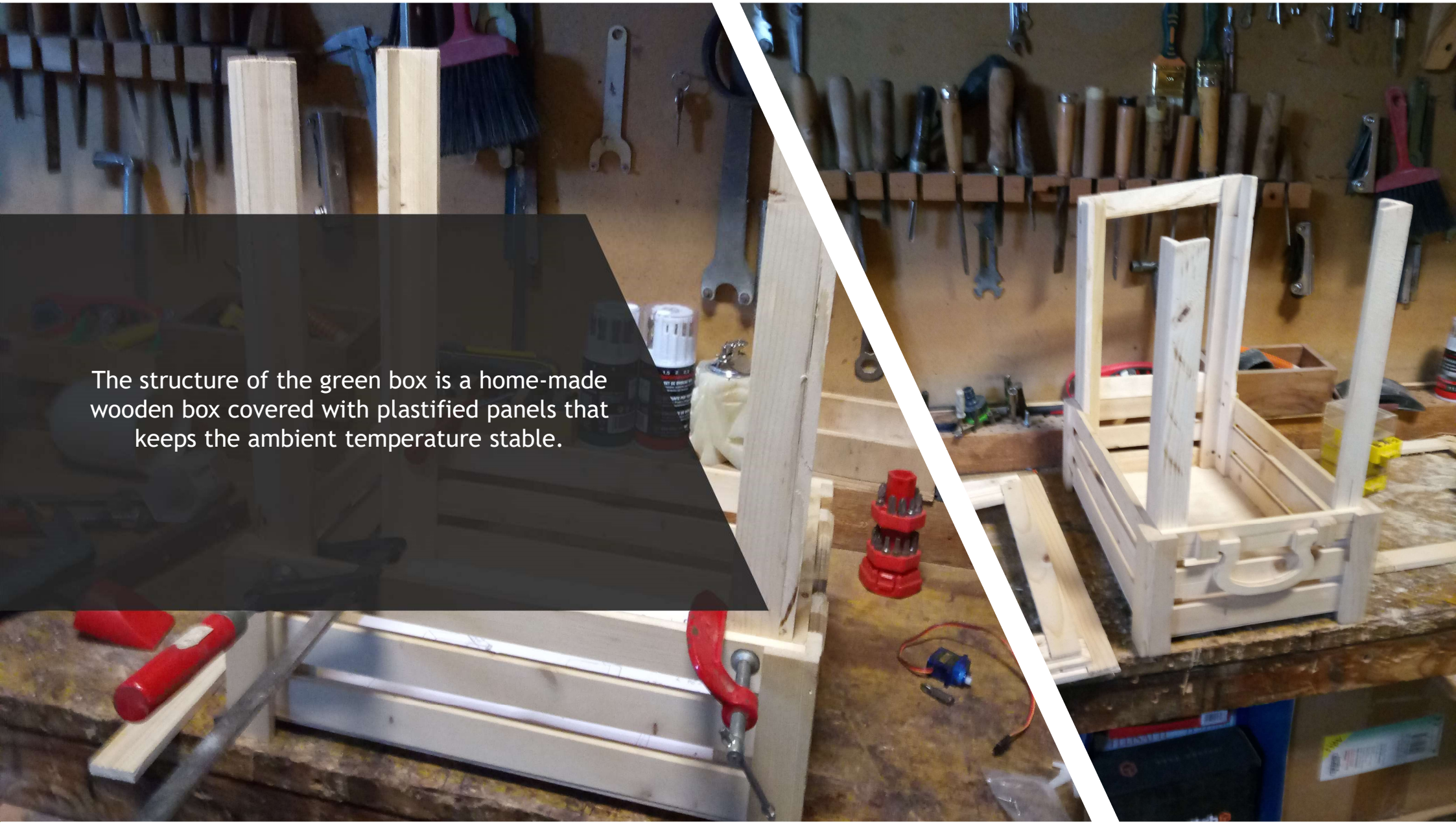
Green Box

By Francesco Spera and Simone Lungarella

Green box is a domestic green house that allows us to menage our plants when we are in home and when we are not, thanks to the IoT concept. Our idea borns to make easier to keep our plants alive without wasting time, so we constructed it with some sensor and a valve menaged with Zerynth's application. The app helps us to check vital parameters of the plant, like temperature, moisture and brightness, and to allow the irrigation and air cycling.

