POLITECNICO DI TORINO

Innovative Thermostat Project

“Projects and laboratory on communication systems”

Professor: Guido Albertengo

Students: GROUP PL-14
Marco Florian, Juan David Quintero Gómez, Bernardita Štitić

1. Delivered material

- PL-14 relation
- controllerPL-14.gif: animated gif of the interface
- screenshot: screenshot of the interface
- controllermd: java code
- controllermd-0.2.7-SNAPSHOT.jar: controller logic (all the main method are commented)
- controllermd-doc: all the documentation related to the controller jar (README to understand it better)
- chromium.sh: the script used to launch chromium in kiosk mode
- mainActuator.py mainSensor.py: the python script that handle the embedded sensor and actuator
- thermostat_*.service: the service that will run the controller, mainActuator.py and mainSensor.py
- relay.ino sensor.ino: the code used by the esp8266

2. Selection of hardware components

2.1. SoC for multi zone management: ESP8266

For multi zone management, we've chosen the ESP8266, a low-cost SoC that provide:

- Wi-Fi connectivity
- MQTT protocol support
- Enough GPIO pins operating on 3.3V logic like the Rpi
- 512 bytes of internal EEPROM, to save data that should be retrieved after resetting the system
- Compatible with the Arduino IDE

2.2. Temperature and humidity sensor: DHT22

Reason behind this choice:

- meets the specification requirements
- measure relative humidity
- is easy to use with the esp and the rpi through free available library

2.3. Actuator: Solid State Relay (SSR) double module

To simulate the activation (or deactivation) of heaters and coolers, a relay was chosen for the demonstration. It's a simple SSR (brand: Tongling) which supports the following electrical specifications on the high voltage side:

- 5VDC
- 10A, 250 VAC
- 15A, 125 VAC

Therefore, it can be used to interface the low voltage side (the Raspberry Pi or the ESP8266, which both operate using 3V3 logic) with a high voltage side of 10A, 220 VAC as the project requires to simulate how the SoCs turn on/off the heaters and coolers. Moreover, this relay module is especially developed for Arduino like projects, so the microcontrollers chosen for this project can be connected to the SSR easily.

2.4. Other components

Only to mention them, other components chosen for the demonstration of the project include high precision resistors, LEDs, an Arduino UNO which supplies 5V (which will act as power supply), a battery, jumper wires and breadboards.

3. Hardware setup

3.1. Sensor station

A sensor station consists of an ESP8266, a DHT22 and a 10 kΩ resistor, which is needed for the connection. A schematic of the setup in the case of an ESP8266 is shown below. The connections for the Raspberry Pi (the main station), which also has a sensor connected to it, are exactly the same; what changes is the GPIO pin being used.

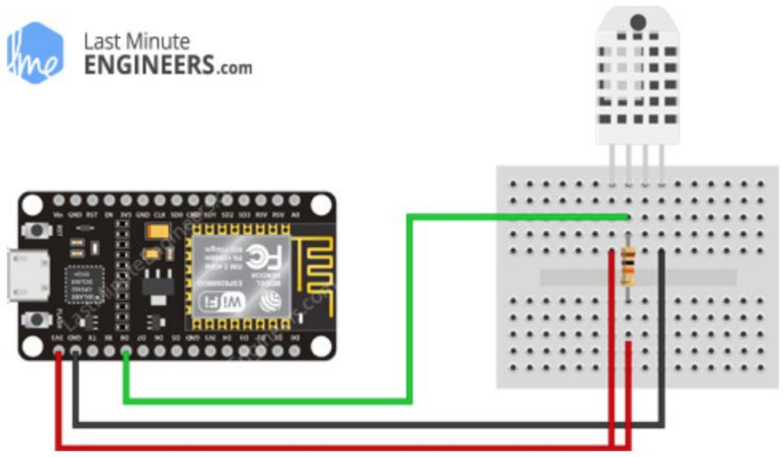


Figure 3.1.1. Sensor schematic, taken from [1].

As it can be seen above in Figure 3.1.1., the sensor has four pins. From left to right, these pins are: VCC, DATA (signal, digital), NULL, GND. The VCC pin of the sensor needs to be connected to the 3V3 output power pin of the ESP. The DATA pin should be connected to a GPIO pin of the ESP8266; for this project, this will be GPIO D1 on the board. As it can be seen, a resistor is needed; for the project a high precision, blue, 10 kΩ resistor was selected.

Finally, the other node of the resistance should be connected to the 3V3 output power pin of the board, and the GND pin of the sensor should be connected to a GND pin of the ESP8266.

3.2. Actuator station

An actuator station consists of an ESP8266, the double relay module (solid state), a high precision 10 kΩ resistor, an LED and the high voltage source (which can be a battery, or also an ARDUINO UNO can be used, since it can output 5V DC, for the demonstration). The connections are similar for the case of the Raspberry Pi; only the GPIO pin changes.

It wasn't possible to find an exact schematic for the ESP8266 on the internet. Instead, different parts of the final schematic and instructions were found, including relay connections to Arduino boards, which were used as reference to make the connections [3, 4, 5]. Figure 3.2.1 below shows the relay double module of this project with its pins, and Figure 3.2.2 shows the connections (real station).

