ASSIGNMENT NO:

AIM: Design a web interface to control connected LEDs remotely using Raspberry-Pi/Beagle board/Arduino.

Theory:

One of the function of IOT is remote control of devices, being able to trigger action from aremote location. This is known as remote configuration, Whether it is a home, office or an industrial site, connected devices rely on some form of automation through sensors or machines and need a mechanism to control their operation remotely. Here we will build a small IOT simulation using Rpi and showcase the most commeon operation switching on and off a remote device.

Hardware Requirements:

- RPi model with Raspbian OS
- 1 LED AND 1 490 OHM RESISTOR

Software Requirements:

- RPi GPIO, GPIO Python library for Raspbian OS.
- Apache2

We will create 2 subsystems. one will be controller, a basic web application in the form of a webpage which can display the current states of device and send control messages to it and second one is the actual device simulated as on LED and controlled via Raspberry Pi. (https://www.youtube.com/watch?v=cp8Ni_0xXCw)

Here we will learn how to create a fully controlled IoT Raspberry Pi LED. The idea is to control the LED, using low level commands written with shell scripts commanded straight from a PHP page. Using higher level languages as Python for example, makes the LED very responsive and quick to act.

In this part of the experiment we will learn:

- Install and use the library WiringPi to control the RPi GPIOs
- Make the RPi a WebServer using a PHP to control the LED over internet

The block diagram shows the basic idea of the project. The RPi will be set as a WebServer and will receive commands from an PHP page. Those commands will change the GPIOs status, making the RPi control LED. As you can realize, the LED is in fact a particular case of an IoT project. You can control what you want.

Controlling LED with Raspberry Pi:

The next step is to create a logic to control the LED.

Let's assigned the GPIOs for LED inputs:

- LED: GPIO.4

The next step is to create shell scripts to run the LEDs. Every script file is essentially plain text. When a text file is attempted to be executed, shells will parse through them for clues as to whether they're scripts or not, and how to handle everything properly.

So, let's save the scripts in directory html under www under var, where our webpage with extension PHP will be located under it:

```
sudo mkdir /var/www/html
```

Now, let's move all files to this new directory:

```
sudo mv /*.py  /var/www/html
```

```
cd /var/www/html
```

Using the line command ls, you can see the files created.

```
ls
```

Now, anytime that the RPI starts, it will be ready to control the designed outputs.

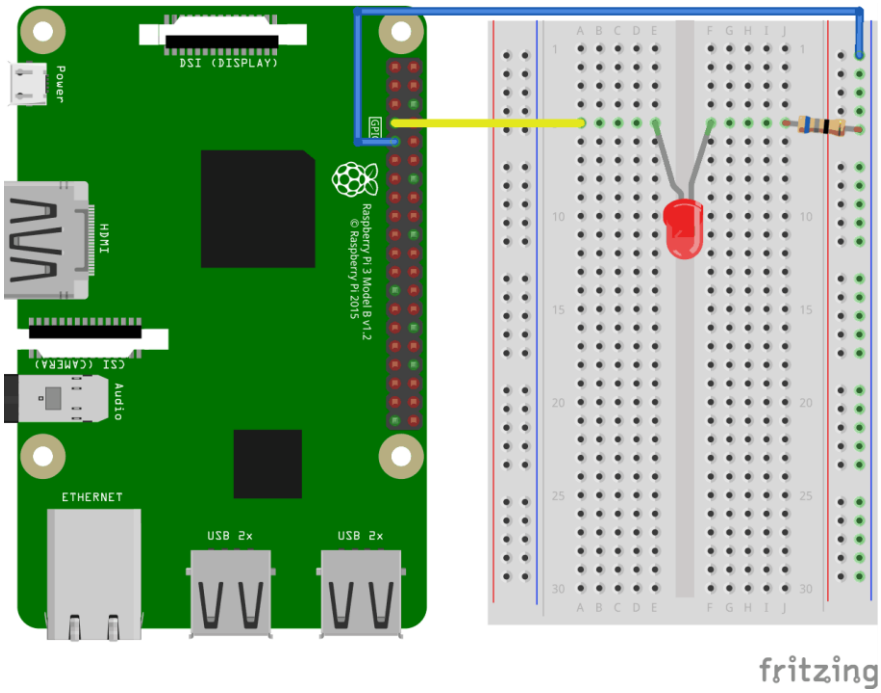


Fig: Interfacing of LED with Raspberry Pi

Installing the PHP WebServer:

We will install PHP that is a very “light” and fast WebServer using of Apache for example.

