Practical No : 7

IoT based Web Controlled Home Automation using Raspberry Pi

Required Components:

For this project, the requirements will fall under two categories, Hardware and Software:

I. Hardware Requirements:

1. Raspberry Pi 3 (Any other Version will be nice)
2. Memory card 8 or 16GB running Raspbian Jessie
3. 5v Relays
4. 2n222 transistors
5. Diodes
6. Jumper Wires
7. Connection Blocks
8. LEDs to test.
9. AC lamp to Test
10. Breadboard and jumper cables
11. 220 or 100 ohms resistor

II. Software Requirements:

Asides the Raspbian Jessie operating system running on the raspberry pi, we will also be using the **WebIOPi** frame work, **notepad++** running on your PC and **filezila** to copy files from the PC to the raspberry pi, especially the web app files.

Also you **dont need to code in Python** for this Home Automation Project, WebIOPi will do all the work.

Preparing the Raspberry Pi:

To **update the raspberry Pi** below commands and then reboot the RPi;

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

With this done, the next thing is for us to **install the webIOPi framework**.

Make sure you are in home directory using;

```
cd ~
```

Use wget to get the file from their sourceforge page;

```
wget http://sourceforge.net/projects/webiopi/files/WebIOPi-0.7.1.tar.gz
```

When download is done, extract the file and go into the directory;

```
tar xvzf WebIOPi-0.7.1.tar.gz
cd WebIOPi-0.7.1/
```

At this point before running the setup, we need to **install a patch as this version of the WebIOPi** does not work with the raspberry pi 3 which I am using and I couldn't find a version of the WebIOPi that works expressly with the Pi 3.

Below commands are used to install patch while still in the WebIOPi directory, run;

```
wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch
patch -p1 -i webiopi-pi2bplus.patch
```

Then we can run the setup installation for the WebIOPi using;

```
sudo ./setup.sh
```

Keep saying yes if asked to install any dependencies during setup installation. When done, reboot your pi;

```
sudo reboot
```

Test WebIOPi Installation:

Before jumping in to schematics and codes, With the Raspberry Pi back on, we will need to test our WebIOPi installation to be sure everything works fine as desired.

Run the command;

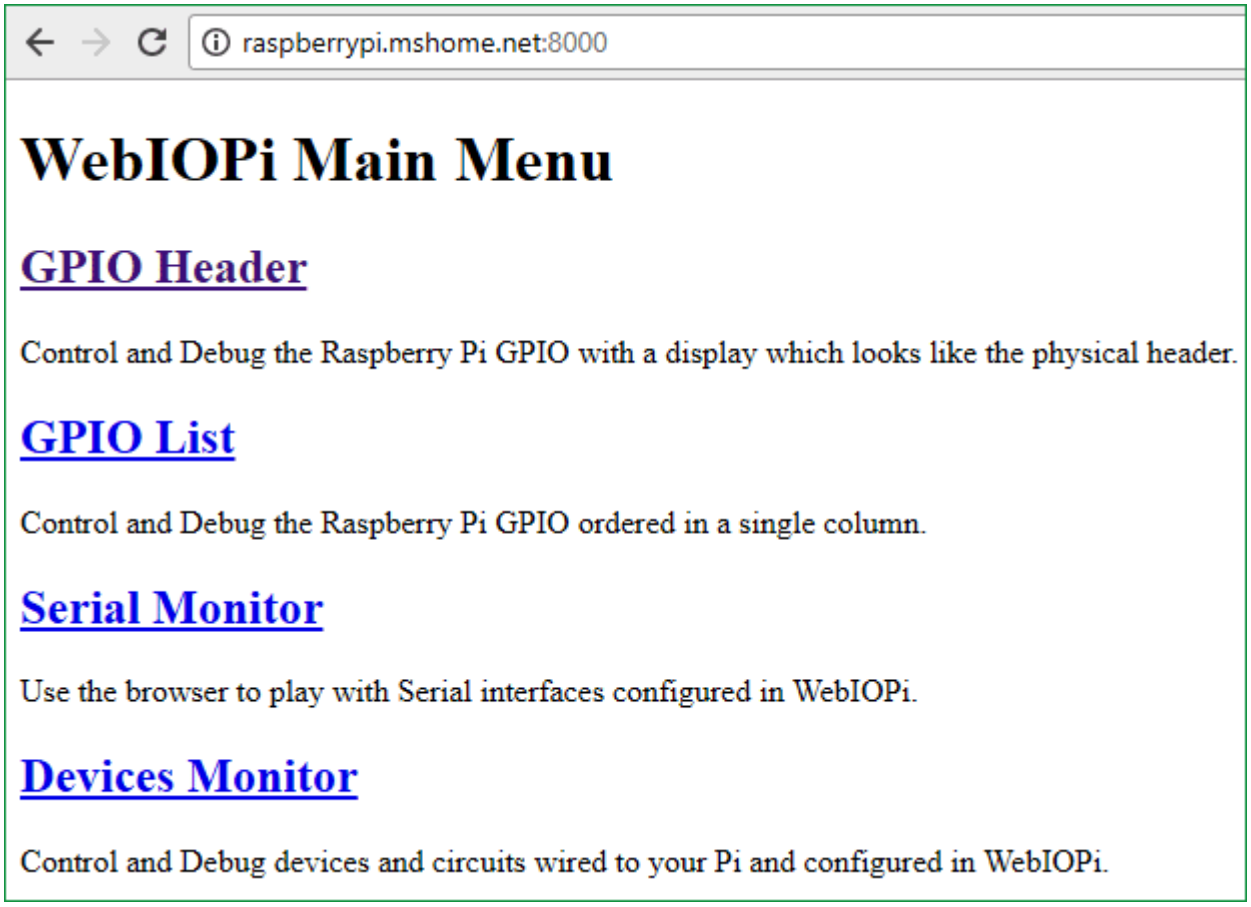
```
sudo webiopi -d -c /etc/webiopi/config
```