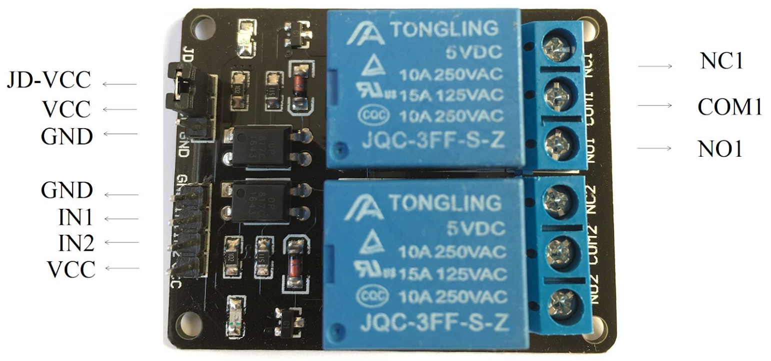


Figure 3.2.1. SSR double module; relay picture without pin names taken from [5].

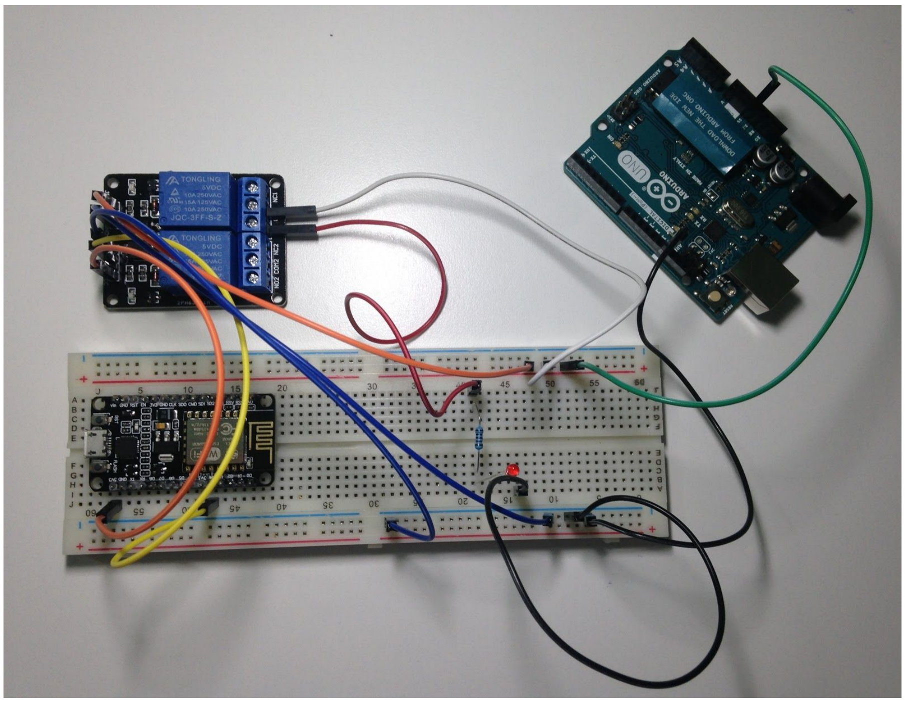


Figure 3.2.2. Actuator station connections, ESP8266 version.

Basically, considering Figure 3.2.1, first the jumper cap must be removed. Now, the connections will be described starting from JD-VCC on the left side of the picture at the top. Once the last pin on this list is reached, then the connections to the pins on the right will be described, starting from the top (NC1). Therefore:

- JD-VCC must be connected to the 5V power pin (coming from Arduino UNO, or battery, for the Raspberry Pi demonstration).
- VCC is left disconnected.
- GND goes to the GND of the Arduino UNO or battery.
- GND (the second one), is also connected as above.
- IN1 is connected to GPIO pin D4 of the ESP8266.
- IN2 is left disconnected.
- VCC is connected to the 3V3 output power pin of the ESP8266 (or Raspberry Pi).
- NC1 is left disconnected.
- COM1 is connected to the 5V output power pin of the Arduino UNO (or 5V battery).
- NO1 is connected to the resistor.

Finally, the LED is put in the system to simulate how a cooler/heater is turned on/off. A resistor is used to limit the current and protect the LED. Therefore, one node of the resistor is connected to NO1 as already said, and the other node of the resistor to the LED (+ pin). The other pin of the LED, the shortest one, is connected to the GND of the Arduino UNO or battery.

4. Software

4.1 Controller

To support the thermostat logic we have developed a web service, that provides the HTML control page and extra REST api to communicate with the backend. Right now is running locally, but by its very nature, can be made available through the internet.
The web server has been developed using the popular framework spring.
The main reasons behind this choice was the possibility to run the business logic on pretty much whatever machine: good for testing (3 members group, 1 rpi), but also in production. This is possible because spring will provide an instance of tomcat (embedded) and so you will only need java.

4.1.1 What you will need to run the main logic

- java
- python: is used to handle directly connected sensor/actuator (if you want to have them as in our case)
- mosquitto broker: provided by the rpi in our case or by a free internet service
- redis server: same as the mosquitto broker, local on the rpi in our case or as a “free” internet service

4.1.2 How the main logic will run

The java executable file is handled by a system service that will automatically run on startup, as the two python script related to the sensor and the actuator. Chromium will be opened in kiosk mode on the main thermostat page when the java executable will be up and running. Right click are disabled, the usb port and ssh are enabled for debugging, but can be easily closed.

