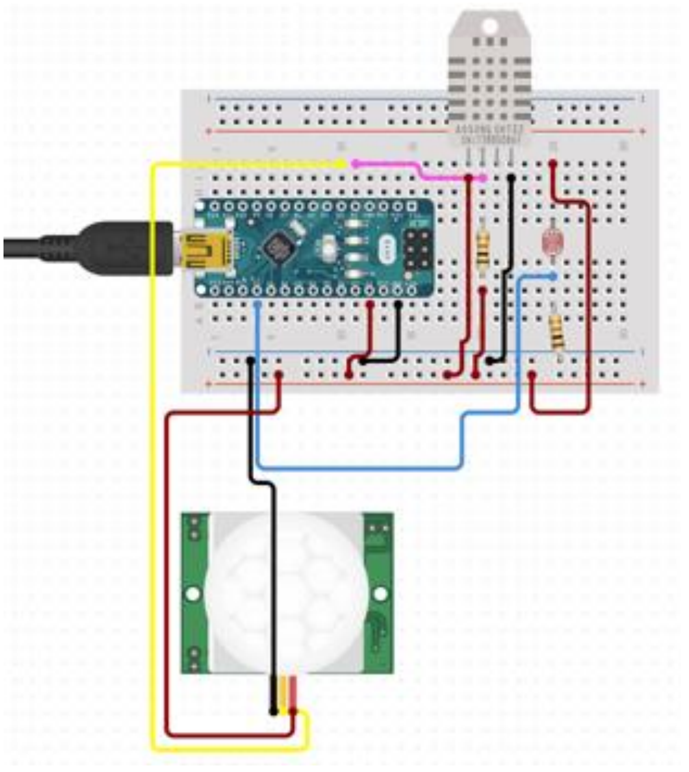
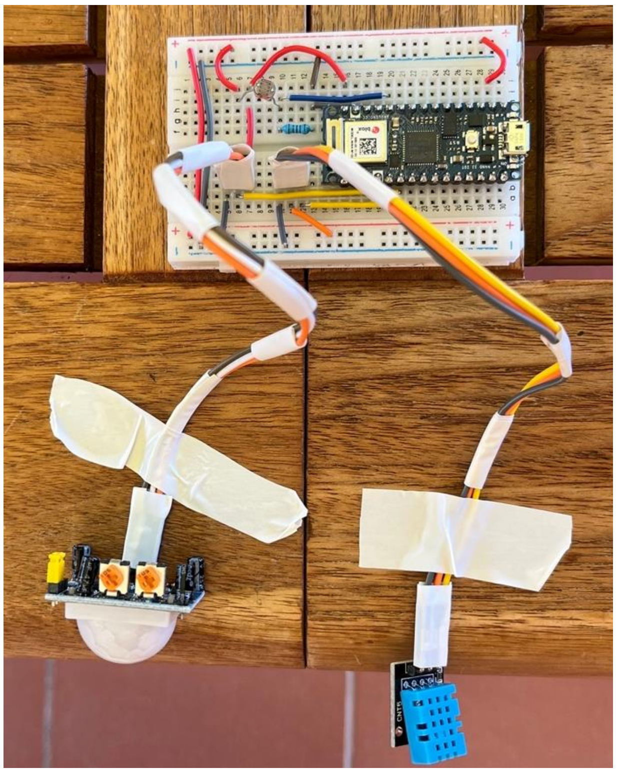
The bedrooms present a different and simpler layout, as they both have only an AC and a roller shutter and they are not provided with any sensing apparatus.
As there are no sensors in the bedrooms they are not featured with any automation because we thought that relying on sensors set in another room wouldn't have been a robust choice and that putting an Arduino board in each room would, on the other hand, have been a waste of hardware, considering also the lack of actuators.
We preferred to provide a wired connectivity to the Raspberry Pi being it the key device of the entire system, so we put it in one of the two bedrooms (referred to as main bedroom or Pietro bedroom) because it's where the router is located.