After issuing the command above on the pi, point the web browser of your computer connected to the raspberry pi to **http://raspberrypi.mshome.net:8000 or http;//thepi'sIPaddress:8000**. The system will prompt you for username and password.

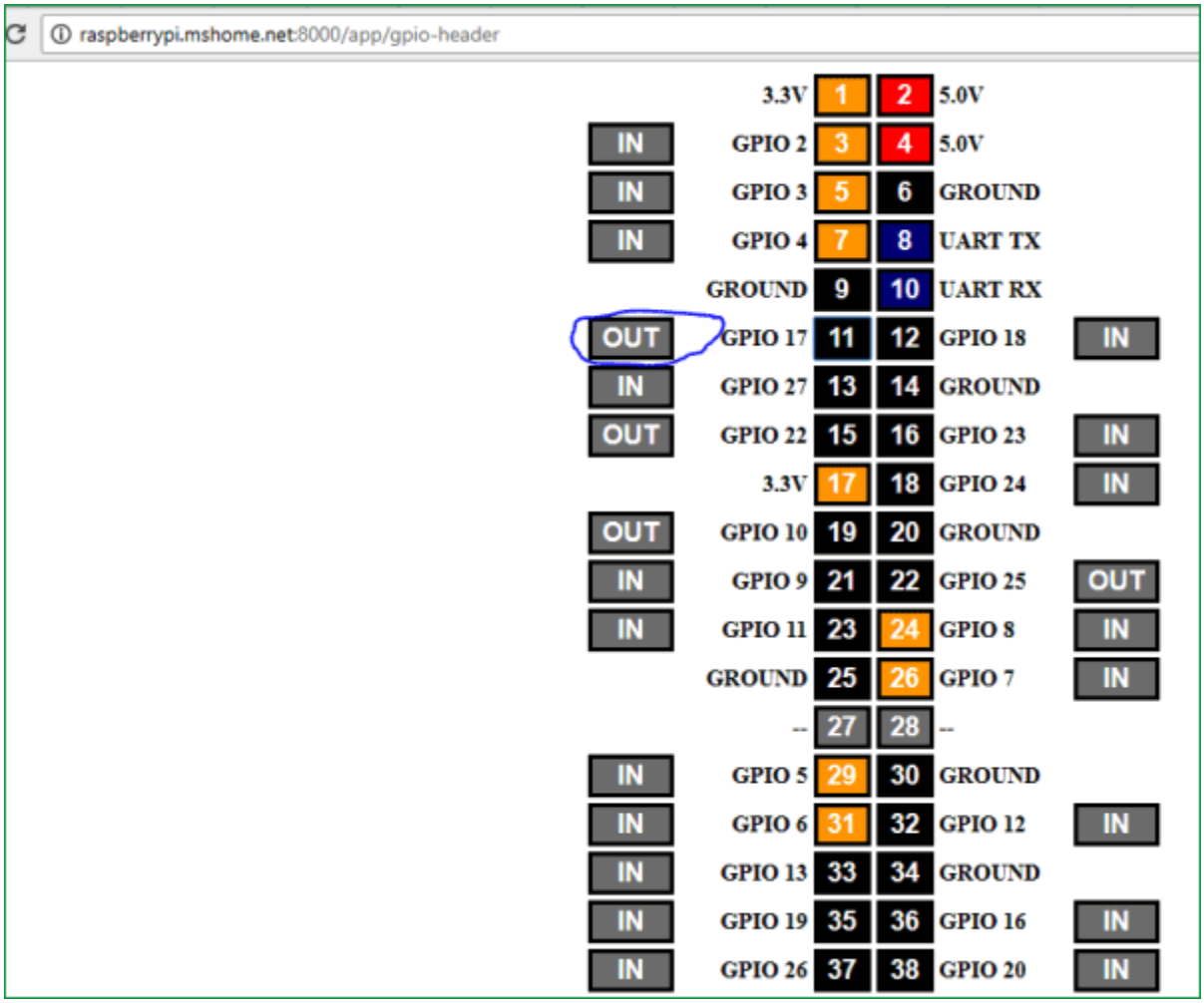
```
Username is webiopi
Password is raspberry
```