4.1.3 How you can replicate on an other rpi

- install the last raspbian OS from https://www.raspberrypi.org/downloads/raspbian/
- `sudo apt-get update --allow-releaseinfo-change` if there are still problem with the buster release
- `sudo apt-get update` `sudo apt-get upgrade`
- `sudo apt-get install default-jdk`
- `sudo apt-get install mosquitto`
- `sudo systemctl stop mosquitto.service`
- `sudo nano /etc/mosquitto/mosquitto.conf` adding this line at the end:
  ```
  listener 1883
  listener 9001
  protocol websockets
  ```
  Websocket is used to communicate between frontend and backend the sensor data
- `sudo systemctl restart mosquitto.service`
- `sudo apt-get install redis-server`
- `sudo apt-get install python3-dev python3-pip`
- `sudo pip3 install Adafruit_DHT`
- `sudo pip3 install paho-mqtt`
- this complete set of command except `sudo nano /etc/dhcpcd.conf`
  https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md
- `sudo nano /etc/network/interfaces` add
  ```
  auto lo
  iface lo inet loopback
  auto eth0
  iface eth0 inet manual
  allow-hotplug wlan0
  iface wlan0 inet manual
  ```

wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

- cd /home/pi/Documents
- git clone https://github.com/MarcoFlo/Thermostat.git
- cd Thermostat/exe/controller/
- sudo chmod a+x chromium.sh
- sudo cp thermostat_sensor.service /etc/systemd/system/thermostat_sensor.service
- sudo cp thermostat_actuator.service /etc/systemd/system/thermostat_actuator.service
- sudo cp thermostat_controller.service /etc/systemd/system/thermostat_controller.service
- cd /etc/systemd/system
- sudo chmod a+x thermostat_actuator.service
- sudo chmod a+x thermostat_sensor.service
- sudo chmod a+x thermostat_controller.service
- sudo systemctl daemon-reload
- sudo systemctl enable thermostat_actuator.service
- sudo systemctl enable thermostat_sensor.service
- sudo systemctl enable thermostat_controller.service
- sudo systemctl daemon-reload
- sudo nano /home/pi/.config/lxsession/LXDE-pi/autostart

```
@xscreensaver -no-splash
@xset s off
@xset -dpms
@xset s noblank
@unclutter
@sh /home/pi/Documents/Thermostat/exe/controller/chromium.sh
```

- sudo reboot

4.1.4 Business logic

The controller is running an infinite loop that will check each room and set the actuator on/off depending on the mode in which they are:

- manual -> will reach precisely the required temperature
- programmed -> will try to keep the preset temperature +/-0,5 (easy tunable)
- antifreeze -> will keep 10 degree indefinitely
- leave -> will keep the leave temperature specified by the user, but start reaching the programmed temperature before he comebacks, in particular considering an easy tunable 1 degree/hour

We have decided to sacrifice some customizability in favor of a leaner user experience: the daily program is split in 4 time slot, delimited by wake time, leave time, back time, sleep time, each one has an associated temperature. The user can specify one for the weekend and another for midweek days. Each room has its own program.

4.1.5 Connectivity

The rpi is able to connect to a wifi network, but internet connectivity is not needed and it’s able to run in Access Point mode, therefore creating its own network

4.1.6 AWS

We are responding to the notification topic and we are publishing this events:

- new esp (id=10)
- new command to actuator (id=11)
- new sensor data (id=12)

4.1.7 Perceived temperature

Since we have chosen the DHT22 we can use the perceived temperature in our thermostat, providing a more accurate representation of the temperature. The equation used to calculate the perceived temperature is based on the model by Robert G. Steadman, 1994 [6]. The model is the following [6]:

$$AT = Ta + 0.33\rho - 0.70ws - 4.00$$

and with:

$$\rho = rh \cdot 6.105 \cdot exp(17.27 \cdot Ta/(237.7Ta + Ta))$$

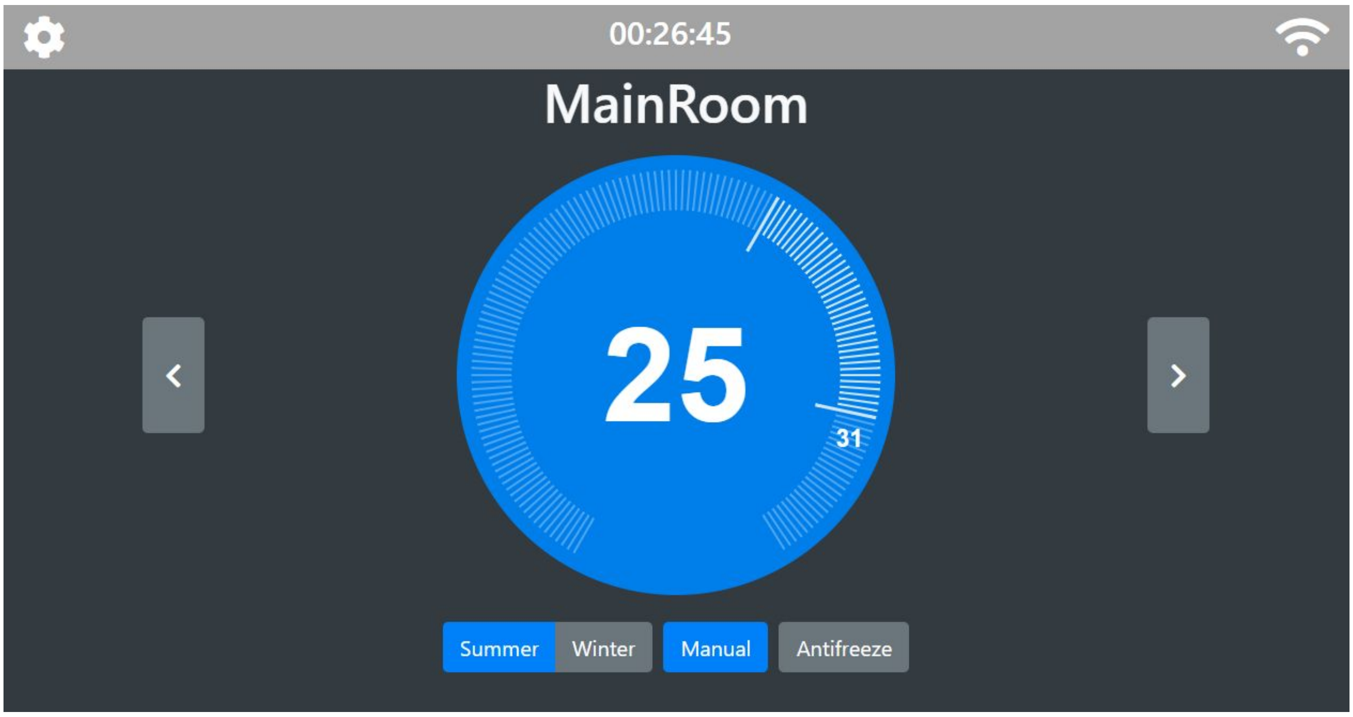
Where:

- AT is the apparent temperature in °C.
- Ta is the measured temperature in °C (also known as dry bulb temperature).
- ρ is the water vapor pressure in hPa.
- ws is the wind speed at 10 meters above ground level, in m/s.
- rh is the measured relative humidity in %.

4.2 Interface

The delivered gif [controllerPL-14.gif] shows the main features of our interface

4.2.1 Main screen

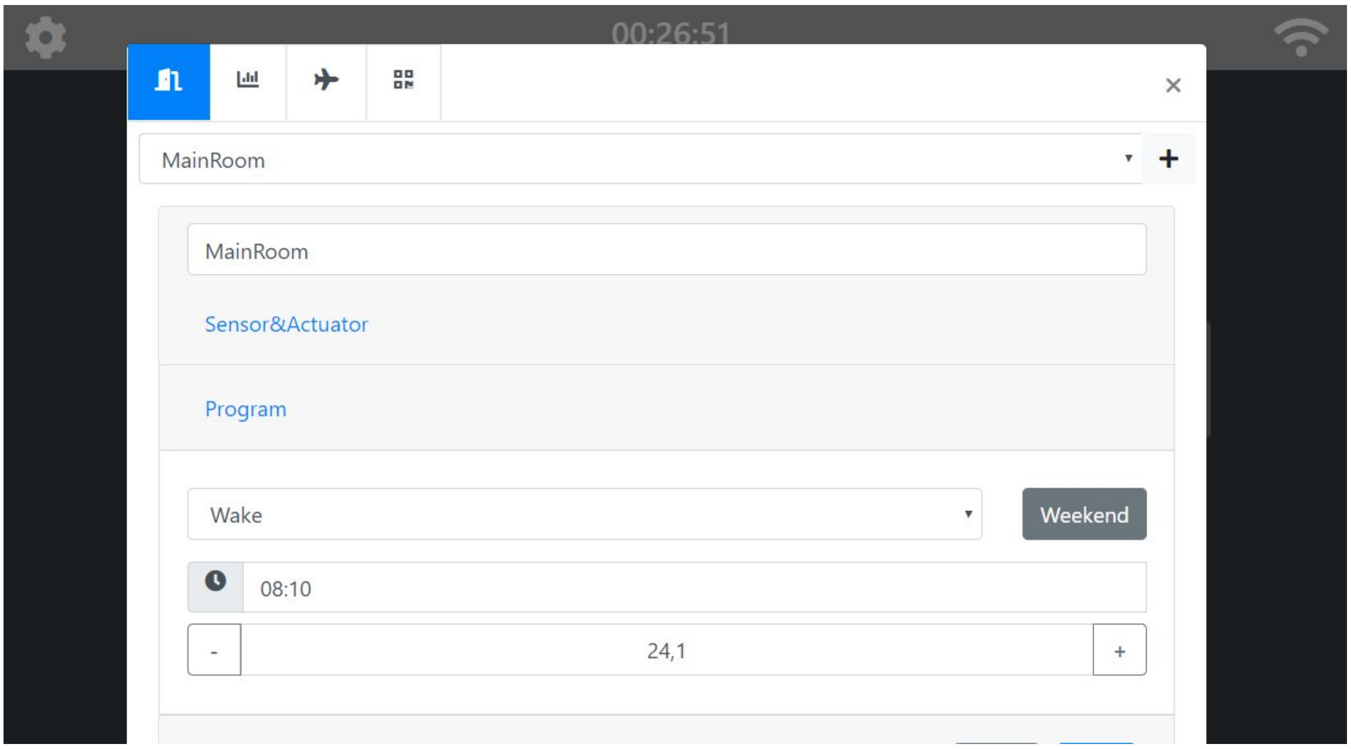


Here you can switch between room and throughout the main thermostat mode:

