Internet of Things - 089073

Home Automation and Monitoring

Angelo Claudio Re
Giovanni Scotti

Professor: *Matteo Cesana*

Academic Year 2017/2018

Document version: 1.0

Contents

Section 1

Introduction

1.1 Purpose

This document is intended to describe a *Home Automation and Monitoring* system, its electronic components as well as its constraints and the interaction with the real world and the end user. A useful example of work environment will be provided to clarify the scope of the system itself and how it can be deployed.

The documentation is full of diagrams, pictures and schematics to let the final user easily implement the infrastructure and take advantage of its countless features.

This document is mainly addressed to computer scientists, IoT enthusiasts, makers and anyone, with some electronics and programming skills, who wants to keep an eye on his/her home and to control devices remotely.

1.2 Scope

The *Home Automation and Monitoring* system aims to offer a smart solution to home automation and monitoring needs. It is intended for those kind of users who want to visualize information about their home, such as temperature or humidity, and control devices remotely.

The system consists mainly of:

- a *back-end*, which is in charge of managing MQTT messaging protocol and runs Node-RED. Data coming from boards provided with sensors and actuators are gathered and sent to a remote service provider to store information into the cloud and to open up further analysis.

- a *front end*, which is offered by a user-friendly and web-accessible Node-RED dashboard that can be easily reached from any device either connected to the same network or to the Internet by means of port forwarding.

The system must be secure. This is why authentication is required in order to access the dashboard. Moreover, MQTT must be secured too and clients provide credentials to the broker to join the network. Last but not least, power consumption has been taken into account: battery-powered boards can enter a deep sleep state after publishing data.

1.3 Definitions, Acronyms, Abbreviations

- **ADC:** Analog to Digital Converter.
- **AP:** wireless Access Point, it allows Wi-Fi devices to connect to a wired network. Usually it is an integral component of routers for home or office use.
- **Back-end:** any device or computer program that remains in the background and offers application logic and communication interfaces to work with the front-end counterpart. It can provide a data access layer.
- **Front-end:** any part of a system the users directly interact with. It provides the so called presentation layer.
- **IoT:** Internet Of Things.
- **LAN:** Local Area Network.
- **MQTT:** Message Queuing Telemetry Transport, a publish-subscribe messaging protocol.
- **System:** the entire infrastructure and all the devices involved in the *Home Automation and Monitoring* project.
- **SoC:** System-On-Chip, an integrated circuit that contains various electronic components designed to work together to achieve a common goal.
- **UI**: User Interface.

Section 2

System Implementation

2.1 Architecture Overview

The overall system architecture is depicted by Figure 2.1.

To begin with, the whole system mainly operates within the user Local Area Network. A wireless router is the central point of the entire infrastructure because all the involved devices are connected to it via wireless communication.

The Raspberry Pi board is the backbone of the back-end processing activities. It is in charge of running Node-RED and acts as MQTT broker. It collects information that are sent out by several WiFi-capable modules taking advantage of the publish-subscribe MQTT connectivity protocol. Each module is provided with sensors that gather information about the surrounding environment.

ESP-12E and ESP-32 NodeMCU are the IoT WiFi boards that have been chosen to design the infrastructure. There exists a great variety of sensors that can be hooked up to these ESP modules and ease of programming via Arduino IDE makes them an appropriate choice too. Moreover, they offer full TCP/IP stack support. Each board does not only publish data, but can also subscribe to a specific topic and receive remote commands to control actuators, such as relays.

The end user is able to visualize a user-friendly, web-accessible dashboard directly from his/her own devices, which must be in the same network of the other system components, in case port forwarding is disabled. The interface is provided by Node-RED development tool. Furthermore, data are forwarded to ThingSpeak by the Raspberry Pi board in order to log them in a remote database and to enable further analysis: ThingSpeak has integrated support from the numerical computing software MATLAB.

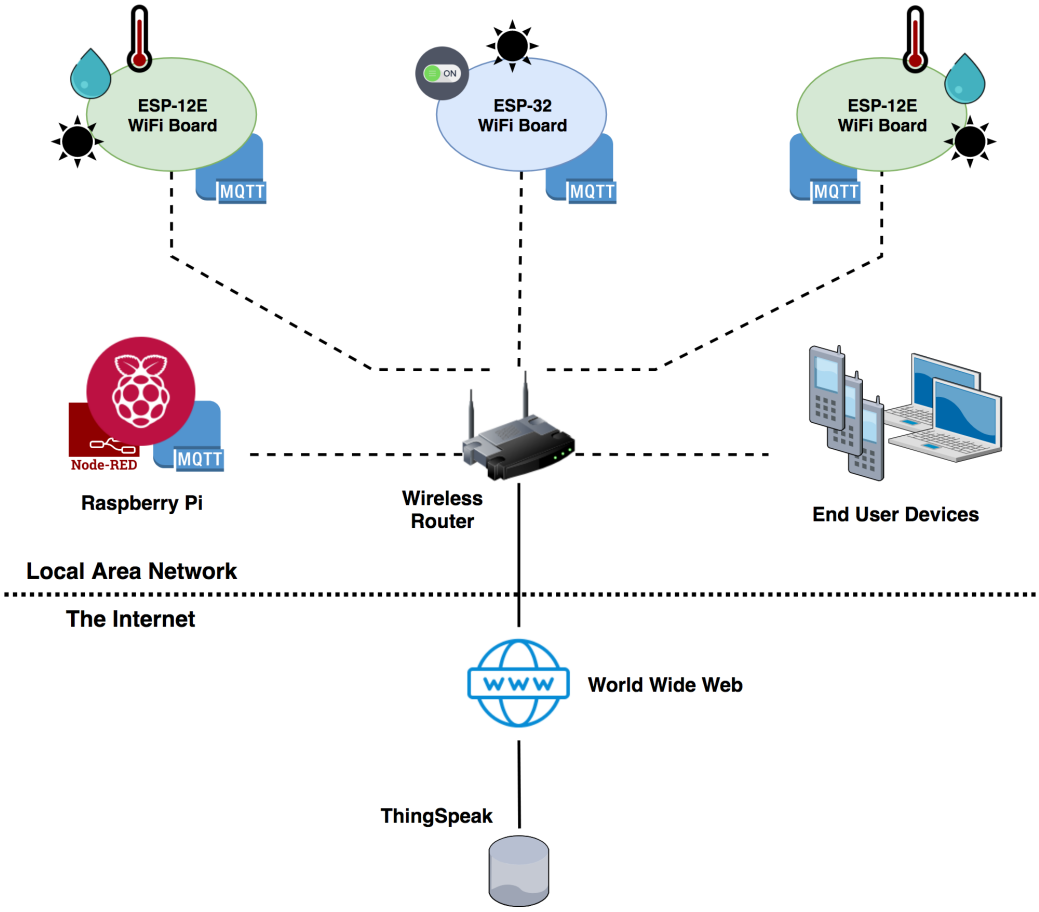

Figure 2.1: System architecture overview. Dashed lines represent wireless communication, solid lines stand for wired connections.

