Practical 1: Starting Raspbian OS, Familiarising with Raspberry Pi Components and interface, Connecting to ethernet, Monitor, USB.

Hardware Guide:

for getting started with raspberry pi for the first time you will require the following hardware

1. Raspberry Pi(latest model)

2. Monitor or TV

3. HDMI Cable

4. Ethernet Cable

5. USB Keyboard

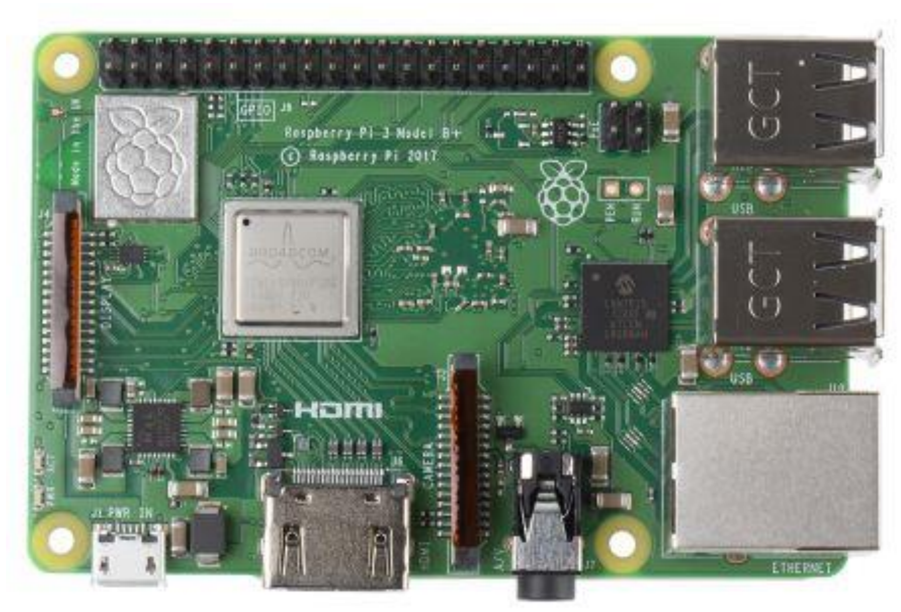
6. USB Mouse

7. Micro USB power supply

8. 8GB or larger microSD card

9. SD Card Reader

Diagram:



Raspberry Pi 3 Model B:

The Raspberry Pi 3 is the third generation Raspberry Pi. it replaced the Raspberry Pi 2 Model B in the February 2016. Compared to the Raspberry Pi 2 it has:

A 1.2 GHz 64-bit quad-core ARMv8 CPU

802.11n Wireless LAN

Bluetooth 4.1

Bluetooth Low Energy(BLE)

Like the Pi 2, it also has:

4 USB Ports

40 GPIO Pins

Full HDMI Port

Ethernet Port

Combined 3.5mm audio jack and composite video

Camera interface(CSI)

Display interface(DSI)

Micro SD Card slot(now push-pull rather than push-push)

Video Core iV 3D graphics core

The Raspberry Pi 3 has an identical form factor to the previous Pi 2(and Pi 1 Model B+)and has complete compatibility with Raspberry Pi 1 and 2.

Monitor or TV:

A monitor or TV with HDMI in can be used as a display with a Raspberry Pi. Most modern television sets and monitors have an HDMI port, and are the easiest to get working thye Raspberry Pi. You can use an HDMI cable to connect the Raspberry Pi directly to the television or monitor.

Some older monitors have a DVI port. These work well with the Raspberry Pi, althought you"ll need an HDMI-to-DVI adapter to attach to an HDMI cable, or a one-piece HDMI-to-DVI cable. Some old monitors have a VGA port. These can be trickier to use as you'll need an HDMI-to-VGA converter, which can change digital video to analogue video. A simple port adapter won't work.

HDMI to HDMI Cable:

Connect Raspberry Pi to a Monitor or TV with a HDMI to HDMI cable.

Ethernet cable:

Ethernet cable will allow your Pi to connect with the internet. It is also useful for headless setup of Raspberry Pi.