- summer/winter: depending on which you chose the heater or the cooler won’t be used
- manual: you can choose to specify by yourself temperature or let it decide
- antifreeze: will keep 10 degree indefinitely

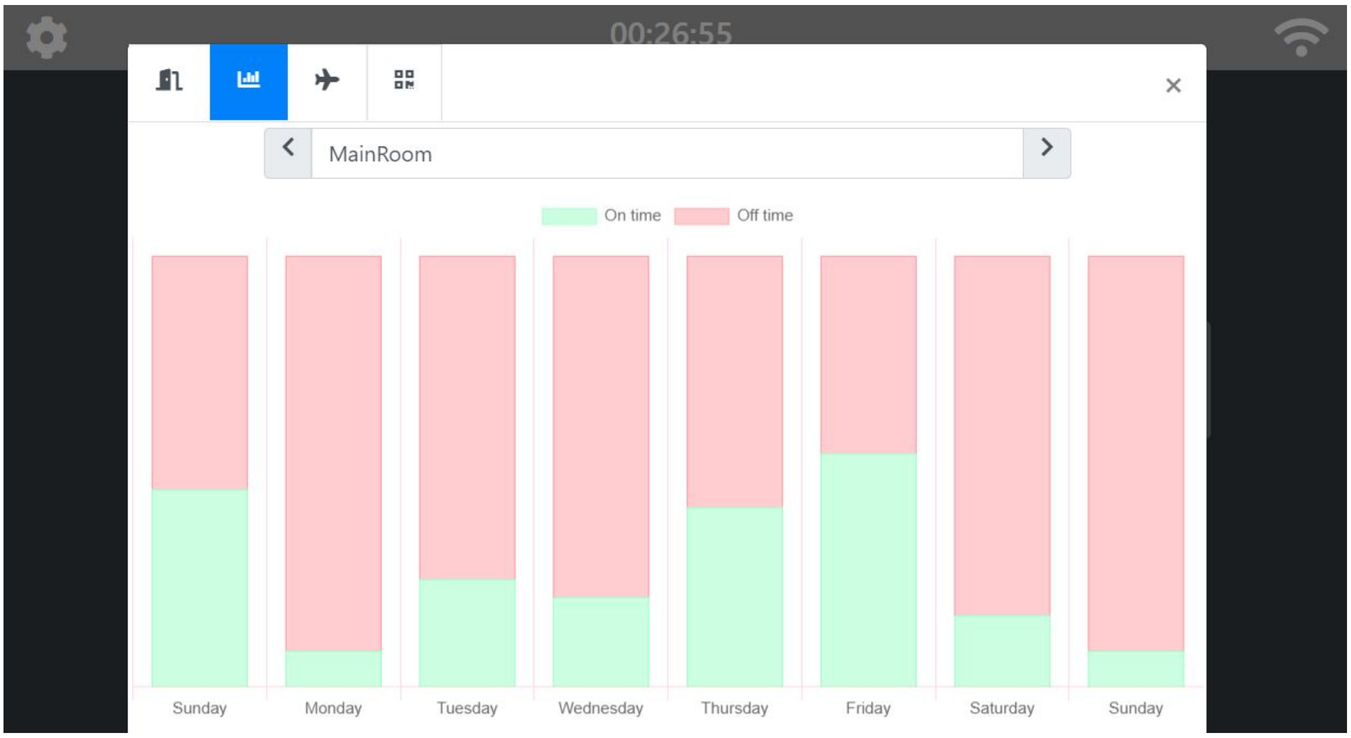
4.2.2 Setting screen

Room program



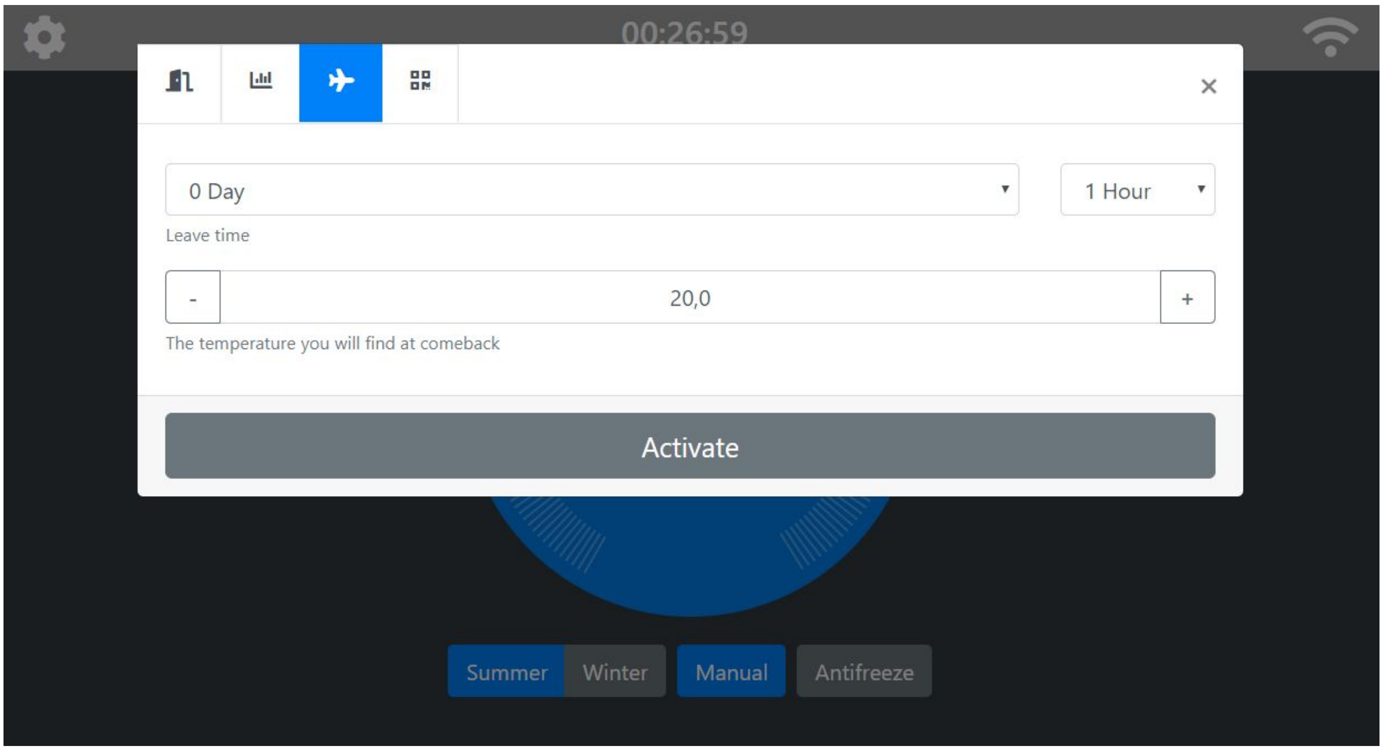
Here you can set up each room with his sensor/actuator and the program the system will follow.