This login can be removed later if desired but even your home automation system deserves some extra level of security to prevent just anyone with the IP controlling appliances and IOT devices in your home.

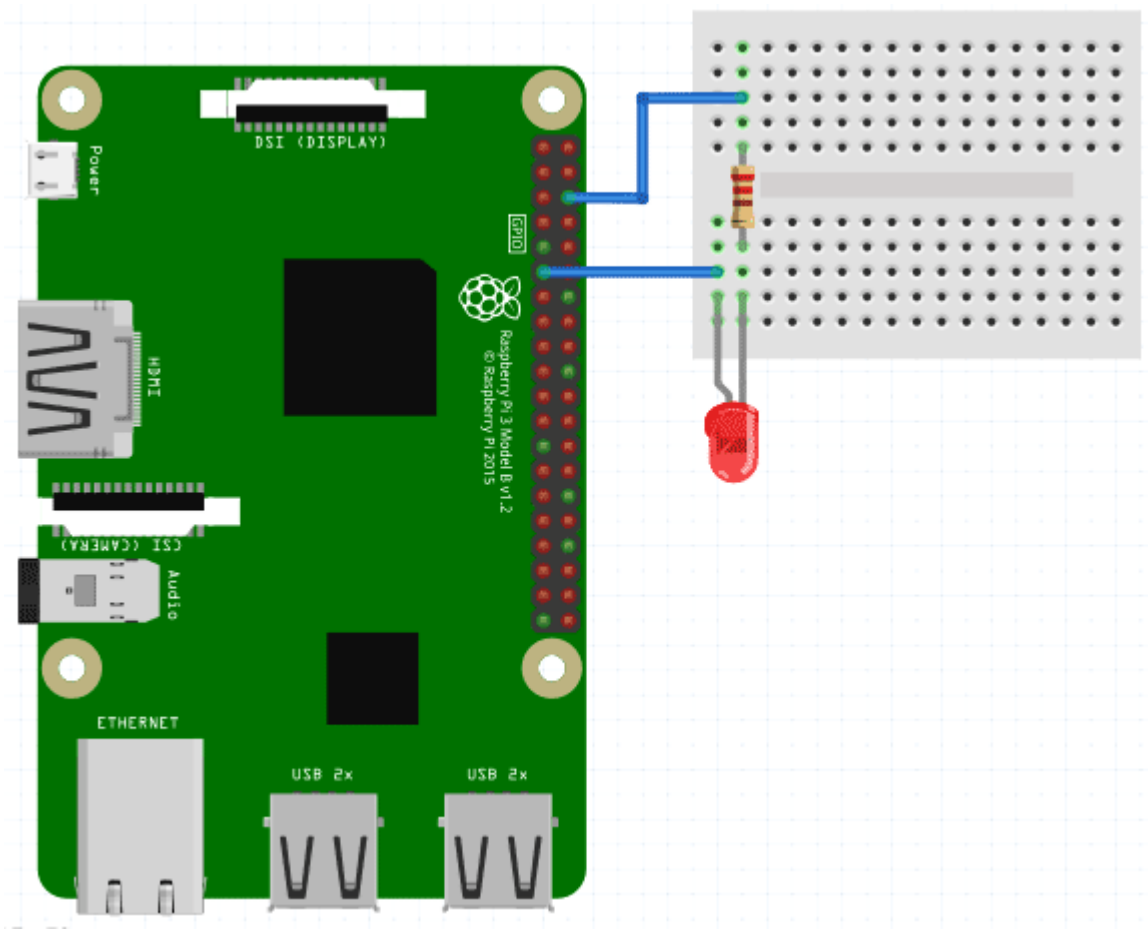
After the login, look around, and then **click on the GPIO header link**.