SENSORS: Arduino Schematics

Below there is a mock-up of our circuital layout (on the left) designed with *circuito.io*, a tool found online, and the real circuit on the right.
The white sensor (in the drawing) is a DHT humidity and temperature sensor, the one showed is a DHT22 while we actually used its "brother" DHT11 (the blue one, not available in the graphic tool) which has a sensing range from 0 °C to 50 °C with ±2°C of accuracy for temperature whereas it measures RH from 20% to 90% with ±5%RH accuracy.
The red one on the right part of the breadboard is the representation of an LDR photocell (pin 10-6 in our breadboard); it is an analogic sensor and it works thanks to a small resistor which variates its resistance properties with respect to the amount of light it is subject to, with inverse proportionality. By supplying the series of the photocell and a common resistor (ours is 10kΩ) with a constant voltage it is possible to infer the value of the voltage on the LDR by measuring the voltage drop on the 10kΩ resistor and applying the voltage divider equation knowing that the supply voltage is constant.
The sensor below is a passive infrared motion (PIR) sensor to detect human presence. It is not very fast as it takes almost 60s to heat up and to start sensing correctly and its minimum response time is 5s. It has two regulations through potentiometers that allow to select the sensitivity (30cm – 7m) and the delay, namely the amount of time the output will be high after a detection (3s – 200s).
It also has a yellow jumper to select the trigger mode (L -> after a trigger output stays high for a delay time; H-> same as L but if it is re-triggered the timer restarts).
Our setup is 3m sensitivity and 2 minutes time delay (ballpark, as they depend on potentiometers without levels) with re-triggering.
We supplied this system both by wire and by a 12V pack of 8 AA NiMH batteries, even if we didn't have the time to test out its endurance in the latter supplying mode, we estimated it to be around one month, due also to the software power saving implementation.

MQTT NETWORK

As already mentioned, the Raspberry Pi is the Broker of the MQTT Network while of course each device is a client, including the NodeRed instance running on the Raspberry Pi itself.
The MQTT topics were created consistently to be easily checked and to have the possibility to expand the network adding new devices without any particular problem, here are some topics examples:

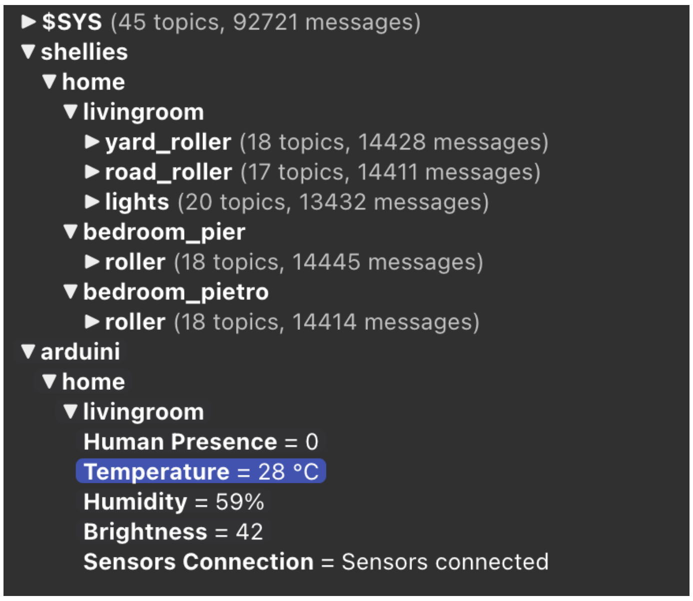
Rollers sub topic: *shellies/home/*room_name*/*device_name*/roller/0/command/pos*
Rollers pub topic: *shellies/home/*room_name*/*device_name*/roller/0/pos*
Lights sub topic: *shellies/home/*room_name*/*device_name*/relay/*relay_name*/command*
Lights pub topic: *shellies/home/*room_name*/*device_name*/relay/*relay_name**

From the examples above it is clear that the Shellies are setup to publish their status on some topics which NodeRed must be subscribed to and, likewise, NodeRed publishes commands for the Shellies on other similar topics which the actuators are subscribed to.
Shellies also publish information about their connection status, their temperature and power consumption together with a lot of other data we didn't really directly use for our project.
The Arduino sensors are MQTT clients too but they only publish periodically their measurements on the related topics.