2.2 Configuring and Programming the RPi

The Raspberry Pi is a tiny single-board computer featuring a Broadcom System-On-Chip with an integrated ARM compatible processor. There exist several different generations and models and its role in the system will suit them all. Anyway, using a Pi 3 Model B is highly recommended: it has a quad-core processor and on-board WiFi, Bluetooth and USB boot capabilities which do not require any extra module.

The Raspberry Pi board needs a fresh installation of *Raspbian Stretch*, the official Debian-based Linux operating system developed by the Raspberry Pi Foundation. Other third-party operating systems are not taken into account by this document.

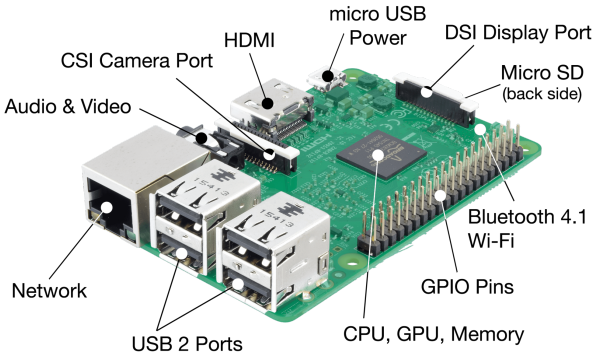

Figure 2.2: Location of connectors and main integrated circuits on the Raspberry Pi 3 Model B.

The board is used as *headless computer*, a device that has been configured to operate without a monitor, keyboard and mouse. The command line is accessed remotely from another computer on the same network using SSH. Since Raspbian has the SSH server disabled by default, it needs to be manually enabled from the desktop or in the terminal via `raspi-config`.

Setting up a static IP address for the Raspberry is strongly recommended. It can be assigned in the DHCP context menu of the wireless router. Since the procedure is different on any device, the router handbook should be referred to.

Before proceeding with the following sections, all the software must be up-to-date. This straightforward operation is carried out by executing the commands below in a terminal window:

```
sudo apt-get update && sudo apt-get upgrade
```

2.2.1 MQTT

MQTT is a lightweight messaging protocol that works on top of TCP/IP. It provides resource-constrained network clients with a simple way to distribute and exchange information. MQTT is based on the publish-subscribe messaging pattern that requires a message broker to work properly: in this case the role perfectly fits the Raspberry Pi board.

Information about a given topic are sent, or published, to a server that behaves as MQTT message broker. Then, the latter pushes the information out to those clients that have previously subscribed to the above-mentioned topics.

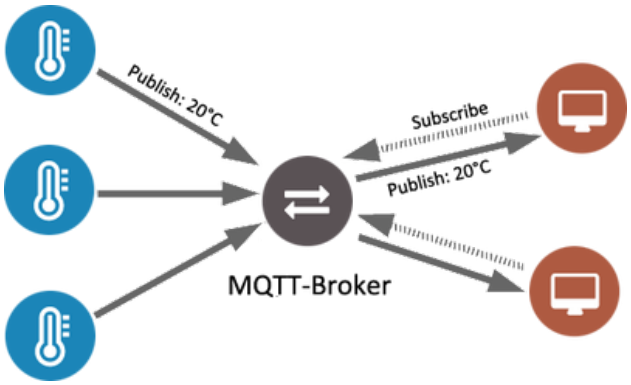

Figure 2.3: The message broker is the central point of the MQTT infrastructure. It receives published information and forwards it to subscribed clients.

MQTT offers three quality of service (QoS) levels to guarantee a certain degree of performance to a data flow:

- **Level 0.** At most once delivery: this is the minimal level and guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored by the sender.
- **Level 1.** At least once delivery: all the messages will be delivered at least once to the receiver. This is the QoS level used in the *Home and Automation Monitoring* system because the overhead is lower than QoS 2 and it guarantees the message arrives at least once. Duplicates are not a problem at all because the user interface would be updated one or more time with the same piece of information and this would be completely unnoticeable by the end user.

- **Level 2.** Exactly once delivery. It is the safest and the slowest quality of service level.

2.2.2 Mosquitto

Mosquitto is an open source message broker that implements the MQTT protocol. It is suitable for both low-power single board computers and powerful servers.

To install the broker, the following command must be run from the Raspberry Pi terminal:

```
sudo apt-get install mosquitto
```

The broker will automatically start whenever the Raspberry is powered up. The next step is adding users and passwords to secure the MQTT connection. The current directory must be `/etc/mosquitto` while typing the successive commands:

```
// Add the first user
sudo mosquitto_passwd -c passwordfile user

// Add another user to the existing password file
sudo mosquitto_passwd -b passwordfile other_user password
```

The following two lines must be added to the `mosquitto.conf` file.

```
allow_anonymous false
password_file /etc/mosquitto/passwordfile
```

As far as this system is concerned, just a single user has been created to manage the MQTT connection: username is `admin` and password is `hamrpi`. These credentials are used to set up MQTT by Node-RED and any ESP node.

2.2.3 Node-RED

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to combine flows using the wide range of nodes in the palette, which can be deployed to its runtime in a single click.

Installation

Node-RED comes pre-installed on the full Raspbian OS image that can be downloaded from *raspberrypi.org* website.