For this test, we will be connecting an LED to GPIO 17, so go on and set GPIO 17 as an output.



With this done, connect the led to your raspberry pi as shown in the schematics below.



After the connection, go back to the webpage and **click the pin 11 button to turn on or off the LED**. This way we can control the Raspberry Pi GPIO using *WebIOPi.*

After the test, if everything worked as described, then we can go back to the terminal and stop the program with CTRL + C. If you have any issue with this setup, then hit me up via the comment section.

More info on Webiopi can be found on its Wiki page (http://webiopi.trouch.com/INSTALL.html)

With the test complete, we are then set to start the main project.

Building the Web Application for Raspberry Pi Home Automation:

Here we will be editing the default configuration of the WebIOPi service and add our own code to be run when called. The first thing we will do is get ***filezilla*** or anyother FTP/SCP copy software installed on our PC. You will agree with me that coding on the pi via the terminal is quite stressful, so filezilla or any other SCP software will come in handy at this stage. Before we start writing the html, css and java script codes for this **IoT Home automation Web application** and moving them to the Raspberry Pi, lets create the project folder where all our web files will be.

Make sure you are in home directory using, then create the folder, go into the created folder and create html folder in the directory:

```
cd ~
mkdir webapp
cd webapp
mkdir html
```

Create a folder for scripts, CSS and images inside the html folder

```
mkdir html/css
mkdir html/img
mkdir html/scripts
```

```
pi@raspberrypi:~/projects $ cd ~
pi@raspberrypi:~ $ mkdir webapp
pi@raspberrypi:~ $ cd webapp
pi@raspberrypi:~/webapp $ mkdir html
pi@raspberrypi:~/webapp $ mkdir html/css
pi@raspberrypi:~/webapp $ mkdir html/img
pi@raspberrypi:~/webapp $ mkdir html/scripts
pi@raspberrypi:~/webapp $ cd html
pi@raspberrypi:~/webapp/html $ ls
css  img  scripts
pi@raspberrypi:~/webapp/html $
```

With our files created lets move to writing the codes on our PC and from then move to the Pi via filezilla.

The JavaScript Code:

The first piece of code we will write is that of the javascript. Its a simple script to communicate with the WebIOPi service.

Its important to note that for this project, our web app will consist of four buttons, which means we plan to control just four GPIO pins, although we will be controlling just two relays in our demonstration. Check the **full Video** at the end.

```
 webiopi().ready(function() {
                        webiopi().setFunction(17,"out");
                        webiopi().setFunction(18,"out");
                        webiopi().setFunction(22,"out");
                        webiopi().setFunction(23,"out");
                                                var content, button;
                        content = $("#content");
```

```
                                                button = webiopi().createGPIOButton(17," Relay
 1");
                        content.append(button);
                                                button = webiopi().createGPIOButton(18,"Relay
2");
                        content.append(button);
                                                button = webiopi().createGPIOButton(22,"Relay
3");
                        content.append(button);
                                                button = webiopi().createGPIOButton(23,"Relay
4");
                        content.append(button);
                                    });
```

The code above runs when the WebIOPi is ready.

The CSS Code:

The CSS helps us make our IoT Raspberry Pi home automation webpage look pretty.