USB Keyboard and Mouse:

Any standard USB Keyboard and mouse can be used with the Raspberry Pi. This plug and play devices will work without any additional driver. Simply plug them into the Raspberry Pi and they should be recognised when it starts up.

Power Supply:

It is recommended that you use a 5V, 2A USB power supply for all models of Raspberry Pi.

SD Card:

The latest version of Raspbian, the default operating system recommended for the Raspberry Pi, requires an 8GB(or larger) micro SC card. SD card will store the operating systems as well as all the file and applications created by you.

Installation Guide:

Now since you have all the required hardware, we will now learn how to get the operating system onto your microSC card so that you can start using software on your Raspberry Pi.

Get Raspbian OS on your microSD card:

Raspbian comes pre-installed with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java, Mathematica and more.

1. To download Raspbian log on to raspberrpi.org and click on the download, then click on Raspbian and lastly download the RASPBIAN JESSIE WITH DESKTOP file. You can choose either the Torrent file or ZIP file

2. The downloaded file will be in zip format. To unzip the file, you will require an unzip tool. You can use any unzipping tool via WINRAR, 7ZIP etc. After unzipping the file, you will find a disc image file in the unzipped folder.

3. Now format the SD Card before writing the disc image file on the SD card. You can use SDFormatter tool or any other tool of your wish.

4. To write the image file of the operating system on the SD card you will require a Disk Imagger tool. for this you can use Win32 Disk Imagger tool.

5. Once the image is written on the SD Card, your untitled SD card will now have the name boot. Your SD Card will now hold the Raspbian Operating System required for the first-time setup.

Plugging in your Raspberry Pi:

1. Begin by placing your SD card into the SD card slot on the Raspberry Pi. it will only fit one way.

2. Next, plug your keyboard and mouse into the USB ports on the Raspberry Pi.

3. Make sure that your monitor or TV is turned on, and that you have selected the right input(e.g. HDMI 1, DVI, etc).

4. Connect your HDMI cable from your Raspberry Pi to your monitor or TV.

5. If you intend to connect your Raspberry Pi to the internet, plug an Ethernet cable into the Ethernet port, or connect a WI-Fi dongle to one of the USB ports(unless you have a Raspberry Pi 3).

6. when you're happy that you have plugged all the cables and SD card in correctly, connect the micro USB power supply. This action will turn on and boot your Raspberry Pi.

Practical 2:Displaying different LED patterns with Raspberry Pi.