In case Node-RED is not already installed on the user Linux-like operating system, the Node-RED upgrade script should be executed:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red
/raspbian-deb-package/master/resources/update-nodejs-and-nodered)>
```

It is also necessary to install another core component to manage authentication and secure access. From the Raspberry Pi terminal it is needed to run the following commands:

```
cd ~/.node-red
sudo npm install -g node-red-admin
```

Finally, from the Node-RED development environment, it is necessary to install a plug-in to provide the user with an interactive and awesome dashboard that sums up all the collected data from sensors.

From the Node-RED settings, click on the *Manage Palette* menu entry and get the option to install extensions on the *install* tab. At this point, proceed with the installation of `node-red-dashboard` extension.

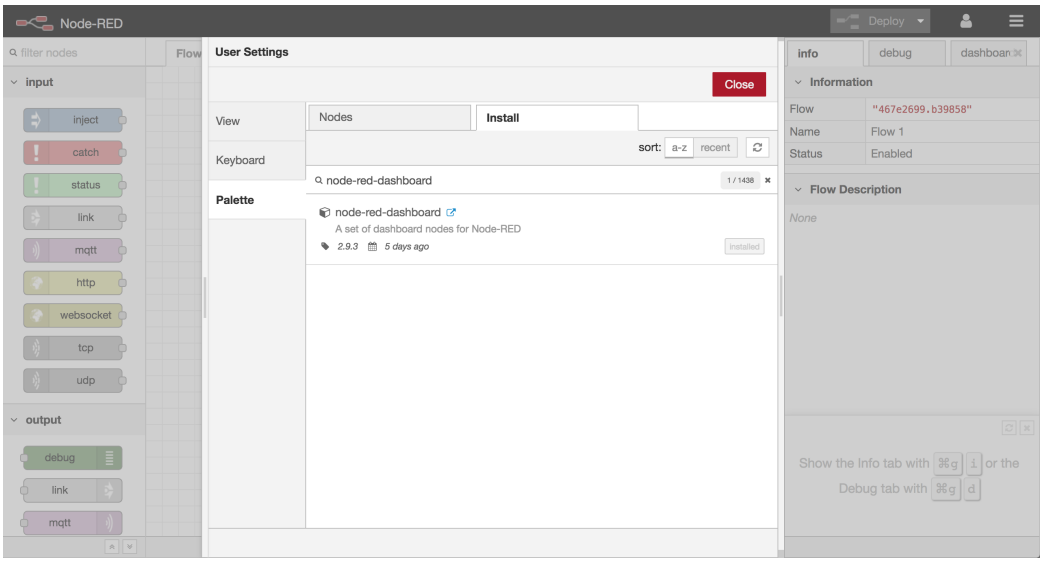

Figure 2.4: Node-RED Manage Palette. *Install* tab has been entered to proceed with the extension installation.

Security

Once `node-red-admin` has been installed, secure access must be properly configured. To begin with, a password hash has to be created. It will be saved in the Node-RED configuration file `setting.js`. This operation can be simply performed by running:

```
node-red-admin hash-pw
```

After that, insert the chosen password and copy the related hash, as in Figure 2.5.

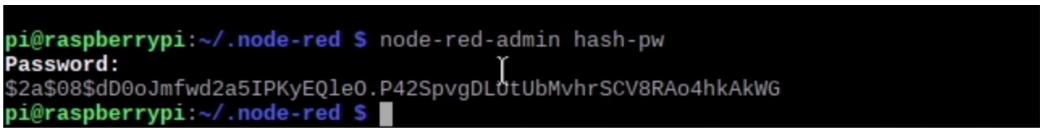

Figure 2.5: Hash password generation example.

The second step is to modify the configuration file `settings.js` adding a user and the previously generated password hash. The file can be opened by typing in the terminal:

```
nano ~/.node-red/settings.js
```

Users have to be added to a list using JSON syntax. Proceed as shown in Figure 2.6 to add a user whose username is *admin* and whose password corresponds to the pasted hash.

```
// Securing Node-RED
// -----------------
// To password protect the Node-RED editor and admin API, the following
// property can be used. See http://nodered.org/docs/security.html for details.
adminAuth: {
    type: "credentials",
    users: [{
        username: "admin",
        password: "$2a$08$yUY.89AhwvZUViC6dSnxuuBS6RFci2I7bloOql3gfhK5lHGYeunoO",
        permissions: "*"
    }]
},

// To password protect the node-defined HTTP endpoints (httpNodeRoot), or
// the static content (httpStatic), the following properties can be used.
// The pass field is a bcrypt hash of the password.
// See http://nodered.org/docs/security.html#generating-the-password-hash
httpNodeAuth: {user:"admin",pass:"$2a$08$yUY.89AhwvZUViC6dSnxuuBS6RFci2I7bloOql3gfhK5lHGYeunoO"},
//httpStaticAuth: {user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN."},
```

Figure 2.6: Node-RED `setting.js` file. Secure access enabling.

To start Node-RED automatically on power up and reboot the Raspberry Pi board, execute the next commands:

```
sudo systemctl enable nodered.service
sudo reboot
```

After the system has been rebooted, whenever Node-RED is accessed for the first time, it must ask for valid user credentials.

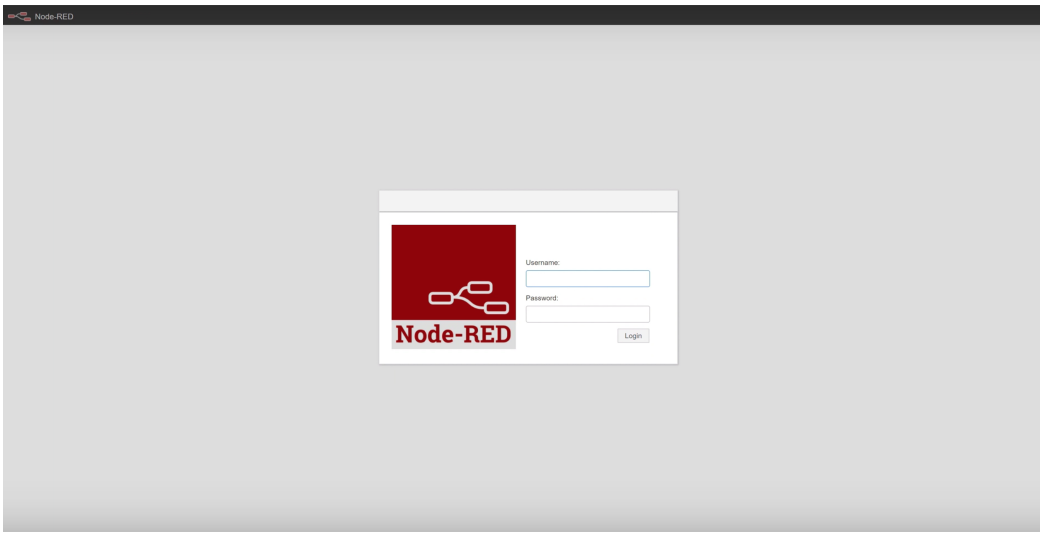

Figure 2.7: Node-RED login screen.

Flow building blocks

The main building blocks used in the development environment to create the dashboard and provide MQTT connection are the following:

- **MQTT**: taken from *input* and *output* menus.
  These nodes are used to manage the connection with the MQTT broker installed on the Raspberry Pi to get and send data based on the subscriptions and publications of the ESP nodes.

- **GAUGE**: taken from the *dashboard* menu.
  This node allows to display the current value measured from a certain sensor with different types of representation (i.e. gauge, donut, compass or level) in the dashboard.

- **CHART**: taken from the *dashboard* menu.
  This node is used to show the most recent data captured by sensors in the dashboard.

- **SWITCH**: taken from the *dashboard* menu.
  This node is responsible for managing the status of the relay that the ESP-32 board is equipped with.

Dashboard

The dashboard is basically a UI that neatly displays data. Desktop version for large screens is depicted by Figure 2.8 while mobile version designed for smaller screens is shown in Figure 2.9.

The dashboard can be accessed at the same address of the development environment plus `/ui` at the end:

```
http://{Raspberry Pi IP address}:1880/ui
```

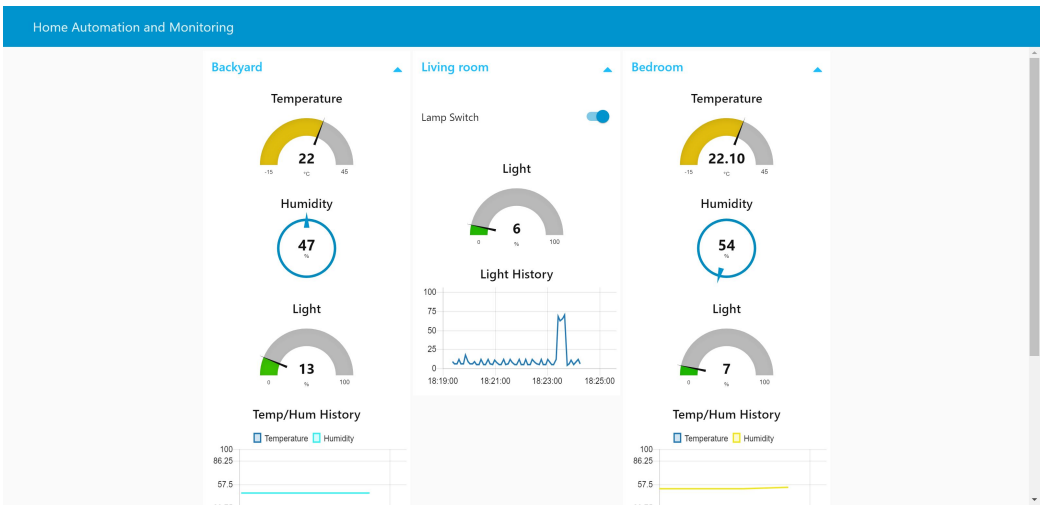

Figure 2.8: Node-RED dashboard: desktop version.

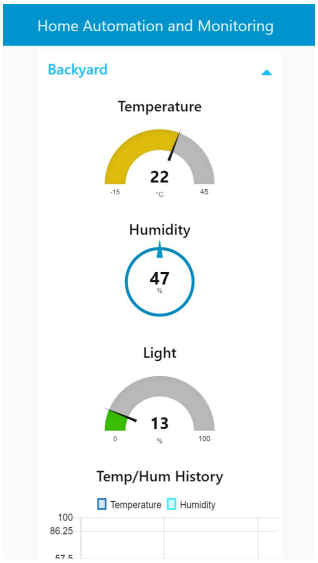

Figure 2.9: Node-RED dashboard: mobile version.

2.2.4 ThingSpeak

ThingSpeak is an IoT platform that allows to collect and store sensor data in the cloud. It provides applications that make possible to analyze and visualize user data in MATLAB. Sensor data can be sent to ThingSpeak from Arduino, Raspberry Pi, BeagleBone Black, and many other boards equipped with a microcontroller and Internet connectivity.

Channel setup

Once logged in ThingSpeak, create a new channel and set it up as shown in Figure 2.10. Every field is linked to a different sensor and collects all the data sent by it.

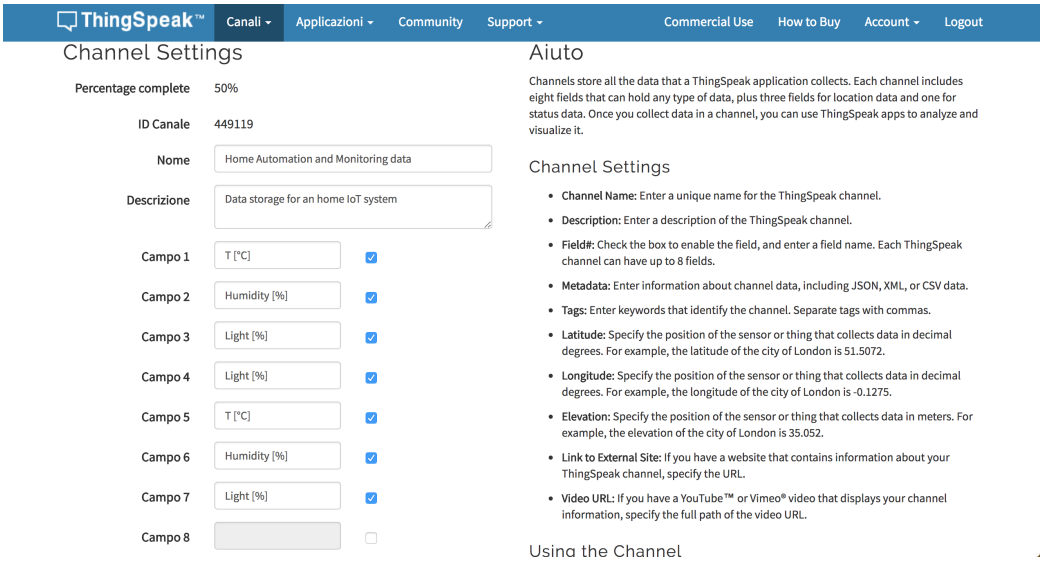

Figure 2.10: ThingSpeak channel setup.

After this, copy the API key that is required to send data to ThingSpeak from Node-RED. In more detail, the *Update a Channel Feed* URL must be used to perform HTTP get requests.

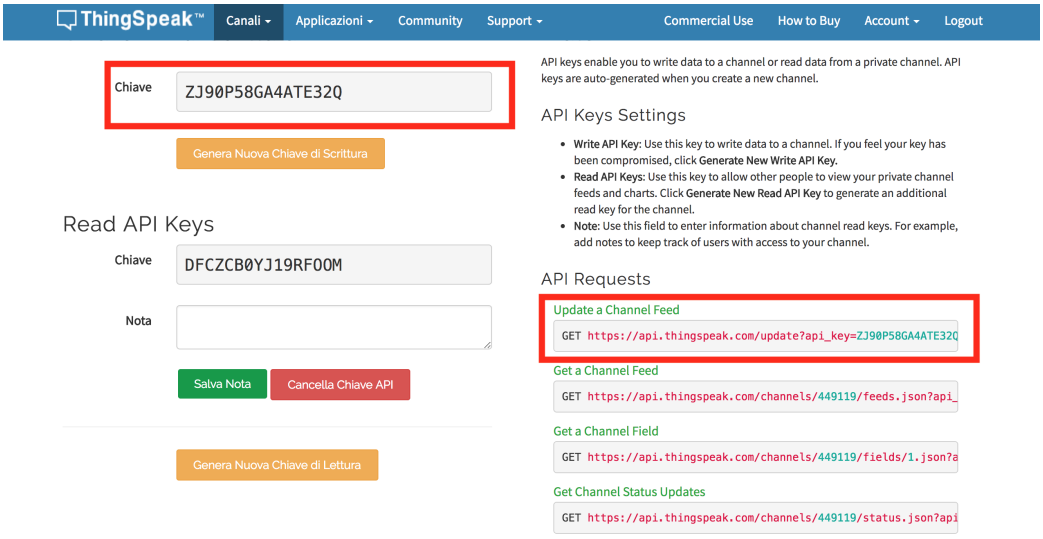

Figure 2.11: ThingSpeak APIs.

Node-RED communication

The blocks used to enable the communication with ThingSpeak are the following:

- **FUNCTION**: taken from the *function* menu.
  This node is used to setup the HTTP get request and it is configured with the API key taken from ThingSpeak.
- **HTTP REQUEST**: taken from the *function* menu.
  This node is responsible to perform the HTTP get request towards ThingSpeak with the parameters passed by the function block.

ThingSpeak charts

Once some data samples have been received, this is how ThingSpeak shows them.

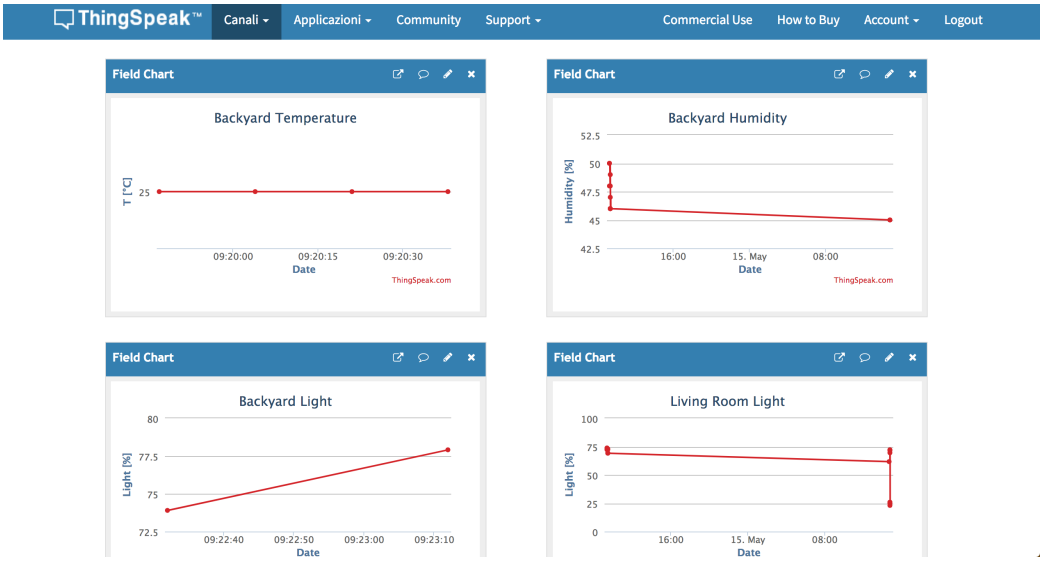

Figure 2.12: ThingSpeak charts and UI.

2.3 Getting Started with ESP Boards

The *Home Automation and Monitoring* system makes use of ESP modules to collect data from sensors and transmit them to the Raspberry Pi taking advantage of MQTT protocol.

Two kinds of boards have been selected to play the role of MQTT client; namely NodeMCU ESP-12E and ESP-32 development kits. The former is based on the widely explored ESP8266 System-On-Chip, which combines WiFi and microcontroller capabilities. The latter utilizes an ESP32 SoC that is very similar, but far more powerful: it has been designed to provide robustness, versatility and reliability in a wide variety of applications.

Both the boards can be easily programmed using the Arduino IDE after installing the appropriate cores via Arduino Boards Manager. The core installation will not be covered by this document.

Please notice that there is not a particular reason to use one specific board or the other: they both can fulfill the requirements of the system. The usage of different boards creates heterogeneity and variety, which are always present in the real world.

2.3.1 ESP-12E: Sensors, Wiring and Code