Stats



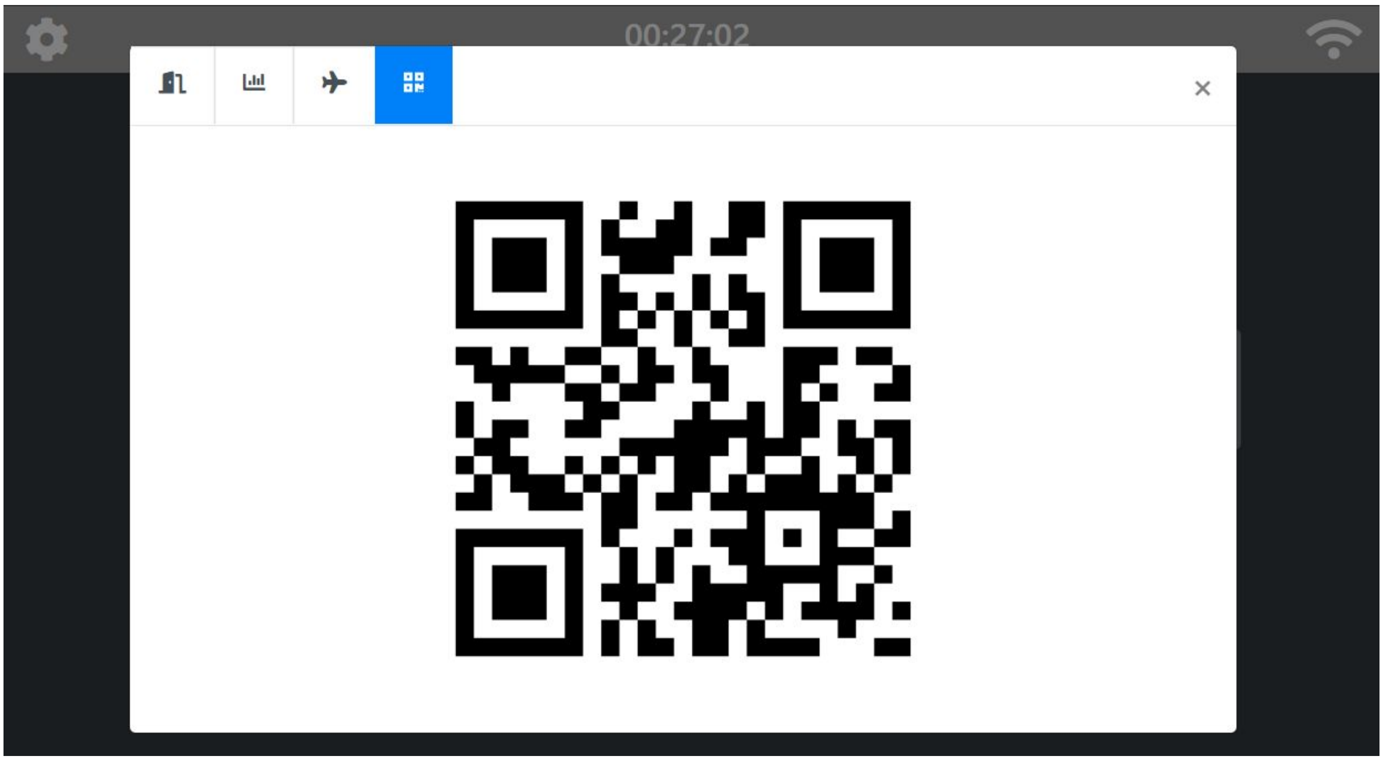
Previous 7 day on-off time

Leave mode



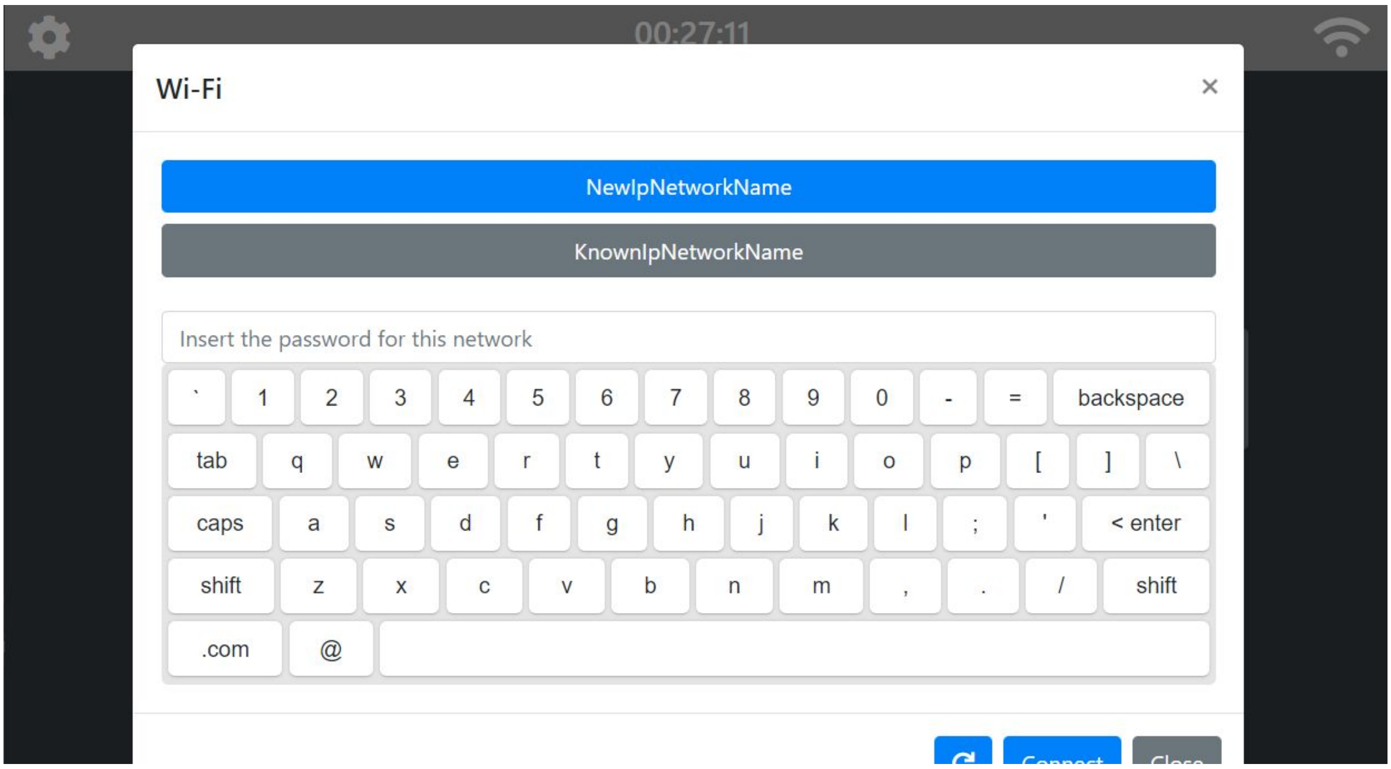
You can specify how much time you will be leaving and the temperature you wish is maintained in the meantime

Remote control



Provide a qr code to control the thermostat from your phone. You will be redirected to the main thermostat page.

4.2.3 Wifi screen


00:27:11
Wi-Fi
×
NewIpNetworkName
KnownIpNetworkName
Insert the password for this network
` 1 2 3 4 5 6 7 8 9 0 - = backspace
tab q w e r t y u i o p [ ] \
caps a s d f g h j k l ; ' < enter
shift z x c v b n m , . / shift
.com @

You can connect to a new net. If the rpi is not connected to any network it will automatically switch to Access Point mode, providing its own network

4.3 Description of ESP8266 sensor and actuator stations

The software for the remote zone control (multi zone management) which is handled by ESP8266s is pretty much the same both for sensor and actuator stations. The only difference is when the ESP connects to the MQTT broker (on the Raspberry Pi, which is the central station with the main sensor and actuator).

The general behavior can be described with the following steps. Naturally, for control flow, there are different flags involved in the code, and several global variables, which are explained in the code files.