"Devices" list:

- Mosquitto (MQTT Broker)
- NodeRed (Control Unit)
- Arduino Nano (Sensor)
- Livingroom Road Roller Shutter (Actuator)
- Livingroom Yard Roller Shutter (Actuator)
- Livingroom Lights (Actuator)
- Main Bedroom Roller Shutter (Actuator)
- Second Bedroom Roller Shutter (Actuator)

Here is a hierarchical view of the devices and topics:



```
▶ $SYS (45 topics, 92721 messages)
▼ shellies
  ▼ home
    ▼ livingroom
      ▶ yard_roller (18 topics, 14428 messages)
      ▶ road_roller (17 topics, 14411 messages)
      ▶ lights (20 topics, 13432 messages)
    ▼ bedroom_pier
      ▶ roller (18 topics, 14445 messages)
    ▼ bedroom_pietro
      ▶ roller (18 topics, 14414 messages)
▼ arduini
  ▼ home
    ▼ livingroom
        Human Presence = 0
        Temperature = 28 °C
        Humidity = 59%
        Brightness = 42
        Sensors Connection = Sensors connected
```

SOFTWARE

ARDUINO SOFTWARE

The Arduino Board, as already addressed, is an MQTT client that only publishes. Its publishing topics are the ones related to < *arduini/home/livingroom/#* > where it updates the sensed values once a minute.
We implemented in the Arduino code a power saving functionality such that the board is basically by default into *deepSleep* mode and it wakes up every minute to measure the sensors statuses.
We initially thought that this solution would have been unfeasible due to the fact that the PIR sensor must be continuously supplied in order to be effective. By increasing its delay time and connecting it directly to the power supply shield instead of the 5V board pin of the Nano, we were able to implement both the power saving mode and an effective application for the PIR.

In the setup phase the sensor board establishes the Wi-Fi connection and then connects to the MQTT broker.
In the loop of the sketch the Arduino checks the Wi-Fi connection, if it is not able to connect to Wi-Fi in 10 attempts the board goes into deep sleep mode (we developed this function because during a test we were doing, supplying the board with a 9V alkaline battery, there was a blackout, the board kept trying to reconnect vainly and drained the entire battery in just 4 hours).
Then, if not already connected, it reconnects to the MQTT broker (if it is not able to connect in a bunch of iterations it goes into deep sleep for 10 minutes) and, finally, publishes on the *Sensor Connection* topic a "Sensors Connected" message; we set this topic to be also the *lastWill* one and, knowing that after 30s from the disconnection the *lastWillmessage* is sent, we decided that to be "Sensors Sleeping" so it gave a clue on the sensors status.
To detect if the sensor disconnects, we used kind of a trick by publishing through NodeRed on the same topic as above a "Sensors disconnected" message each 5 minutes. A MQTT subscriber node then checks the topic and, if there are 3 consecutive disconnection messages, it shows the status of the sensors as disconnected.
It is possible to check the sensors status in Telegram with the *</STATUS>* command which shows (among many things) the sensor as either *Connected*, *Sleeping* or *Disconnected* (see Telegram commands paragraph).
In the MQTT connection process we also specified the *clean session* type of connection as our Arduino does not subscribe to any topic so it would have been useless a non-clear session.
After all these processing the sensors are finally read and the measured values are published on the related topics. The board can finally go to sleep.