An accurate picture of the board pinout is shown in Figure 2.13. The ESP8266 module requires 3.3V power supply. Luckily the board provides a voltage regulator and can be connected to 5V using microUSB connector or *Vin* pin.

There is a number of two ESP-12E boards involved in the system. They both have the same configuration, sensors and wiring, but can be deployed in two different areas. In more details, each board takes advantage of temperature, humidity and light sensors.

Status LEDs

Three LEDs give information about the status of the board:

- **Red**: the board is trying to get WiFi access.
- **Yellow**: the board is setting up MQTT connection. The MQTT broker must be up and running to succeed.
- **Green**: the board is active and it is sending sensor data to the MQTT broker or listening to messages related to its subscriptions.

Temperature and humidity sensor

DHT11 and DTH22 are low cost temperature and humidity sensors. They are made of two parts: a thermistor and a capacitive humidity sensor. A small chip inside performs analog to digital conversion to let any microcontroller read the digital output signal.

Even though the two DHT sensors are interchangeable, DHT22 is more accurate and covers a larger range of temperatures.

Since the system involves two ESP-12E boards, both the sensors are used for the sake of completeness. Moreover, the fact that it is fairly easy to connect up to them, as shown in Figure 2.14, the low cost and ease of interfacing make these sensors an appropriate choice for the system.

Light sensor

A light sensor can be used to detect the current ambient light level, in particular the presence or absence of light. There exist several types of light sensors: photoresistors, phototransistors, photodiodes...

The chosen approach is making use of a photoresistor, taking advantage of the ADC provided by the ESP boards.

A photoresistor changes its resistance based on the amount of light that falls upon it: the more the light, the less the resistance. A simple voltage divider can be used to give an input to the ADC, as illustrated by Figure 2.14.

Deep sleep

ESP8266, the core of ESP-12E boards, supports sleeping. While sleeping, the device draws much less power than while awake: this could be a good approach to save power when batteries are the only source of electricity.

Deep sleep is one out of four supported types of sleep mode: everything is off but the Real Time Clock. This is the most power efficient option.

Pin D0 (GPIO 16) needs to be connected to RST to wake up the device when deep sleep is over: whenever RST receives a LOW signal, it restarts the microcontroller. Once the device is in deep sleep mode, a LOW signal will be sent to GPIO 16 as soon as the timer is up.

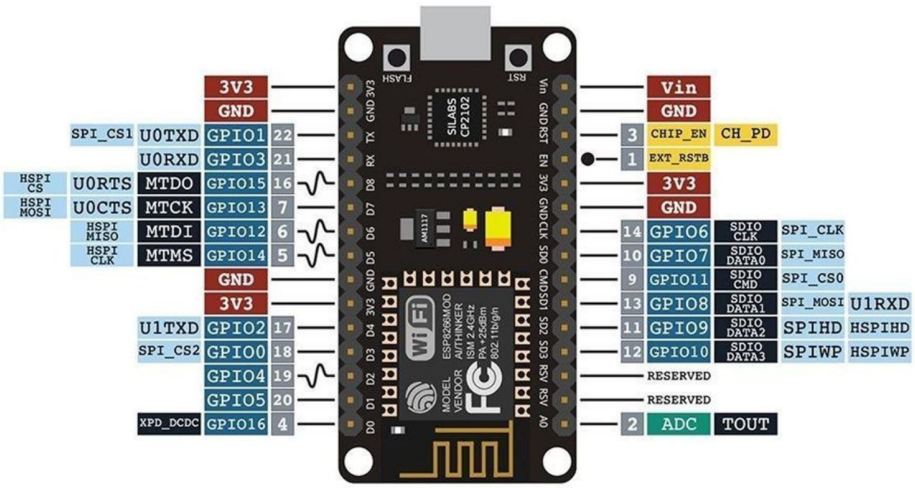

Figure 2.13: ESP-12E pinout.

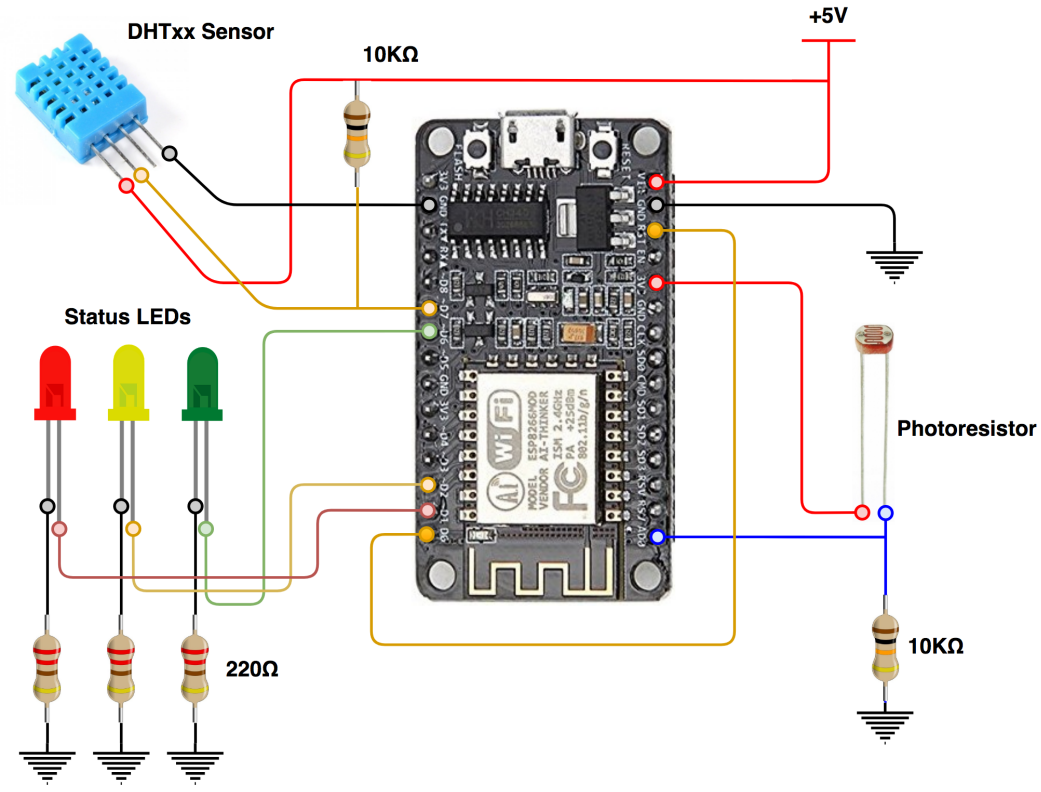

Figure 2.14: ESP-12E schematic with sensors and status LEDs. Notice the wire to tie D0 to RST pin to enable deep sleep.