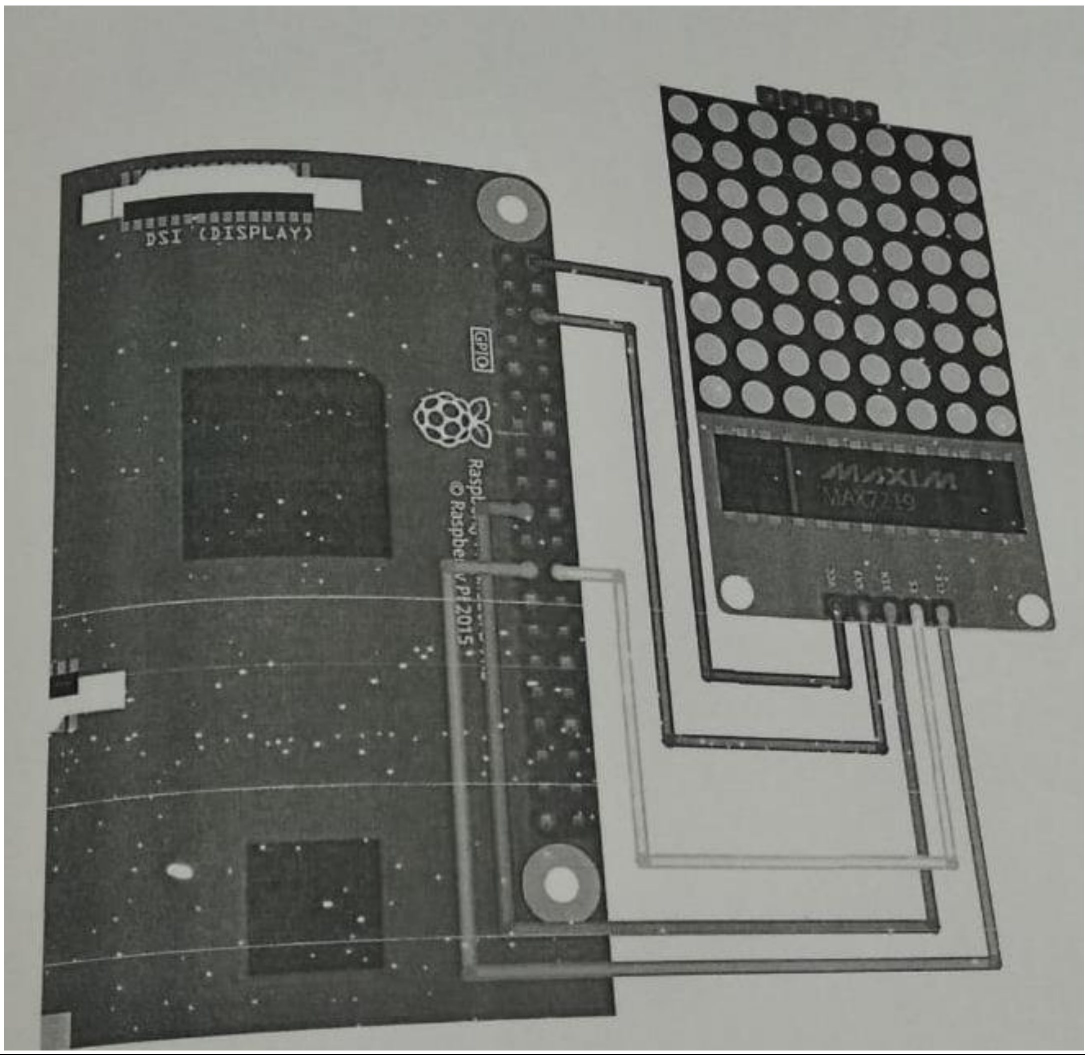
Hardware Requirements:

1. Led
2. Resistor
3. Connecting Wires
4. Breadboard

Software Requirements:

1. Python
2. Raspbian OS

Circuit Diagram:



Code:

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BOARD)

led1 = 29

led2 = 31

led3 = 33

led4 = 35

led5 = 36

led6 = 37

led7 = 38

led8 = 40

#setup the ledPin as output

GPIO.setup(led1, GPIO.OUT)

GPIO.setup(led2, GPIO.OUT)

GPIO.setup(led3, GPIO.OUT)

GPIO.setup(led4, GPIO.OUT)

GPIO.setup(led5, GPIO.OUT)

GPIO.setup(led6, GPIO.OUT)

GPIO.setup(led7, GPIO.OUT)

GPIO.setup(led8, GPIO.OUT)

GPIO.output(led1, False)
GPIO.output(led2, False)

GPIO.output(led3, False)

GPIO.output(led4, False)

GPIO.output(led5, False)

GPIO.output(led6, False)

GPIO.output(led7, False)

GPIO.output(led8, False)

def ledpattern(ledVal1, ledVal2, ledVal3, ledVal4, ledVal5, ledVal6, ledVal7,
ledVal8):

    GPIO.output(led1, ledVal1)

    GPIO.output(led2, ledVal2)

    GPIO.output(led3, ledVal3)

    GPIO.output(led4, ledVal4)

    GPIO.output(led5, ledVal5)

    GPIO.output(led6, ledVal6)

    GPIO.output(led7, ledVal7)