I wanted the webpage to look like the image below and thus had to write the *smarthome.css* style sheet to achieve it.

Below we have **explained the CSS code**:

The CSS script feels too bulky to include here so I will just pick part of it and use them for the breakdown. You can **download the full CSS file** from here. CSS is easy and you can understand it just by going through the CSS code. You can easily skit this part and use our CSS code straight away.

The first part of the script represent the stylesheet for the body of the web app and its shown below;

```
 body {
        background-color:#ffffff;
        background-image:url('/img/smart.png');
        background-repeat:no-repeat;
        background-position:center;
        background-size:cover;
        font: bold 18px/25px Arial, sans-serif;
        color:LightGray;
    }
```

I want to believe the code above is self-explanatory, we set the background color as #ffffff which is white, then we add a background image located at that folder location (Remember our earlier folder setup?), we ensure the image doesn't repeat by setting the background-repeat property to no-repeat, then instruct the CSS to centralize the background. We then move to set the background size, font and color.

With the body done, we written the css for **buttons** to look pretty.

```
button {
        display: block;
        position: relative;
        margin: 10px;
        padding: 0 10px;
        text-align: center;
        text-decoration: none;
        width: 130px;
        height: 40px;
        font: bold 18px/25px Arial, sans-serif;  color: black;
        text-shadow: 1px 1px 1px rgba(255,255,255, .22);
        -webkit-border-radius: 30px;
         -moz-border-radius: 30px;
         border-radius: 30px;
```

To keep this brief, every other thing in the code was also done to make it look good. You can change them up see what happens, I think its called learning via trial and error but one good thing about CSS is things being expressed in plain English which means they are pretty easy to understand. The other part of the button block has few extras for text shadow on the button and button shadow. It also has a slight transition effect which helps it look nice and realistic when the button is pressed. These are defined separately for webkit, firefox, opera etc just to ensure the web page performs optimally across all platforms.

HTML Code:

The html code pulls everything together, javascript and the style sheet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta name="mobile-web-app-capable" content="yes">
        <meta name="viewport" content = "height = device-height, width = device-width, user-scalable = no" />
        <title>Smart Home</title>
        <script type="text/javascript" src="/webiopi.js"></script>
        <script type="text/javascript" src="/scripts/smarthome.js"></script>
        <link rel="stylesheet" type="text/css" href="/styles/smarthome.css">
        <link rel="shortcut icon" sizes="196x196" href="/img/smart.png" />
</head>
```

```
<body>
                    </br>
                    </br>
                    <div id="content" align="center"></div>
                    </br>
                    </br>
                    </br>
                    <p align="center">Push button; receive bacon</p>
                    </br>
                    </br>
</body>
</html>
```

WebIOPi Server Edits for Home Automation:

At this stage, we are ready to start creating our schematics and test our web app but before then we need to **edit the config file of the webiopi** service so its pointed to use data from our html folder instead of the config files that came with it.

To edit the configuration run the following with root permission;

```
sudo nano /etc/webiopi/config
```

Look for the http section of the config file, check under the section where you have something like, *#Use doc-root to change default HTML and resource files location*

Comment out anything under it using # then if your folder is setup like mine, point your doc-root to the path of your project file

```
doc-root = /home/pi/webapp/html
```

Save and exit. You can also change the port from 8000, in case you have another server running on the pi using that port. If not save and quit, as we move on.

Its Important to note that you can change the password of the WebIOPi service using the command;

```
sudo webiopi-passwd
```

It will prompt you for a new username and password. This can also be removed totally but security is important right?

Lastly run the WebIOPi service by issuing below command:

```
sudo /etc/init.d/webiopi start
```

The server status can be checked using;

```
sudo /etc/init.d/webiopi status
```

It can be stopped from running using;