Controlling the LED remotely with SSH is pretty cool but the interface (console) isn't very user friendly and typing the commands every time is long and annoying. That's why we need a graphical interface for our project.

Programming an app for each OS (IOS, Android, Windows phone, Mac, Linux, Windows,...) would be too long and would require to know a lot of different languages for nearly nothing. It would also require to do an application running on the Raspberry Pi. Making it this way would be overkill and time wasting.

That's why a website is the best solution, it's compatible with all devices and you "only" need to know four languages: HTML (for the page's skeleton), CSS (page's style), PHP (interactions with the server) and JavaScript (interactions with the user).

We indeed need to install a web server on the Raspberry Pi. In our case, we don't need a MySQL database, only a HTTP server and its PHP extension.

After updating your Raspberry Pi with the

sudo apt-get update

Type:

sudo apt-get install apache2 php5 libapache2-mod-php5

to install Apache HTTP server and PHP5 extension. You can now test if your server is working by typing the IP of your Raspberry Pi in your browser. You should now see a "**It works!**" page with two other lines. If you don't, then check your board's IP, try re-installing Apache or rebooting your Raspberry Pi. This page is showing that your

Apache server is working properly but not its PHP extension. To check it, navigate to your "*/var/www/html*" directory by using:

cd /var/www/html

If you use the "***ls***" command, you should have only one file named "*index.html*". This file corresponds to the "**It works!**" page. You can now delete it ("***sudo rm index.html***") and create another one called "*index.php*"

sudo nano index.php

Then type the following text:

```
<?php
  phpinfo();
?>
```

After saving it using ^o (Ctrl + o), exit nano editor with ^x (Ctrl + x). Now if you refresh your browser, you should see a long page with lots of information about your server and PHP. If you don't, check the *index.php* file, try re-installing PHP or try to understand the error displayed instead of the page (Google it if necessary).

If both pages were correctly displayed, then you now have a fully functional Apache/PHP server but using nano every time is annoying and not very comfortable. We indeed need to transfer files from your computer to your Raspberry Pi. You may want to install a FTP server but it isn't necessary, you can already transfer files using the SFTP protocol. All you need is an SFTP client on your computer. If you try transferring files before reading what's next, you'll probably have issues such as "access refused" or "cannot write here". It's due to the fact that the user pi isn't owning the www directory. Indeed, if you try the "**ls -l /var/www**" command, you'll see that only root (the super user) is owning the www directory. You can use the

sudo chown -R pi /var/www

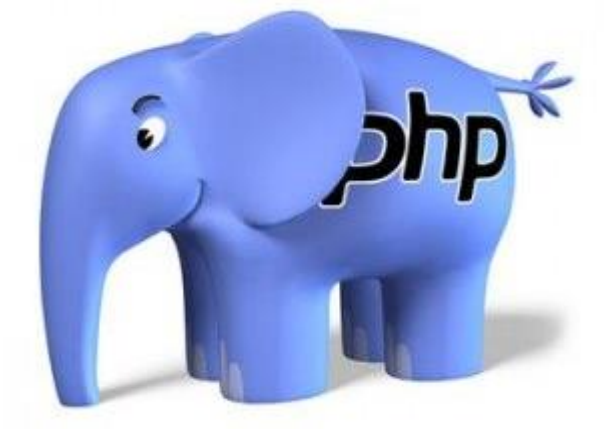
command to change it or create a group named www-data in which you place the pi user then use the

sudo chown -R www-data /var/www

The **-R** flag is standing for recursive, it means that the user/group isn't owning only the directory itself but also everything inside (index.php as example).

You now have your server ready to work and to receive web pages.

At this point the web server is running and if a page index.php is located at /var/www/html, we can access it from any browser, typing the RPi IP address/index.php:



PHP stands for "*PHP: Hypertext Preprocessor*", It's a server side scripting language. It means that the PHP code is executed once (each time the page is requested) by the server and cannot be seen by the client. I used this language because it's the most popular (and that's the only one I know) but you have to know that they are other server side languages like Python, Ruby, Lua, Perl.