    GPIO.output(led8, ledVal8)

def patternOne():

    for i in range (0,3):

        ledpattern(1,0,1,0,1,0,1,0)

        time.sleep(1)

        ledpattern(0,1,0,1,0,1,0,1)
        time.sleep(1)
def patternTwo():
    for i in range (0, 3):
        ledpattern(1,0,0,0,0,0,0,0)
        time.sleep(0.1)
        ledpattern(0,1,0,0,0,0,0,0)
        time.sleep(0.1)
        ledpattern(0,0,1,0,0,0,0,0)
        time.sleep(0.1)
        ledpattern(0,0,0,1,0,0,0,0)
        time.sleep(0.1)
        ledpattern(0,0,0,0,1,0,0,0)
        time.sleep(0.1)
        ledpattern(0,0,0,0,0,1,0,0)
        time.sleep(0.1)
        ledpattern(0,0,0,0,0,0,1,0)
        time.sleep(0.1)
        ledpattern(0,0,0,0,0,0,0,1)
        time.sleep(0.1)
def patternThree():
    for i in range (0, 3):
        ledpattern(0,0,0,0,0,0,0,1)
        time.sleep(0.1)

        ledpattern(0,0,0,0,0,0,1,0)

        time.sleep(0.1)

        ledpattern(0,0,0,0,0,1,0,0)

        time.sleep(0.1)

        ledpattern(0,0,0,0,1,0,0,0)

        time.sleep(0.1)

        ledpattern(0,0,0,1,0,0,0,0)

        time.sleep(0.1)

        ledpattern(0,0,1,0,0,0,0,0)

        time.sleep(0.1)

        ledpattern(0,1,0,0,0,0,0,0)

        time.sleep(0.1)

        ledpattern(1,0,0,0,0,0,0,0)

        time.sleep(0.1)

try:

    while True:

        patternOne()

        patternTwo()

        patternThree()

finally:

    GPIO.cleanup()
```

Practical 3 : Displaying Time over 4-Digit 7-Segment Display using Raspberry Pi

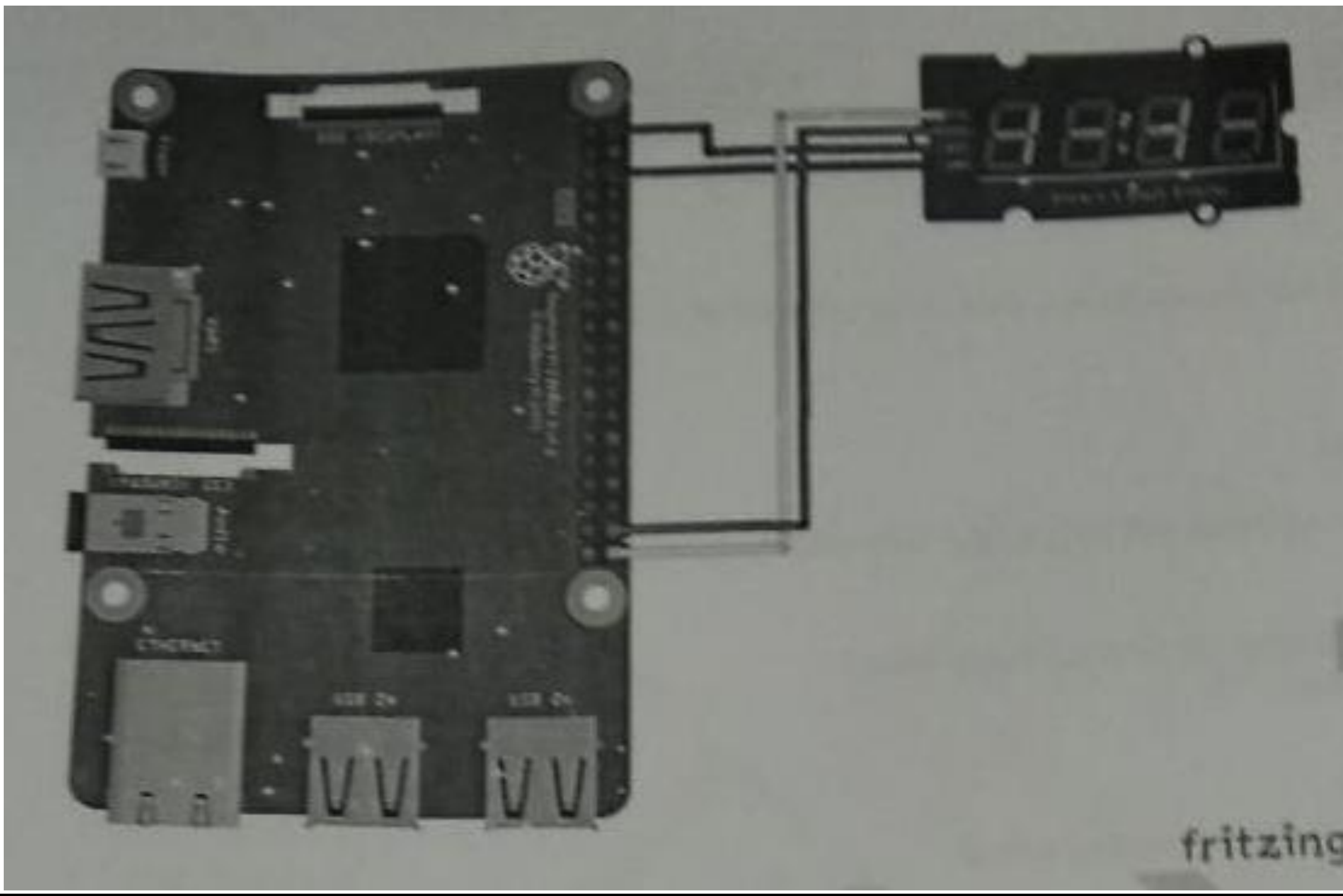
Hardware Requirements:

1. TM1637 4-Digit seven segment.
2. Connecting Wires

Software Requirements:

1. Python
2. Raspbian Os
3. Actor-led-7segment-4numbers file from github

Circuit Diagram:



Code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from time import sleep
import tm1637
try:
    import thread
except ImportError:
    import _thread as thread
# Initialize the clock (GND, VCC=3.3V, Example Pins are DIO-20 and CLK21)
Display = tm1637.TM1637(CLK=21, DIO=20, brightness=1.0)
try:
    print ("Starting clock in the background (press CTRL + C to stop)")
    Display.StartClock(military_time=False)
    print ('Continue Python script and tweak Display!')
    sleep(5)
    Display.ShowDoublepoint(False)
    sleep(5)
    loops = 5
    while loops > 0:
        for i in range(0, 10):
            Display.SetBrightness(i / 10.0)
```