```
sudo /etc/init.d/webiopi stop
```

To setup WebIOPi to run at boot, use;

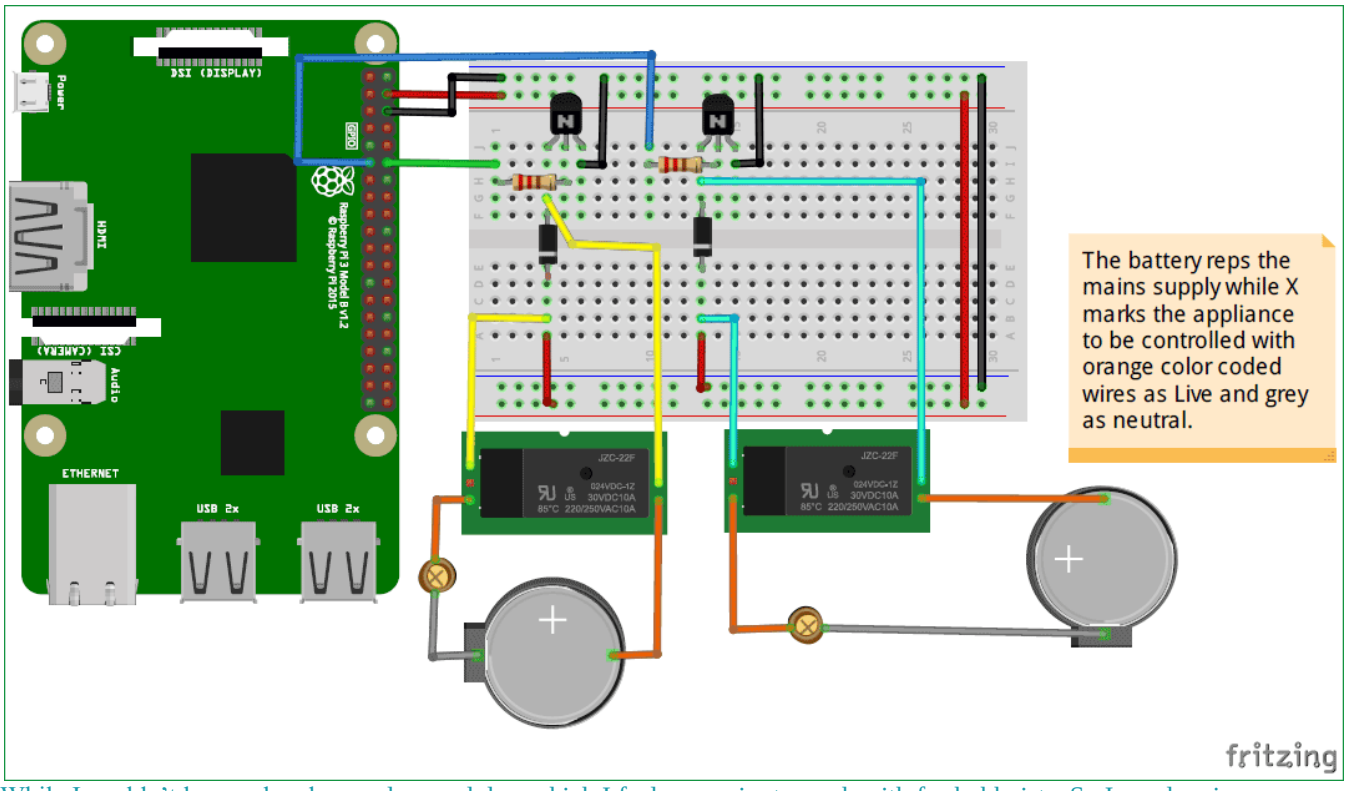
```
sudo update-rc.d webiopi defaults
```

If you want to reverse and stop it from running at boot, use;

```
sudo update-rc.d webiopi remove
```

Circuit Diagram and Explanation:

With our software set up done, we are all set to start creating the schematics for this **Web controlled Home Appliance project**.



While I couldn't lay my hands on relay modules, which I feel are easier to work with for hobbyists. So I am drawing here the schematics for ordinary standalone single 5v relays.

Connect your circuit as shown in the fritzing circuit above. You should note that COM, NO (normally open) and NC (normally Close) of your own relay may be located at different sides/points. Kindly use a millimeter to test it out. Learn more about relay here.

With our components connected, fire up your server, from a webpage, go to the IP of your Raspberry Pi and indicate the port as described earlier, login with your username and password, and you should see a webpage that looks exactly like the image below.

Now you can easily **control four AC Home appliaces from anywhere wirelessly**, just by clicking on the buttons. This will work from Mobilephone, tablet etc. and you can add more buttons and relays to control more devices. Check the **full video** below.