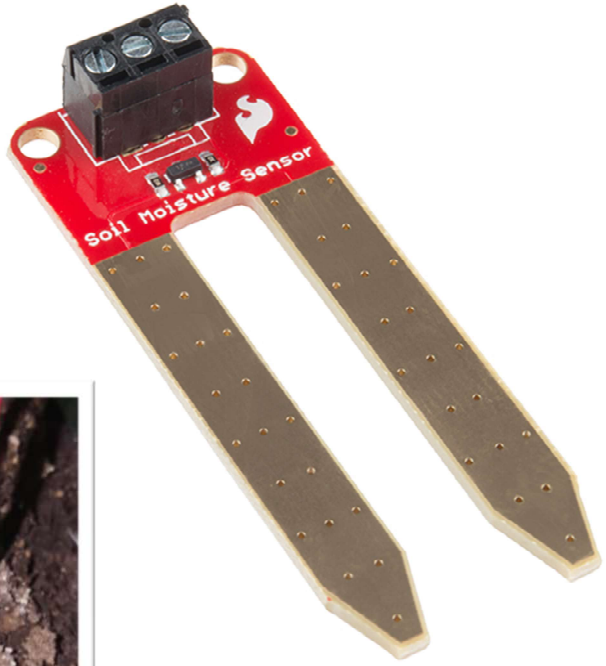


The structure of the green box is a home-made wooden box covered with plastified panels that keeps the ambient temperature stable.

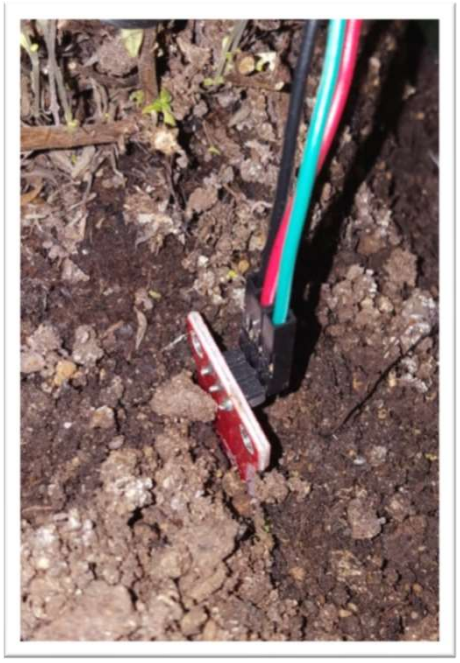
Sensor and actuators used

```
def temperature(*args):
    row_temp = analogRead(pin_temp)
    temp = row_temp*25/750
    temp = "%.1f" % temp
    print (row_temp)
    return temp
```

The temperature sensor is used to get the temperature of the ambient. The value obtained from the sensor represent the voltage that passes through the resistor so the function used transform this value in centigrade.



The moisture sensor measure the humidity level of the terrain. It sends a value between 0 to 4096 so we used a function to transform this value in a percentage value.

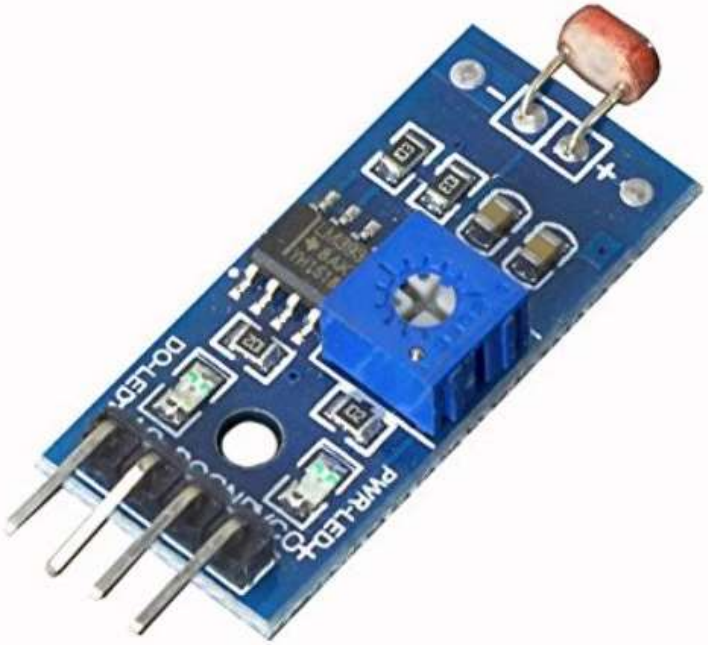


```
max_humidity_value = 4096
def humidity(*args):
    row_hum = analogRead(pin_hum)
    hum = row_hum *100 / max_humidity_value
    hum = "%.1f" % hum
    print (hum)
    return hum
```

```
def brightness(*args):
    row_bright = analogRead(pin_bright)
    if row_bright <= 2500:
        bright = 2 #means very bright
    if row_bright > 2500 and row_bright <= 3100:
        bright = 1 #means normal brightness
    if row_bright > 3100:
        bright = 0 #means very dark
    print (row_bright)
    return bright
```