Code

The main parts of the code are described by the following tables. In more detail, libraries are reported by Table 2.1, define preprocessor directives by Table 2.2, global variables by Table 2.3 and, lastly, functions by Table 2.4.

Library	Description
`ESP8266WiFi.h`	Required to enable WiFi connectivity.
`DHT.h`	Required to interface with DHT sensors.
`Adafruit_MQTT.h` `Adafruit_MQTT_Client.h`	Required to take advantage of MQTT protocol.

Table 2.1: Libraries involved in the sketches for ESP-12E boards.

#define Directive	Description
`DEBUG`	Enables debug over serial communication.
`WLAN_SSID`	Name of the WiFi network to connect to.
`WLAN_PASS`	Password to join the WiFi network.
`HOST`	MQTT broker IP address.
`PORT`	MQTT broker port.
`USERNAME`	Credentials to set up the MQTT connection.
`PASSWORD`	Credentials to set up the MQTT connection.
`SLEEP_TIME`	Deep sleep duration.
`CONN_ATTEMPTS`	Number of attempts to get WiFi connection before entering deep sleep mode.
`MQTT_ATTEMPTS`	Number of attempts to connect to MQTT service provided by the broker.
`DHT_TYPE`	Type of DHT sensor.
`DHT_PIN`	Digital pin to read data coming from DHT sensor.
`LED_RED`	GPIO that controls the red led.
`LED_YELLOW`	GPIO in charge of controlling the yellow led.
`LED_GREEN`	GPIO that controls the green led.

Table 2.2: Define directives specified in the sketches for ESP-12E boards.

Global Variable	Description
`client`	This object is instantiated to enable WiFi connectivity.
`mqtt`	This object handles MQTT connection with the broker.

`temperature`
`humidity`
`light`
	Objects of type Adafruit_MQTT_Publish in order to send data to the broker. They are initialized with the appropriate topic depending on the node that collects information.

`dht`	Object that represent the DHT temperature and humidity sensor. It is instantiated specifying the type of sensor and the input pin on the board.

Table 2.3: Global variables declared in the sketches for ESP-12E boards.

Function	Description
void `setup()`	Calls functions to establish WiFi connection and set up MQTT. It publishes data and enters deep sleep state.
void `loop()`	This function is not required since deep sleep is enabled. Every time the ESP-12E board restarts, the setup function is executed and deep sleep state is entered.
void `enterDeepSleep()`	Turns off LEDs and WiFi and enters deep sleep state.
void `MQTT_connect()`	Connects to MQTT service provided by the MQTT broker.
void `ledOff()`	Turns off the status LEDs.
void `ledRed()`	Turns on the red LED.
void `ledYellow()`	Turns on the yellow LED.
void `ledGreen()`	Turns on the green LED.

Table 2.4: Functions used in the sketches for ESP-12E boards.

2.3.2 ESP-32: Sensors, Wiring and Code

An accurate picture of the board pinout is shown in Figure 2.15. As for ESP8266, also ESP32 requires 3.3V to operate. The board is provided with a voltage regulator and can be powered up by an external power source through USB connection or *Vin* pin.