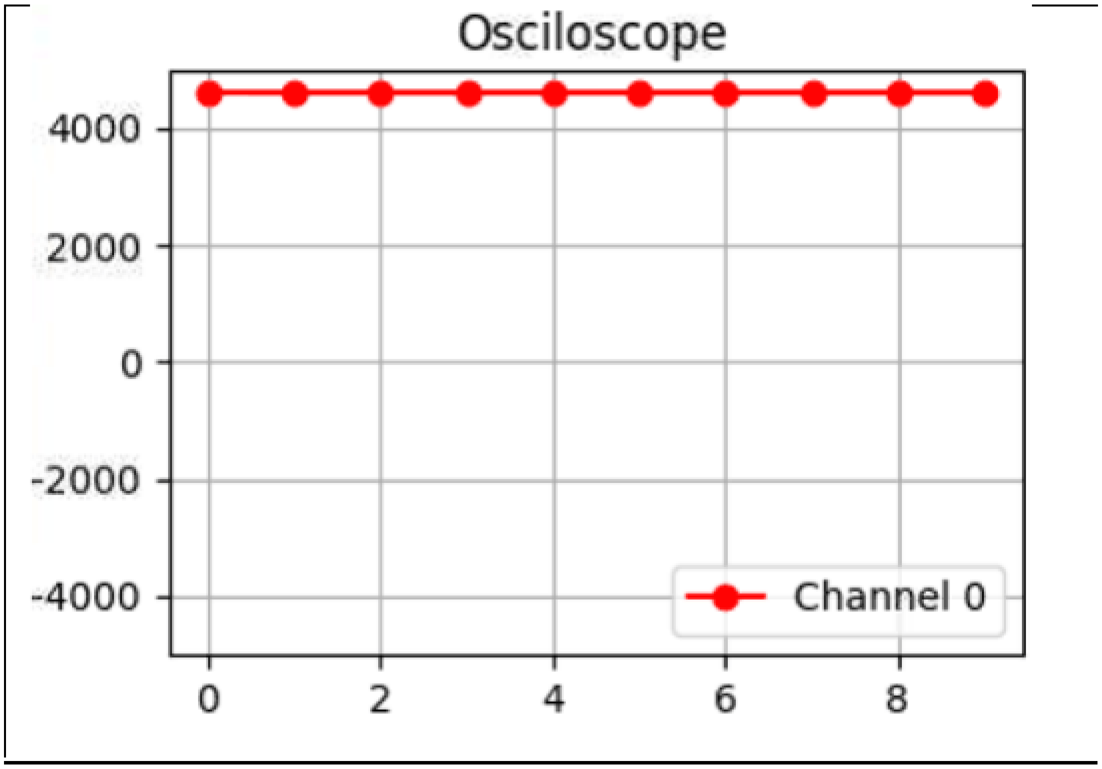
```
          sleep(0.5)
        loops -= 1
    Display.StopClock()
    thread.interrupt_main()
except KeyboardInterrupt:
 print ("Properly closing the clock and open GPIO pins")
    Display.cleanup()
```