Arduino Libraries

<WiFiNINA.h>
<PubSubClient.h>
<DHT.h>
<ArduinoLowPower.h>

NODERED SOFTWARE

Home Automation

To develop the home automations introduced above we decided to use NodeRed because we thought it had the right trade-off between high level user-friendly features and lower level coding possibilities given by the function nodes.
The *Home Automation* NodeRed flow has several MQTT Subscriber nodes to collect Temperature, Humidity, Brightness and PIR output from the Arduino Board and Status/Position/Power from the Shellies. These data are saved into global variables after being filtered so as only if the sensed value changes the variable is actually modified.
Moreover, it is computed a mean over the last five "humiditure" sensed values which are then sent via MQTT to our ThingSpeak channel (*Channel_id*: 1813859) together with roller positions and total power consumption.
Subsequently, it is implemented the real control logic which determines the commands to be sent to the actuators with respect to the measured values.
As already hinted the controller opens/closes the roller shutters and turns on/off the lights to limit unnecessary energy consumptions following the simple principle that, if possible, the house must be lightened with natural light. To put this into code we used, in the developing phase, the Arduino Serial Monitor to display the measured brightness and, after many trials in different conditions, we set our personal "too dark" threshold.
So basically, if the Livingroom is "too dark" there is a check if the roller shutters are open, if so, the lights are turned on. On the other side if it is day and the lights are on there is a check if the roller shutters are closed, in that case they will be opened to let sunlight in and the lights will be turned off, notice that in this situation it might happen that the lights will turn back on due to the "too dark" condition. All the commands to the Shelly actuators are sent by publishing on their command topics.

The air conditioners have a very similar rationale behind their control, except for the fact that they are not managed with MQTT but they rely on Home Assistant (http). Anytime the livingroom is "too hot" they are turned on and anytime the temperature goes behind a "too cold" threshold they are turned off. Moreover, thanks to the humidity measure, they also select the best function mode between "cold" and "dry", providing a dehumidifying action if needed.

For both the Shellies and ACs it is important to understand if anyone is home otherwise it would be pointless to act on the devices, that's why our system provides for a PIR motion sensor which detects if the tenants are in the livingroom. If so, it allows the automations occurrence and, on the other hand, it autonomously disengage them if the PIR sensor doesn't detect any motion for a while. This can be also done manually by the user through the Telegram Bot "go out" command which turns off all the devices and closes all the roller shutters.

The NodeRed flow also manages the Alexa integration with our project. Most of the work was actually done by a developer whose NodeRed palette we found online. This palette, together with a related Alexa Skill, allowed us to register in the developer's website our devices names and, through the Skill, make them appear as other Amazon Alexa compatible IOT devices, more specifically Philips Hue lightbulbs. Once done this, we were finally able to restore the connections with Shellies (not available as Alexa communicates in Cloud and MQTT deactivates Shelly Cloud) and to create new connections with the ACs that were previously non compatible.

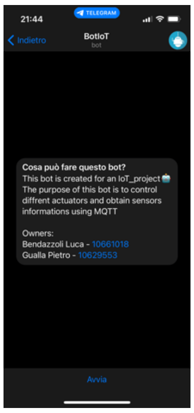
Telegram Bot

The idea of creating a Telegram Bot is not only a "decoration" for the project, it indeed was fundamental for the realisation of an effective and complete system. This because, for security reasons, once enabled MQTT the Shelly Cloud service ceases to work, so without Telegram, we would have actually reduced the functionalities of our already kind of smart home.
The Telegram Bot *(@iot_berna_bot)* was created thanks to the BotFather but developed on NodeRed.
We decided to implement the Bot on NodeRed for several reasons such as the higher flexibility which allows fast and easy changes in Bot menus and the possibility to interact with files.
The latter point was crucial for us because we wanted to develop a security layer to protect the Bot and our house.
In Telegram it is impossible to create a private Bot so, once created, theoretically speaking any user in the world is allowed to access it (of course it must be searched) and exploit its functionalities.
Our idea was to locally create a distinction between users taking advantage of the *chat_id*, which is the unique identifier of a Telegram user. By creating a file with a list of allowed users (from now on called *subscribers*) we implemented the protection layer allowing only to *subscribers* the access to "hotter" commands. It wasn't enough for us, so we introduced also the *admin* figure (we developers are the admins) who has the real power to ask for all the commands. Moreover, admins are the only ones with the authority to promote common users to *subscribers*.
All the different authorizations to send commands to the bot can be find alongside the commands themselves in the Bot Menu: a green/yellow/red dot shows which users are allowed to ask the related commands, respectively anyone/subscribers/admins (see next page).