1. **Wi-Fi connection.** First, the Wi-Fi network connection is managed. Two things can happen at this stage:
    a. Connect to a new network: if the network is changed, or if it's the first time using a certain Wi-Fi network, the ESP8266 will open up an AP. The user must simply select it, from its computer or mobile phone, to open up an interface to configure the network on the ESP8266.
    b. The ESP8266 will save the credentials (username and password) of the Wi-Fi network it last connected to in the EEPROM. Upon initialization, it will retrieve this data and try to connect to it. If this fails, the AP will appear and the system will wait for the user to configure a network.

2. **EEPROM management.** At this step, the ESP8266 retrieves the last working IP address of the broker (last IP the ESP8266 got a positive reply from, which is 'iamrpi). It then sends a GET request to this IP address and closes the connection after retrieving the payload and HTTP code. There's a specific function we've designed, read_EEPROM(), to handle the data accordingly.
    a. If the response is 'iamrpi', a connection to the MQTT broker is initiated, using this IP address. The code stays in this part until it is possible to establish a connection to the broker. Otherwise, the reset button can be pressed.
    b. If not, control flags are handled so that the IP discovery process is started, to discover the IP of the Raspberry Pi.
3. **IP discovery process.** If the previous process failed (2b), then the IP discovery process is initiated, with the function IP_discovery(). For this, a 16 bit range has been considered, meaning IPs ranging from 192.168.0.0 up to 192.168.255.255. For this purpose, to avoid having exceptions raised by the watchdog timer, it's important to use the yield() function. Basically, in this function, the ESP8266 will need to send GET HTTP

requests to each IP. Once the 'iamrpi' response is received, the IP will be saved in the EEPROM and a flag will be raised to initiate the connection to the MQTT broker. Once again, it will stay here until it can connect to the broker. Both in this case and the previous one (point 2), this blocking connection includes a delay() call to avoid getting exceptions from the ESP8266's watchdog.

4. **Main part.** In this part, the sensor station will be sending data to the broker, or receiving commands from it if it's an actuator station. At each iteration, the client.loop() function is used to maintain and refresh the connection to the MQTT broker. In case the connection is lost, the ESP8266 will attempt to reconnect, but this time in a non blocking fashion.
    a. In the case of a sensor, there's careful management of the strings, using the c_str() function for null termination. There's also a call to the delay() function for 5 seconds after data is sent to synchronize the sensor and the central broker (Raspberry Pi).
    b. In the case of actuators, the command that is received is handled accordingly so that the relay is turned on or off according to the command. A function called callback, which checks the received messages under a certain topic, is passed on to client.setCallback() of the PubSubClient library for the same purpose.

In this algorithm, which is briefly explained above, there are some important aspects to emphasize:

- These main steps are part of the void loop(), and call the functions they need.
- If at any point (IP discovery, main part, etc), the Wi-Fi network connection is lost, control flow will be redirected towards Wi-Fi management (this is the priority). There's a function for doing this, called reconnect_esp(), which is called in the main part and the IP discovery process.
    - This function checks the state of the connection. If it's lost, then 30 seconds will be given to the user to reset the network, which can be increased (assuming there was, for instance, a modem failure) otherwise a new network will need to be configured (can happen if the Wi-Fi changes password, or the old network is replaced).
    - In this case, control flow will be redirected to the EEPROM management part, so that it will attempt to connect to the last working IP, otherwise the IP discovery is initiated.
- When failing to connect to the MQTT broker, the failing code is reported, similarly for Wi-Fi.

Next, a small table, Table 4.1, is shown to detail how the MQTT topics work for the sensor and actuator ESP8266 stations. Referring to the table, the string for the id of the sensor station was set to "esp2sensor" and for the actuator station to "esp1actuator" for the demonstration.

Finally, it's important to note that some code examples were considered as basis for the PubSubClient, WiFi Manager and DHT libraries. These were only taken as reference/guide and so small parts were modified accordingly to develop the code for the project. These references can be found in [2, 3, 7] and in Arduino IDE (WiFiManager AutoConnect example, EEPROM examples). In the case of the HTTP, the API was looked up online, as well as some examples on how to send a GET request. Libraries were installed using Arduino IDE (Library manager) or downloaded from Github.

Table 4.3.1. Actuator and sensor ESP8266 stations: MQTT management

TOPIC	PAYLOAD	PURPOSE
/esp8266/id-esp	sensor heater cooler	When an ESP8266 is turned on, it needs to publish this to the broker
/id-esp/sensor	"temperature"_"humidity"	ESP publishes, RPi subscribes
/id-esp/actuator	on off	ESP subscribes, RPi publishes

5. References

[1] https://www.hometree.co.uk/energy-advice/central-heating/how-does-a-thermostat-work.html
[2] https://lastminuteengineers.com/esp8266-dht11-dht22-web-server-tutorial/
[3] https://www.instructables.com/id/Interface-Relay-Module-With-NodeMCU/
[4] https://steps2make.com/2017/10/arduino-5v-relay-module-ky-019/
[5] https://arduino.stackexchange.com/questions/36330/how-to-make-a-5-volt-relay-work-with-nodemcu
[5] https://www.amazon.co.uk/2-channel-trigger-module-JQC-3FF-S-Z-RBTMKR-x/dp/B01NAJ67ML
[6] https://www.vcalc.com/wiki/rklarsen/Australian+Apparent+Temperature+%28AT%29
[7] https://iotdesignpro.com/projects/how-to-connect-esp8266-with-mqtt