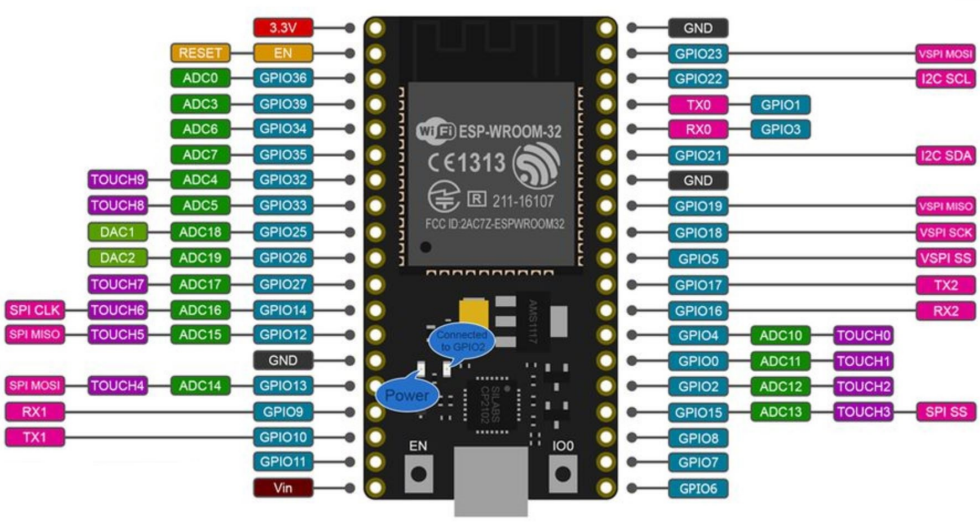

Figure 2.15: ESP-32 pinout. Notice the presence of multiple ADC pins, in contrast with the only one offered by ESP-12E boards.

Just a single board of this type is involved in the system. It is hooked up to a light sensor and to a relay module. The board does not take advantage of deep sleep functionality because it needs to be always up and running in order to reduce the latency to switch on and off the relay.

Status LEDs

Three LEDs fit out information about the status of the board. They work exactly as for ESP-12E boards: make reference to subsection 2.3.1.

Light sensor

A photoresistor and a voltage divider are used to provide light intensity information. The operating principle is the same as for ESP-12E boards: please, make reference to subsection 2.3.1.

Relay module

A relay is an electrical switch operated by an electromagnet. Relays are usually used to control a high voltage circuit with a low power one. A microcontroller can drive relays with one of its digital output pins: this is exactly what ESP32 is used for.

The advanced WiFi connectivity of the board gives the possibility to remotely control the digital pin hooked to the relay module turning it into a remote switch.

As mentioned earlier, in order to reduce the latency to turn on and off the relay, the module is always active and does not enter any sleep state.

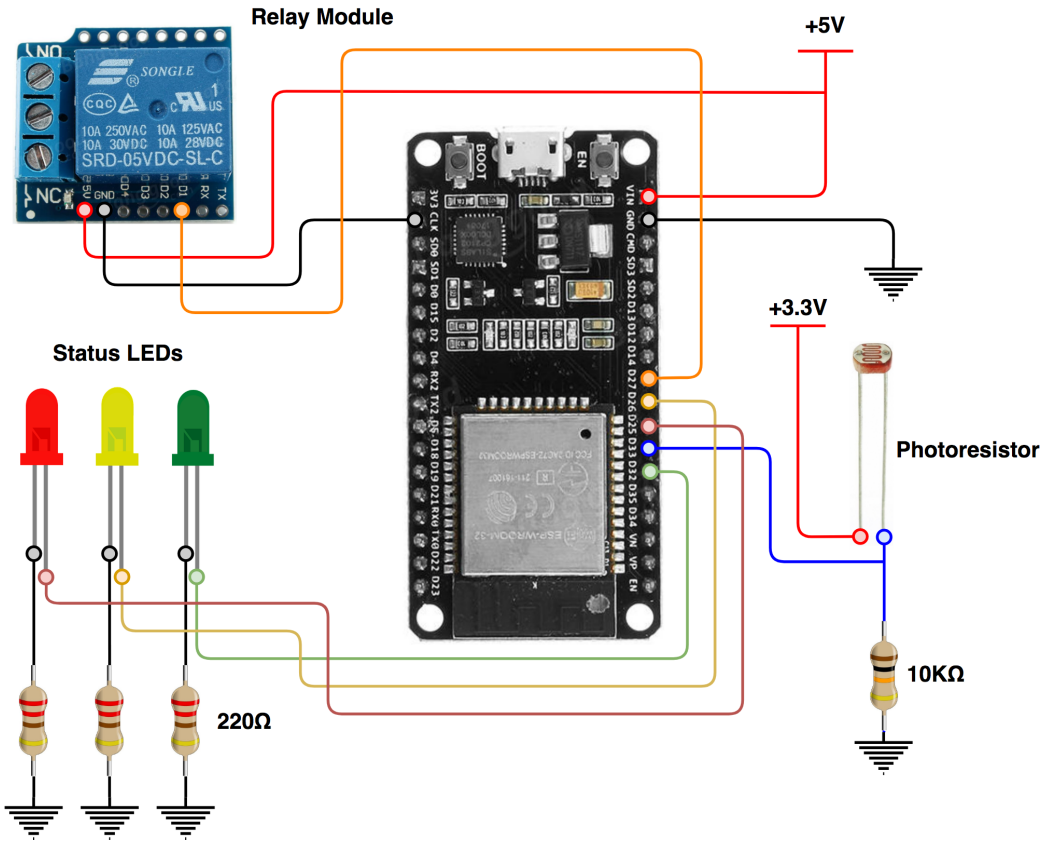

Figure 2.16: ESP-32 schematic with sensors, actuators and status LEDs.

Code

As for ESP-12 boards, the main parts of the code are described by the following tables. In more detail, libraries are reported by Table 2.5, define preprocessor directives by Table 2.6, global variables by Table 2.7 and, lastly, functions by Table 2.8.

Library	Description
`WiFi.h`	Required to enable WiFi connectivity.
`Adafruit_MQTT.h` `Adafruit_MQTT_Client.h`	Required to deal with MQTT protocol.

Table 2.5: Libraries involved in the sketch for ESP-32 board.

#define Directive	Description
`DEBUG`	Enables debug over serial communication.
`WLAN_SSID`	Name of the WiFi network to connect to.
`WLAN_PASS`	Password to join the WiFi network.
`HOST`	MQTT broker IP address.
`PORT`	MQTT broker port.
`USERNAME`	Credentials to set up the MQTT connection.
`PASSWORD`	Credentials to set up the MQTT connection.
`LED_RED`	GPIO that controls the red led.
`LED_YELLOW`	GPIO in charge of controlling the yellow led.
`LED_GREEN`	GPIO that controls the green led.
`LIGHT_ADC`	GPIO that provides ADC functionality to sample the analog voltage coming from the photoresistor.
`SWITCH_PIN`	GPIO to switch on and off the relay module.
`WAIT_TIME`	Interval of time to listen to subscription messages.

Table 2.6: Define directives specified in the sketch for ESP-32 board.

Global Variable	Description
`client`	This object is instantiated to enable WiFi connectivity.