Practical 4: Raspberry Pi Based Oscilloscope

Hardware Requirements:

Software Requirements:

Circuit Diagram:



Code:

```
import time

import matplotlib.pyplot as plt

#import numpy

from drawnow import *

# Import the ADS1x15 module.

import Adafruit_ADS1x15
```

```
# Create an ADS1115 ADC (16-bit) instance.

adc = Adafruit_ADS1x15.ADS1115()

GAIN = 1

val = []

cnt = 0

plt.ion()

adc.start_adc(0, gain=GAIN)

print('Reading ADS1x15 channel 0')

def makeFig():

   plt.ylim(-5000,5000)

   plt.title('Osciloscope')

   plt.grid(True)

   plt.ylabel('ADC outputs')

   plt.plot(val, 'ro-', label='Channel 0')

   plt.legend(loc='lower right')

while (True):

   # Read the last ADC conversion value and print it out.

   value = adc.get_last_result()

   print('Channel 0: {0}'.format(value))

   # Sleep for half a second.

   time.sleep(0.5)

   val.append(int(value))
```

```
    drawnow(makeFig)

    plt.pause(.000001)

    cnt = cnt+1

if(cnt>50):

        val.pop(0)
```

<u>Practical 5: Setting up Wireless Access Point using Raspberry Pi</u>

<u>Required Components:</u>

1. Raspberry Pi 2
2. 8GB SD card
3. WiFi USB dongle
4. Ethernet cable
5. Power supply for the Pi.
6. Monitor (optional)
7. Keyboard (optional)
8. Mouse (optional)

Steps for Setting up Raspberry Pi as Wireless Access Point:

Step 1: Update the Pi

```
sudo apt-get update
```

followed by;

```
sudo apt-get upgrade
```

With the update done, reboot your pi to effect changes.

Step 2: Install "*dnsmasq*" and "*hostapd*"

Next, we install the software that makes it possible to setup the pi as a wireless access point and also the software that helps assign network address to devices that connect to the AP. We do this by running;

```
sudo apt-get install dnsmasq
```

followed by;

```
sudo apt-get install hostapd
```

or you could combine it by running;

```
sudo apt-get install dnsmasq hostapd
```

Step 3: Stop the software from Running

Since we don't have the software configured yet there is no point running it, so we disable them from running in the underground. To do this we run the following commands to stop the *systemd* operation.

```
sudo systemctl stop dnsmasq
```

```
sudo systemctl stop hostapd
```