Executing applications with a PHP code can be done with two different functions: *exec* (for execute) and *system*. Firstly, the "*system*" function. It takes two parameters: "*system (string $command, int $return_var)*", as you guessed it, the first parameter is the command to execute and the second one is the returned status of the executed command. The second parameter isn't compulsory. You can use this function if you don't expect an answer from the command executed. Thus, you can use it if you need to execute "***gpio mode 0 out***" or "***gpio write 0 1***" commands. Example:

```
<?php
    system ( "gpio mode 0 out" );
    system ( "gpio write 0 1" );
?>
```

Then, the "exec" function. This function is making exactly the same work than "system" but it reads and stores what the command printed. It takes three parameters: "exec (string $command, array $output, int $return_var)", again $command and $return_var are the same parameters and the only difference is the $output array. As it's name says

it will store the command's output in an array. Thus, you can use this function if you need what the command prints like with the "**gpio read 0**" command.

Example:

```
<?php
    exec ( "gpio read 0", $status );
    print_r ( $status );
?>
```

let's create a simple test webpage (Note that for testing, previously mov a png file - LED.png and SIT_logo.png- to html directory):

LED.png

cd /var/www/html

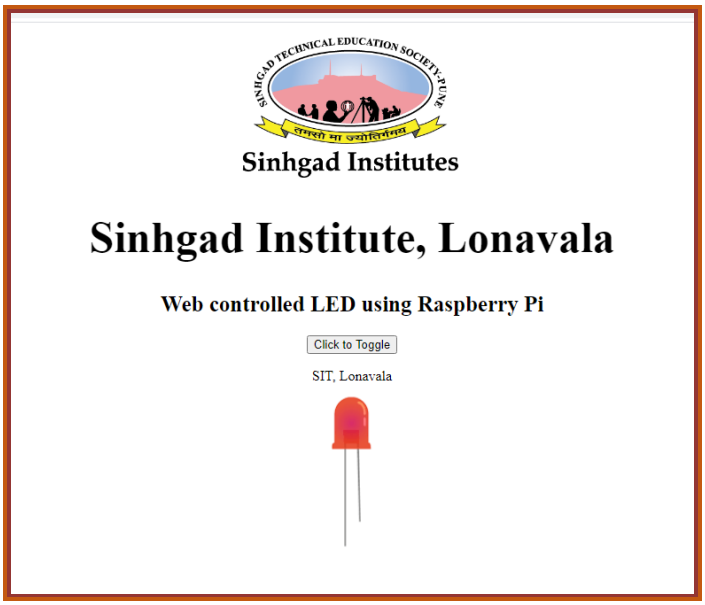
sudo nano index.php

Once you finish the page edition, save it and change the permissions:

sudo chmod 755 index.php

Web application:

Th web interface looks like this:



- This is a very simple webpage with a visual indicator for the device and a button to toggle ON/OFF state of LED.

HTML Script for GUI:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>

<body>
<div align="center">
<img src="SIT_logo.png" alt= "SIT" style="width:20%;">
<h1 style="font-size:50px">Sinhgad Institute, Lonavala</h1>
<h2 style="font-size:25px">Web controlled LED using Raspberry Pi</h2>

<p><button onclick="myFunction()">Click to Toggle</button></p>

<div id="myDIV">SIT, Lonavala</div>

<script>
function myFunction() {
        var x = document.getElementById("myDIV");
        if (x.innerHTML === "LED OFF") {
        x.innerHTML = "LED ON";
        } else {
        x.innerHTML = "LED OFF";
        }
}
</script>
<img src="LED.png" alt= "LED" style="width:20%;">

</body>
</html>
```

- In the background, we have Apache server that helps to performs 2 operations upon receiving certain events.

1) sends a request to toggle the state of device.

2) recives response with current state of the device.

The final result can be seen at screenshot above.

Now, open your browser and type your Raspberry Pi IP Address, for example in my case: 172.17.168.49/index.php

Creating an PHP Page to Control the LED:

Let's think about a simple design for our page. What commands we may have?

1. Two buttons for LED On, LED OFF ==> will work with the scrips: LED On.py, LED OFF.py

Using the index.php that we created in the last step, let's include buttons that will call functions to execute the scripts.

```
<?php
if (isset($_POST['LED ON']))
{
exec("sudo killall python");
exec('sudo python /var/www/html/gpio1.py');
}
if (isset($_POST['LED OFF']))
{
exec("sudo killall python");
exec('sudo python /var/www/html/gpio2.py');
}

?>
```

The above php will control a squared button with a "LED ON", "LED OFF".

When the “LED On” button is pressed, due the command "sudo python /var/www/html/LED/LED On.py" is executed and LED turns “ON”.

When the “LED OFF” button is pressed, due the command "sudo python /var/www/html/LED/LED OFF.py" is executed and LED turns off.

There are some php functions that will organize the look & fill that you can realize looking the full php page. The php source can be seen bellow:

gpio1.py