`mqtt`	This object handles MQTT connection with the broker.
`light`	Object of type Adafruit_MQTT_Publish in order to send light intensity data to the broker.
`toggle_switch`	Object of type Adafruit_MQTT_Subscribe to listen to MQTT messages to enable and disable the relay module.
`sw_status`	Keeps the status of the relay module.

Table 2.7: Global variables declared in the sketch for ESP-32 board.

Function	Description
void `setup()`	Calls functions to establish WiFi connection and set up MQTT.
void `loop()`	Publishes data about light intensity and listens to messages related to the relay status for WAIT_TIME milliseconds.
void `MQTT_connect()`	Connects to MQTT service provided by the MQTT broker.
void `toggleSwitch()`	Flips the status (active/inactive) of the relay module.
void `ledOff()`	Turns off the status LEDs.
void `ledRed()`	Turns on the red LED.
void `ledYellow()`	Turns on the yellow LED.
void `ledGreen()`	Turns on the green LED.

Table 2.8: Functions used in the sketch for ESP-32 board.

Section 3

System Deployment

3.1 Work Environment Example

The following is a typical use case for the system whose implementation has been described in Section 2.

To begin with, a regular house configuration is taken into account, as shown in Figure 3.1. The ESP nodes are placed in the living room, in the bedroom and in the backyard, by way of example. The Raspberry Pi is located near the wireless router: this opens the possibility of using a wired connection to interface to the router itself.

All the boards must be able to connect to the AP and must be in its coverage range to work properly. Their electrical configuration and purpose is the following:

- **Raspberry Pi:** connected to the AC mains and placed near the wireless router. It must be always up and running to provide the user interface and the MQTT broker functionality;
- **Backyard ESP-12E:** hooked up to a battery and uses deep sleep to reduce the power consumption;
- **Living Room ESP-32:** wired to the AC mains and always active because it is in charge of controlling the living room lamp near the couch;
- **Bedroom ESP-12E:** its power source can be either a battery or the AC mains depending on the availability of power outlets nearby. It uses deep sleep to save energy.

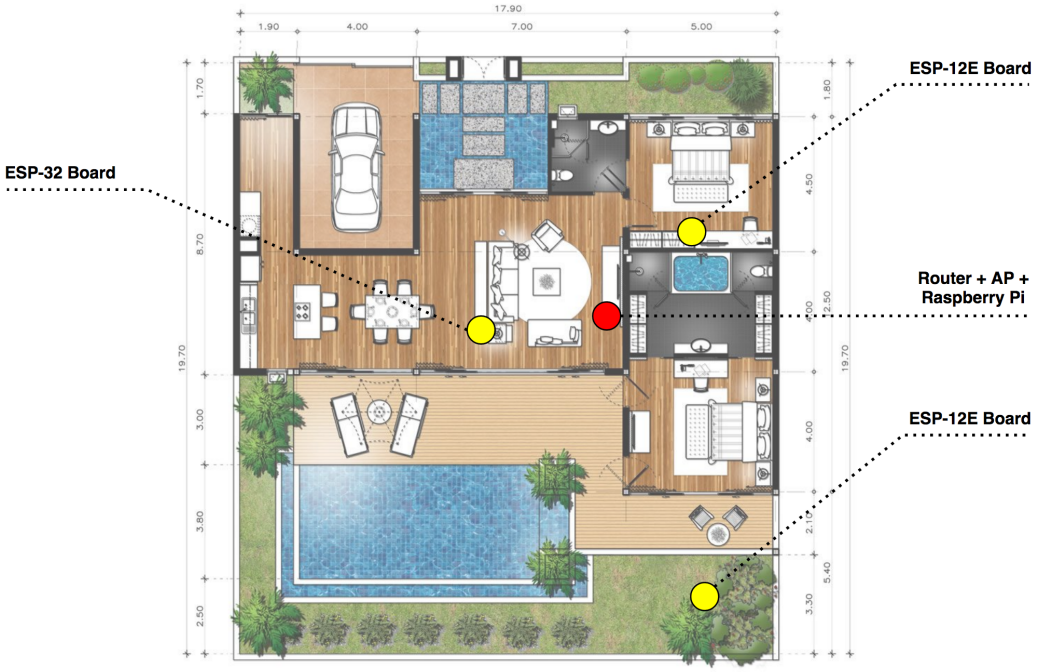

Figure 3.1: Blueprint of a typical house showing the position of all the system nodes.

3.2 Future Extensions

The system implementation opens the possibility of including new features. For example:

- The possibility of adding more boards to cover larger areas and collect more information about the surrounding environment;
- The possibility of using different sensors in addition to the simple temperature, humidity and light ones of the current implementation.
- The possibility of using *energy harvesting* to derive energy from external sources, such as solar power and wind energy.