Step 4: Configure a Static IP address for the wireless Port

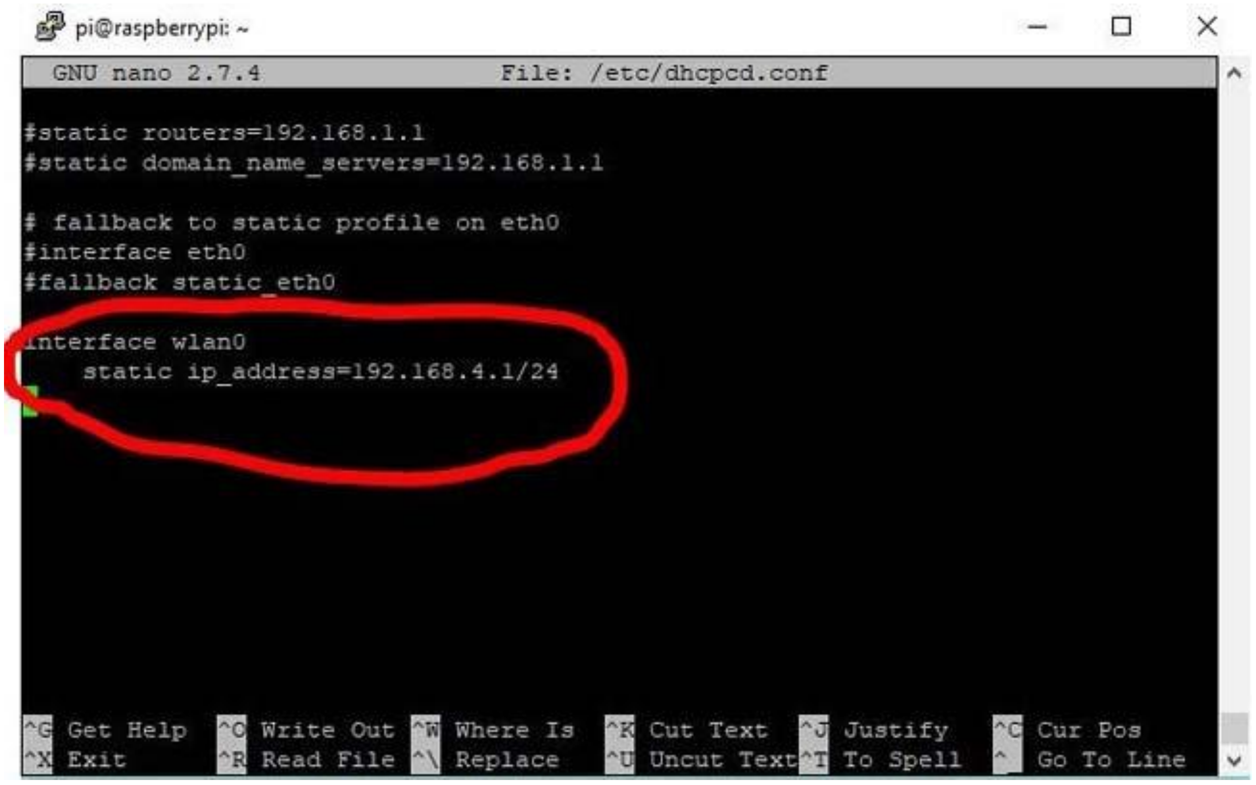
Confirm the *wlan* port on which the wireless device being used is connected. For my Pi, the wireless is on wlan0. **Setting up the Raspberry Pi to act as a server** requires us to assign a static IP address to the wireless port. This can be done by editing the *dhcpcd* config file. To edit the configuration file, run;

```
sudo nano /etc/dhcpcd.conf
```

Scroll to the bottom of the config file and add the following lines.

```
Interface wlan0

static ip_address=192.168.4.1/24
```

After adding the lines, the config file should look like the image below.



```
pi@raspberrypi: ~
  GNU nano 2.7.4                 File: /etc/dhcpcd.conf

#static routers=192.168.1.1
#static domain_name_servers=192.168.1.1

# fallback to static profile on eth0
#interface eth0
#fallback static_eth0

interface wlan0
    static ip_address=192.168.4.1/24


^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

Note: This IP address can be changed to suit your preferred configuration.

Save the file and exit using; ctrl+x followed by Y

Restart the *dhcpcd* service to effect the changes made to the configuration using;

Sudo service dhcpcd restart

Step 5: Configure the *dhcpcd* server

With a static IP address now configured for the Raspberry Pi wlan, the next thing is for us to configure the *dhcpcd*server and provide it with the range of IP addresses to be assigned to devices that connect to the wireless access point. To do this, we need to edit the configuration file of the *dnsmasq* software but the config file of the software contains way too much info and a lot could go wrong If not properly edited, so instead of editing, we will be creating a new config file with just the amount of information that is needed to make the wireless access point fully functional.

Before creating the new config file, we keep the old on safe by moving and renaming it.

