Lab 6
Play with the Ultrasonic sensor and LED using the web server [Raspberry Pi]

Now that we know how to install and configure our web server, we'll use this lab to learn how to create php pages and receive sensor values and save them in a database. Here you'll discover another way of managing sensors and actuators using web pages.

The components

- Raspberry Pi
- Micro SD card with Raspberry PI OS
- Web server installed and configured (Apache/MySQL/PHP/phpMyAdmin)
- USB Type-C cable
- Ultrasonic sensor
- LED
- Resistor
- Jumper wire
- Breadboard

Step 1 : Creating the python files

Before creating our web page, we're going to build our various circuits and the php files that allow them to be manipulated.

Managing the LED

First, we'll create our circuit and install a LED on the GPIO.BCM 18. After that, we'll create a LED.py file that will be placed in the `/var/www/html/ folder`.

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

LED = 18

GPIO.setup(LED, GPIO.OUT)

state = GPIO.input(LED)

if state :
   GPIO.output(LED, GPIO.LOW)
   print("LED is off")
else :
   GPIO.output(LED, GPIO.HIGH)
   print("LED is on")
```

Using a single file, you can switch the LED on or off and check its status. You can run it to check its operation.

```
pi240@raspberrypi240:/var/www/html $ python LED.py
```

Display Raspberry Pi temperature

The following python script is used to obtain and display the temperature of the Raspberry Pi.

```
import os

def getTemperature():
   temp = os.popen("vcgencmd measure_temp").read()
   return temp

if __name__ == '__main__':
   try:
       print(getTemperature().split("=")[1])
   except:
       print("Error getting the temperature")
```

To test it, type the command below:

```
pi240@raspberrypi240:/var/www/html $ python PiTemp.py
```

Reading ultrasonic sensor data

Before writing the python file, we'll first connect the ultrasonic sensor to the following GPIOs:

- Trigger port in BPIO.BCM 4
- Echo port in GPIO.BCM 17

Then use the following code:

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
import MySQLdb as mdb
from datetime import datetime

GPIO_TRIGGER = 4
GPIO_ECHO = 17

def connect_DB():
   db_sh = mdb.connect (host="localhost", user="root", passwd="root",
db="sensorOut")
   return db_sh

def setup_GPIO():
   GPIO.setmode(GPIO.BCM)
   GPIO.setwarnings(False)

   GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
   GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
   GPIO.output(GPIO_TRIGGER, True)
   time.sleep(0.00001)
   GPIO.output(GPIO_TRIGGER, False)
   StartTime = time.time()
   StopTime = time.time()
```

```
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()
    TimeElapsed = StopTime - StartTime
    distance = (TimeElapsed * 34300) / 2

    return distance

def insert_value (val):
    db_sh = connect_DB()
    cursor_sh = db_sh.cursor()
    today = datetime.now()
    dt = today.strftime('%Y-%m-%d')
    tm = today.strftime('%H:%M:%S')
    sql_sh = "INSERT INTO ultrasonic(value,date,time) "
    sql_sh += "VALUES ('"
    sql_sh += str(val)
    sql_sh += "','"
    sql_sh += dt
    sql_sh += "','"
    sql_sh += tm
    sql_sh += "') "

    try:
        cursor_sh.execute(sql_sh)
    except:
        db_sh.rollback()
    db_sh.commit()
    db_sh.close()


if __name__ == '__main__':
    try:
        setup_GPIO()
        dist = distance()
        #insert_value(dist)
```

```
        print(dist)
    except KeyboardInterrupt:
        GPIO.cleanup()
```

This file contains the following functions:

- `connect_DB()`: This function connects to the database. To do this, we'll need to create a `sensorOut` database and an `ultraSonic` table with the following columns: `(ID, value, date, time)`
- `setup_GPIO()`: This function is used to configure ports as input and output.
- `distance()`: This function calculates and returns the distance.
- `insert_value()`: This function inserts the data `(value, date and time)` into the `ultrasonic` table.

To test the script, use the following command:

```
pi240@raspberrypi240:/var/www/html $ python distance_once.py
```

Step 1: Setting up the web files

Now that we've created the python files for managing the LED, displaying the RPi temperature and measuring the distance with the ultrasonic sensor, we'll create the various web pages to display all these values and actions.

Calling a python using a PHP file

In order to make a web page call to the python script, we'll write the following script:

```
<?php
        $command = escapeshellcmd("sudo python <your python file>");
        $output = shell_exec($command);
        echo $output;
?>
```

The following script allows you to execute a python script from a PHP file. We'll use the native php `shell_exec` method, which executes a system command and returns the result to the `$output` variable. To test them, open the PHP file from your laptop using your favorite browser.
In the following, we'll call this PHP file with a jQuery ajax function. To do this, we'll create a `script.js` file and save it in the `/var/ folder`.

```
$(document).ready(function(){
   function executePythonScript() {
       $.ajax({
           url: 'distance_once.php',
           type: 'GET',
           success: function(response) {
               $('#resultDist').html(response);
           },
           error: function(xhr, status, error) {
               console.error('Error on executing the AJAX request : ' + status + ' '
+ error);
           }
       });
       $.ajax({
           url: 'PiTemp.php',
           type: 'GET',
           success: function(response) {
               $('#resultTemp').html(response);
           },
           error: function(xhr, status, error) {
               console.error('Error on executing the AJAX request : ' + status + ' '
+ error);
           }
       });
   }
   setInterval(executePythonScript, 1000);
   var myButtonOn = $("#ledOn");

   myButtonOn.on("click", function(e) {
       e.preventDefault();
       manage_LED();
   });



   function manage_LED() {
       $.ajax({
```

```
            url: 'scriptLED.php',
            type: 'POST',
            success: function(response) {
                $('#ledOn').val(response);
            },
            error: function(xhr, status, error) {
                console.error('Error on executing the AJAX request : ' + status + ' '
+ error);
            }
        });
    }
});
```