The brightness sensor use a photoresistor to measure the ambient light. We use a function to transform the value obtained in three parameters that distinguish three different levels of light:

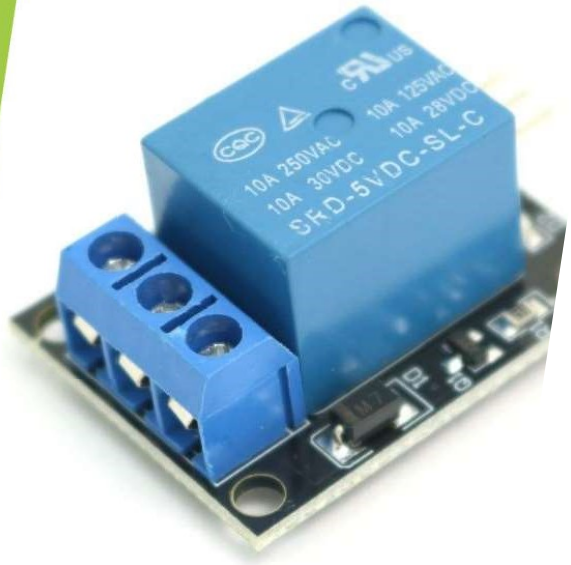
- 0 – very dark
- 1 – normal light
- 2 – very bright



The irrigation is menaged with a solenoid valve that allows the passthrough of water.

The valve wants 9V (generated by a battery) and a resistor of 70 ohm to work properly.