```
#!/usr/bin/python
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,GPIO.OUT) #GPIO4

try:
        GPIO.output(7,GPIO.HIGH) #GPIO4

except KeyboardInterrupt:
    GPIO.cleanup()
```

gpio2.py

```
#!/usr/bin/python
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,GPIO.OUT) #GPIO4

try:
        GPIO.output(7,GPIO.LOW) #GPIO4

except KeyboardInterrupt:
    GPIO.cleanup()
```

Program:

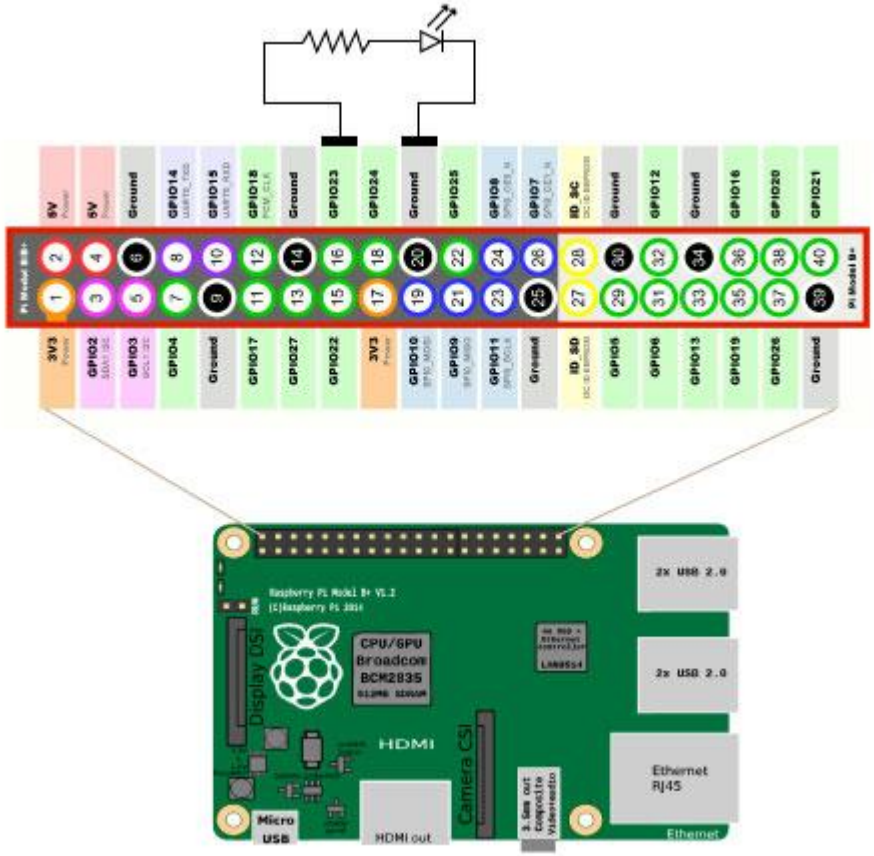
```
<html>
<head>
<div align="center">
<img src="SIT_logo.png" alt= "SIT_logo" style="width:10%;">
<font size="+2"><h1>LED Blinking using RPi </h1></font>
<font size="+2"><h4>Sinhgad Institute of Technology, Lonavala</h4></font>
<br>
</head>
<style>
body {background-color: lightyellow}
h1 {color:blue}
button {
    color: blue;
    background: lightgray;
    border: 5px solid #000;
    border-radius: 8px;
    position: center;
}
</style>
<img src="LED.png" alt= "LED" style="width:20%;">
<body>
<form method="get" action="index.php">
    <input type="submit" style= "font-size: 50 pt" value="OFF" name="OFF" >
    <input type="submit" style= "font-size: 50 pt" value="ON" name="ON" >
</form>
<div aline= >
</div >
<?php
    system("gpio -g mode 4 out");
    if (isset($_GET['OFF']))
        {
            echo "Status: Led is OFF";
            system("gpio -g write 4 0");
        }
    else if (isset($_GET['ON']))
        {
            echo "Status: Led is ON";
            system("gpio -g write 4 1");
        }
?>
</body>
</html>
```

Installing the RPi:

Once all is working, it is time to add the RPI. The complete circuit is shown above fig. Interface the RPi 5V battery between the LEDs at lower level chassis. The RPI is on top. Make all cable connections and turn on the RPi. If all the previous steps were OK, you can control the LED, using the RPi IP address. Open your favorite WebBrowser and safe trip!

Raspberry Pi and LED

Here is the schematic for the raspberry Pi connections to be used in this application.



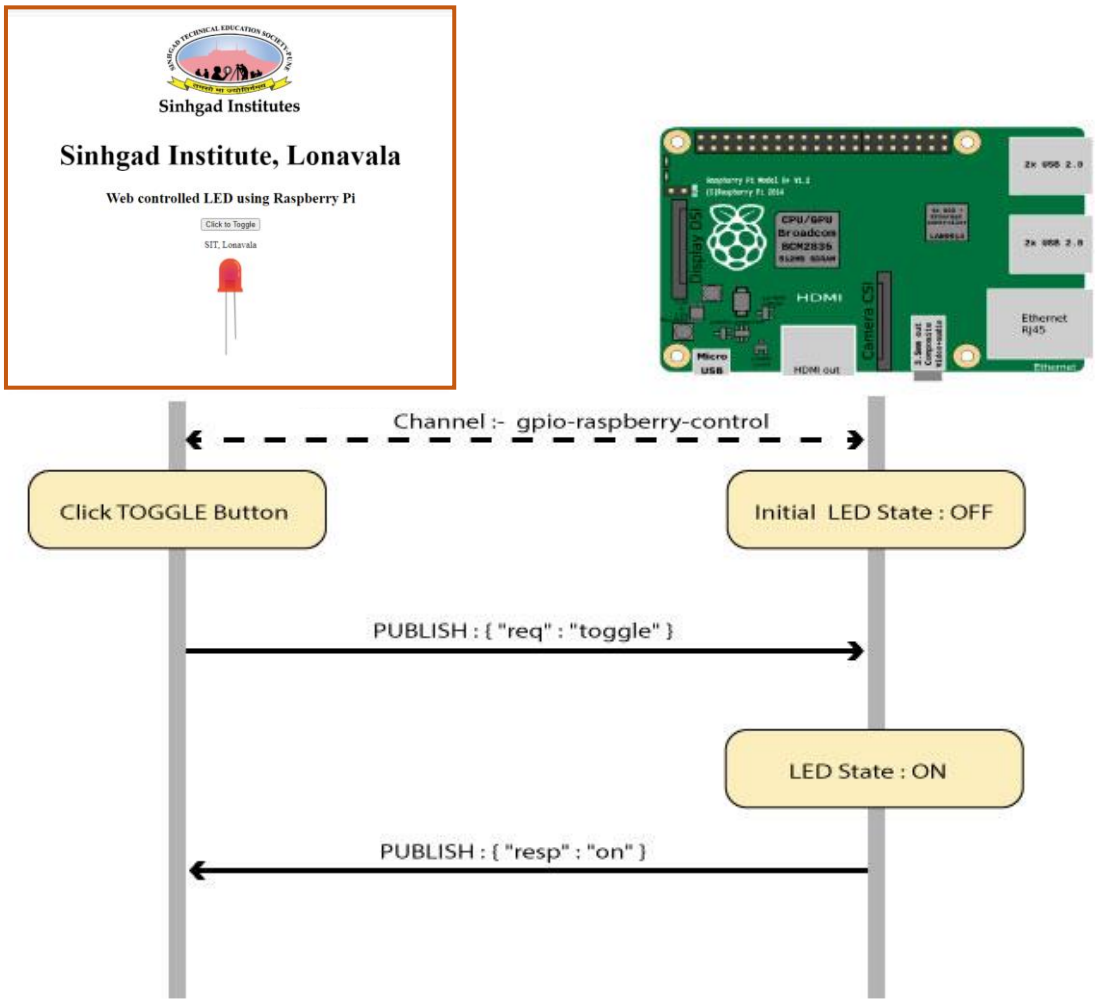
We are going to use the Raspberry Pi GPIO Python library to send the control messages to Raspberry Pi GPIO ports. This library works well with the python environment available by default with Raspbian OS.

The python code for driving the LED is executed as part of the PubNub callback on 'gpio-raspberry-control' channel

When a toggle request is received, the application checks the current state of the GPIO

driving pin of the LED, toggles its state and then sends the new state back to the web application as a response message.

The exchange of messages between the web application and Raspberry Pi can be visualized as follows.



Conclusion:

This demonstration can be generalized as application layer service which sits on the top of an IoT stack. Apache server can provide the middleware to manage the communication primitives of an IoT stack, while various such applications can be deployed on top to monitor & control the devices and visualize and analyze the data generated from them.