The preceding code executes three ajax scripts. The first two run every second and export the results in two `spans`, `resultDist` and `resultTemp` respectively.
The third script runs each time the `ledOut` button is clicked, and displays the LED status (returned by the python script) directly on the button content.
And finally, we'll consolidate everything in the following index.php file, not forgetting the style.css file (created by your comrade ***Mohamed Amin Sekkat***).

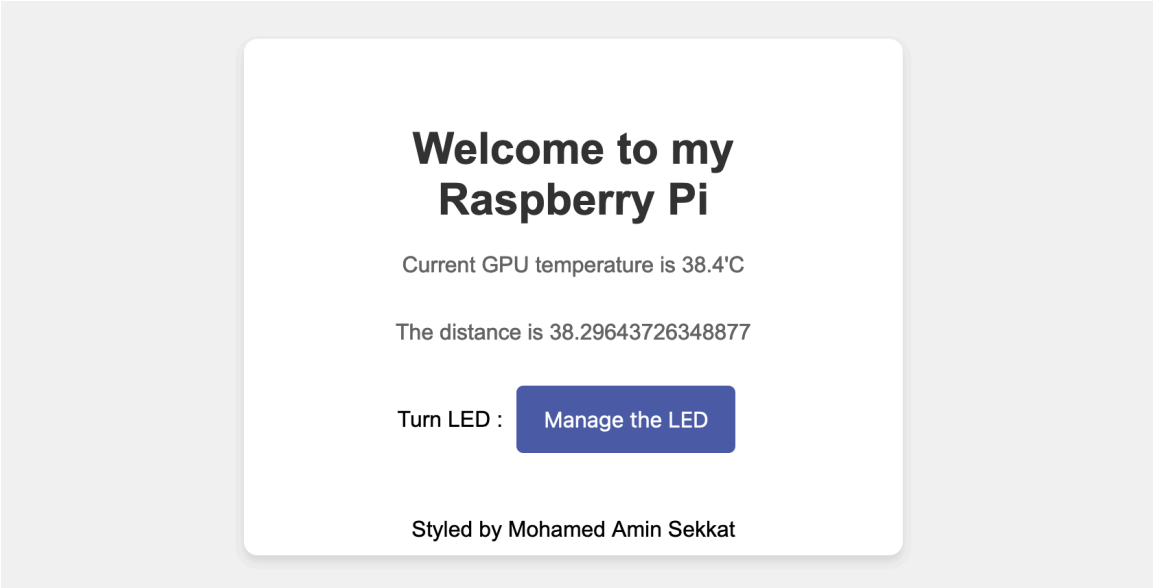
```
<html>
    <head>
      <title>Raspberry Pi</title>
      <link rel="stylesheet" type="text/css"
href="http://raspberrypi240.local/style.css">
      <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   </head>
   <body>
       <div class="container">
       <h1>Welcome to my Raspberry Pi</h1>
       <p>Current GPU temperature is <span id="resultTemp"></span></p>
       <p>The distance is <span id="resultDist"></span></p>
       <form action="/" method="POST">
           Turn LED :
           <input type="submit" name="submit" id="ledOn" value="Manage the LED">
       </form>
       <span class="copyright">Styled by <i>Mohamed Amin Sekkat</i>
```

```
</span>
      </div>
      <script src="script.js"></script>
      </body>
      </html>
```

```
body {
   width: 100%;
   height: 100vh;
   margin: 0;
   padding: 0;
   display: flex;
   justify-content: center;
   align-items: center;
   background-color: #f0f0f0;
   font-family: Arial, sans-serif;
}
.container {
   width: 400px;
   background-color: #fff;
   border-radius: 10px;
   box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
   padding: 40px;
   text-align: center;
   padding-bottom: 10px;
}
h1 {
   color: #333;
   margin-bottom: 20px;
}
p {
   color: #666;
   margin-bottom: 30px;
}
form {
   display: flex;
```

```
    justify-content: center;
    align-items: center;
    padding-bottom: 30px;
}
input[type="submit"] {
    padding: 15px 20px;
    font-size: 16px;
    cursor: pointer;
    border: none;
    border-radius: 5px;
    margin: 0 10px;
    transition: background-color 0.3s ease;
}
input[type="submit"] {
    background-color: #4c5caf;
    color: white;
}
input[type="submit"]:hover {
    background-color: #333;
}
span {
    margin-top: 20px !important;
}
```

To run all these files, you'll need to open your favorite browser and use the hostname in the url:



Challenges

Challenge #1

Create a web page to control 3 LEDs.

Challenge #2

Use the database to save the distance threshold, after reaching this threshold, power on the LED, otherwise power it off. This distance threshold must be updated.

Challenge #3

Ask your instructor to give you any sensor and actuator. Create a web page to display the sensor result and activate the actuator by a web page action and by a threshold to be defined using the database.