Before explaining the actual Bot commands, it is worthy to briefly clarify how they are managed in NodeRed. When the bot receives a generic *command*, besides the trivial situation in which it only answers with a message, it sends back an interactive message to the user called *inline keyboard*.
Once sent the keyboard, the Bot waits for the user interaction and, on its reception, it is collected by a *Callback Query* node. This node, after evaluating the received payload, finally starts the flow that executes the right command.

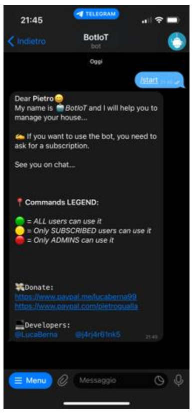
Once a generic user comes across our Bot, he/she receives a welcome message (the same triggered by the *</INFO>* command) that explains the main features of the Bot.
To have access to other commands the user must call the *</SUBSCRIBE>* one (see next page).
After the subscription request, a message is delivered to the admins group, asking the permission to promote the generic user with an inline keyboard. If accepted the user is registered in the subscribers file and, for this reason, gains the access to subscribers' commands.
A subscriber can request *</HISTORY>* command, which sends a link to our ThingSpeak channel (link above), and *</STATUS>* command that returns an inline keyboard that allows subscribers to pull a specific sensor value or to ask the status of all the devices at once (see next page).
Finally, admins are allowed to use *</COMMAND>* command through which it is possible to set a target status for each Shelly actuator thanks again to inline keyboard selection (we did not implement it for air conditioners as it is useless to switch an AC on if you are not home) and *</LIST>* command that displays a list of the authorized subscribers.
Admins also have access to two hidden commands that are *</DEPLOY>* and *</DELETE>*; the first one can trigger a deploy of NodeRed while the latter (very similar to *</LIST>*) shows all the subscribers, again in an inline keyboard, and allows admin to remove them with a single tap.

- The starting page (left) which autonomously activates the </start> command (centre) and the commands Menu (right)



Cosa può fare questo bot?
This bot is created for an IoT_project🤖
The purpose of this bot is to control diffrent actuators and obtain sensors informations using MQTT

Owners:
Bendazzoli Luca - 10661018
Gualla Pietro - 10629553



/start

Dear Pietro😊
My name is 🤖*BotIoT* and I will help you to manage your house...

✍ If you want to use the bot, you need to ask for a subscription.

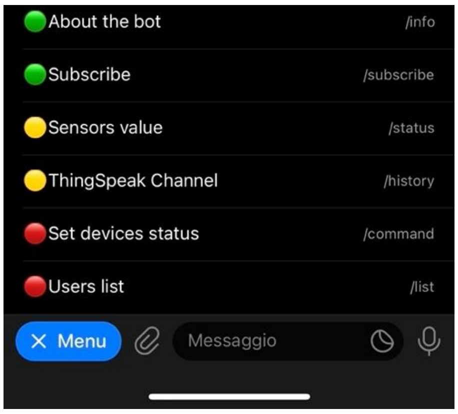
See you on chat...

📍 Commands LEGEND:

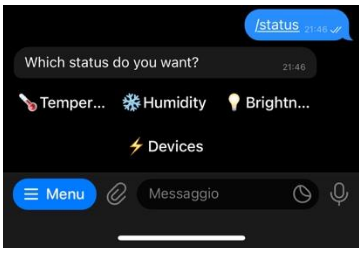
🟢 = *ALL users can use it*
🟡 = *Only SUBSCRIBED users can use it*
🔴 = *Only ADMINS can use it*

💸Donate:
https://www.paypal.me/lucaberna99
https://www.paypal.com/pietrogualla

💻Developers:
@LucaBerna @j4rj4r61nk5

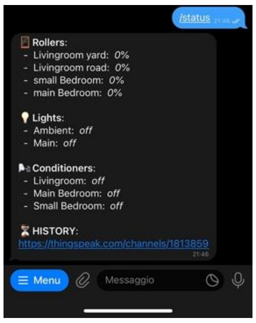


- The </status> command Inline Keyboard (left) and the response to Devices Callback Query