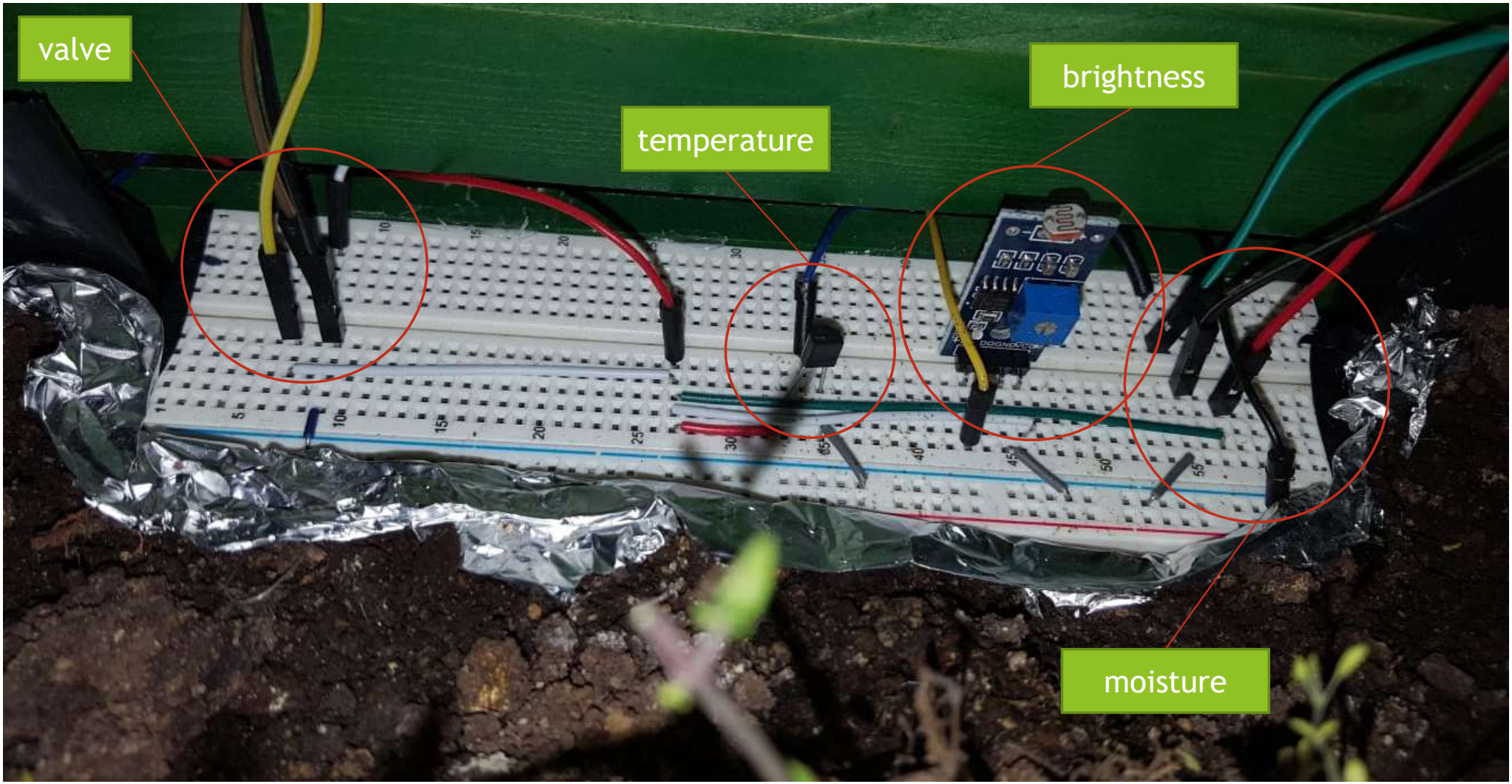
The valve has only 2 wires (positive and negative); When the current flows the flux of the water is interrupted and we need to stop the current to let the irrigation start.



The relay allow us to interrupt the electrical current flow between valve and battery. The relay has 3 pins (+, -, signal) to be menaged by the MCU and other 3 pins to handle the valve state.

```
#irrigation parameters
valve_status = 0    #means closed valve
irrigation_start = 'IRRIGATE'
irrigation_stop = 'STOP'
```

```
def irrigation(*args):
    if valve_status:
        valve_status = 0
        digitalWrite(enable_irrigation,LOW)
        return irrigation_start
    else:
        valve_status = 1
        digitalWrite(enable_irrigation,HIGH)
        return irrigation_stop
```


valve
temperature
brightness
moisture

We use the servo motor to open the door to cycle the air of the ambient.

The rotation of the servo motor is established with functions implemented in the library Servo.

```
from servo import servo
```

We instantiate the class Servo with the following parameters:
min_width: min pulse width (micros).
max_width: max pulse width.
default_width: position where the servo motor is moved when attached.
period: the period of the PWM used for controlling the servo expressed in microseconds.

```
servo_motor = servo.Servo(D2.PWM, min_width=600, max_width=2500, default_width=600, period=100000)
door_status = 0    #means closed door
door_open = 'OPEN DOOR'
door_close = 'CLOSE DOOR'
max_degree = 40
min_degree = 3
```