/status 21:46

Which status do you want? 21:46



/status

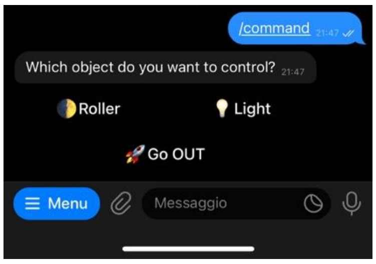
Rollers:
- Livingroom yard: *0%*
- Livingroom road: *0%*
- small Bedroom: *0%*
- main Bedroom: *0%*

💡Lights:
- Ambient: *off*
- Main: *off*

Conditioners:
- Livingroom: *off*
- Main Bedroom: *off*
- Small Bedroom: *off*

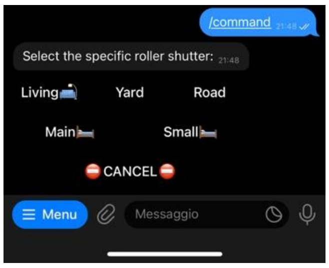
⌛HISTORY:
https://thingspeak.com/channels/1813859
21:46

- The </command> command (left) the Keyboard selecting Roller (centre) and the Keyboard selecting Living (right)



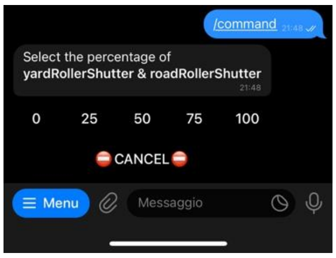
/command 21:47

Which object do you want to control? 21:47



/command 21:48

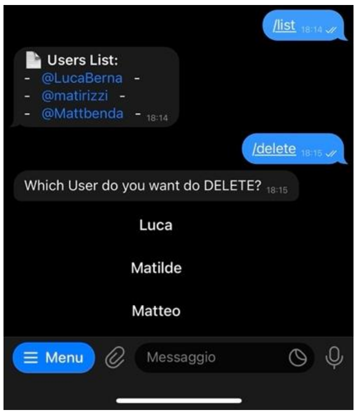
Select the specific roller shutter: 21:48



/command 21:48

Select the percentage of
yardRollerShutter & roadRollerShutter
21:48

- The </list> command and the hidden command </delete>

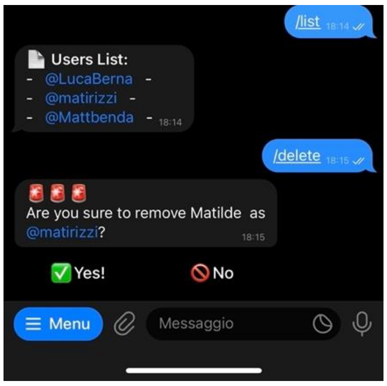


/list 18:14

📄 Users List:
- @LucaBerna -
- @matirizzi -
- @Mattbenda - 18:14

/delete 18:15

Which User do you want do DELETE? 18:15

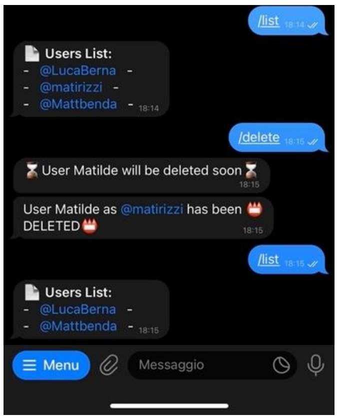


/list 18:14

📄 Users List:
- @LucaBerna -
- @matirizzi -
- @Mattbenda - 18:14

/delete 18:15

🚨🚨🚨
Are you sure to remove Matilde as @matirizzi?
18:15



/list 18:14

📄 Users List:
- @LucaBerna -
- @matirizzi -
- @Mattbenda - 18:14

/delete 18:15

⌛User Matilde will be deleted soon⌛
18:15

User Matilde as @matirizzi has been 📛DELETED📛
18:15

/list 18:15

📄 Users List:
- @LucaBerna -
- @Mattbenda - 18:15

NodeRed Palettes

node-red-contrib-home-assistant-websocket
node-red-contrib-telegrambot
node-red-contrib-fs-ops
node-red-contrib-simpletime
node-red-node-openweathermap
node-red-node-smooth