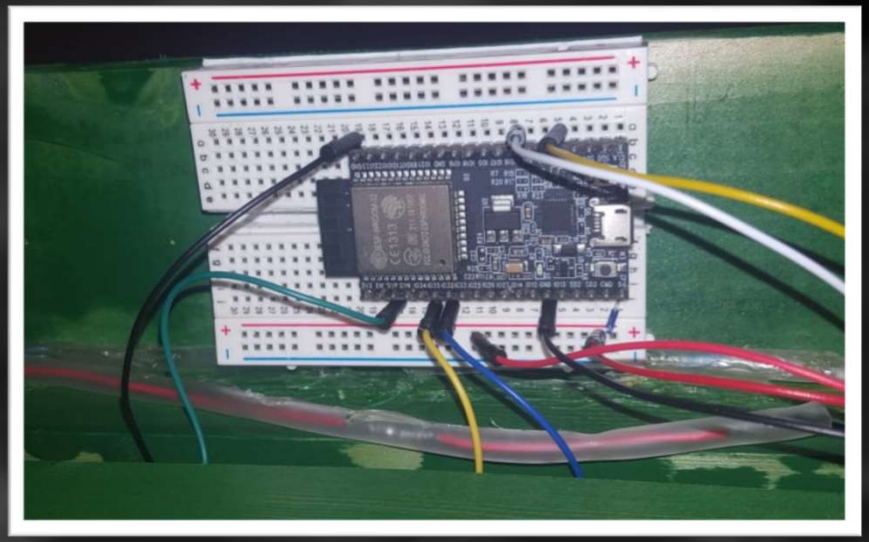
methods:
.attach(): set the servo to the default_width
.moveToDegree(): move the servo between min and max_degree

```
servo_motor.attach()
def opening(*args):
    if door_status:
        door_status = 0
        servo_motor.moveToDegree(min_degree)
        return door_open
    else:
        door_status = 1
        servo_motor.moveToDegree(max_degree)
        return door_close
```

ESP32

The “brain” of GreenBox is the esp32 DevKitC, a low-power, low-cost microcontroller able to handle every sensor and actuators in our project and to menage the communication with the app.

The MCU is positioned outside of the greenhouse, under a large wooden roof, to protect it from water and let us to menage the wires very easly.



The application

The application allow the user to interact with the green house, it shows three vital parameters and two buttons to menage the irrigation and the opening of the door. There is a third button that let the user refresh the values.

The application is divided in two main parts:

- the Python code, written on Zerynth, to handle the ESP32 and its connection with devices

We use the libraries Zerynth wireless and espressif.esp32net to connect ESP32 with the Internet,

```
wifi_driver.auto_init()
sleep(3000)

#connecting to the network, this depends on network_
print('Connecting...')
wifi.link(network_name, wifi.WIFI_WPA2, password)
print('Connected to wifi')
sleep(1000)
```

and the library Zerynth ADM to link the MCU to the template with the function .on() defined in this library.

```
zapp = zerynthapp.ZerynthApp(uid, token)
```

- the template, written in HTML, Javascript and CSS, to menage the user interaction

We used HTML and CSS to make the interface of the application.

Javascript allowed us to create interactive buttons, to make the interface very user-friendly, and help the user with extra information, for example the last time that the values have been updated:

```
function getDate() {
        var d = new Date();
        var months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"];
        document.getElementById("date").innerHTML ="Last update: " + months[d.getMonth()] + "  "
        + d.getDate() + "  " + d.getFullYear() + " " + d.getHours() + ":" + Digits2(d.getMinutes());

}
```

Another import function defined with Javascript is .init() that calls the main functions at the start of the applcation. These functions link the Zerynth application to Javascript functions with the method .call() defined in the library:

```
<script src="https://api.zerynth.com/zadm/latest/z.js"></script>
```

Javascript code

```
$(document).ready(function() {
    getDate();
// initialize the Z object
    Z.init({
        on_connected:  function(){
            Z.call('temperature',[1],temperature_callback),
            Z.call('humidity',[1],humidity_callback),
            Z.call('brightness',[1],brightness_callback)
        }
    })
```

Python code

```
zapp.on('temperature',temperature )
zapp.on('humidity',humidity)
zapp.on('brightness',brightness)
zapp.on('irrigation',irrigation)
zapp.on('opening',opening)
```

HTML code

```
 onclick = "Z.call('irrigation',[1],irrigation_callback)"><p id="I" style="font-weight: bold; font-size:18;"> IRRIGATE </p></button>
onclick = "Z.call('opening',[1],opening_callback)"><p id = "O" style="font-weight: bold; font-size:18;"> OPEN DOOR </p></button>
```

Summing up GreenBox is a very efficient and smart way to check and monitorate a plant to let it grow without any kind of problem inside or outside of a structure, this project is made real by Francesco Spera and Simone Lungarella with the very helpful support of the Zerynth support team and Michela D'Ambrosio.