```
sudo mv /etc/dnsmasq.conf  /etc/dnsmasq.conf.old
```

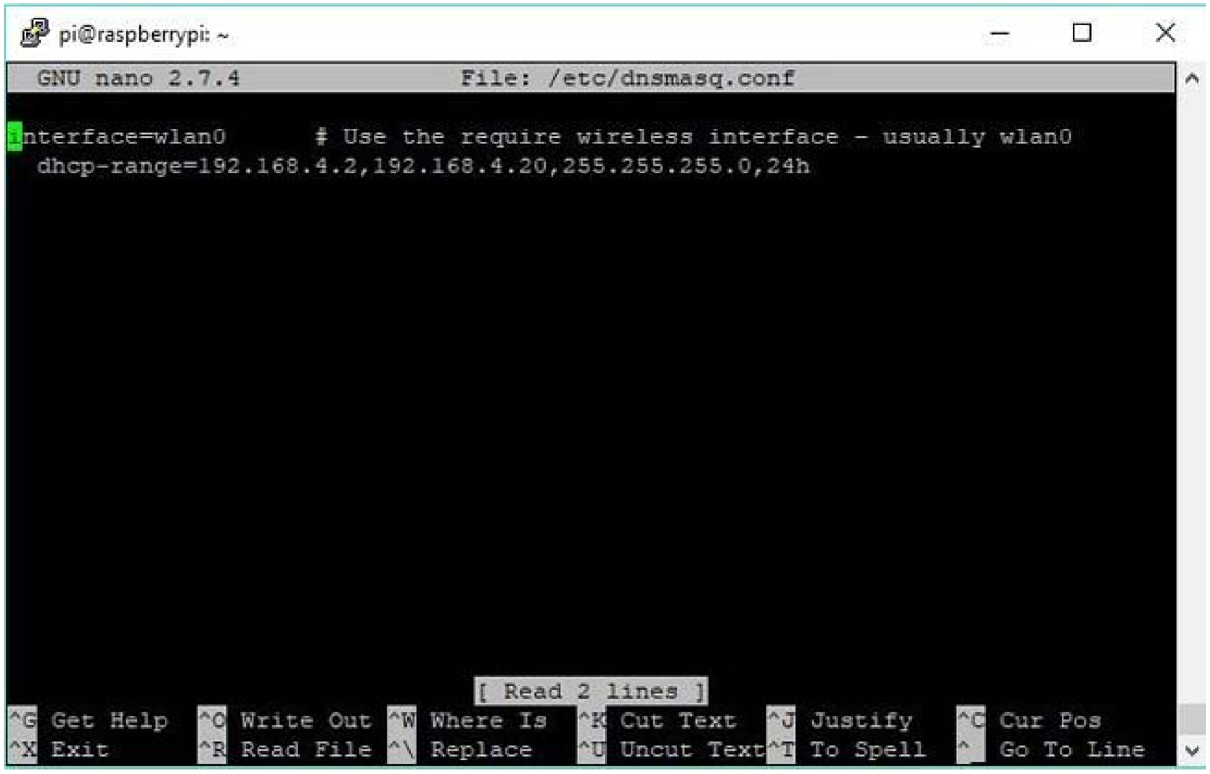
Then launch the editor to create a new configuration file;

```
sudo nano /etc/dnsmasq.conf
```

with the editor launched, copy the lines below and paste in or type directly into it.

```
Interface = wlan0  #indicate the communication interface which is usually wlan0 for wireless

dhcp-range = 192.168.4.2, 192.168.4.20, 255.255.255.0,24h
```

the content of the file should look like the image below.



```
  GNU nano 2.7.4              File: /etc/dnsmasq.conf

interface=wlan0      # Use the require wireless interface - usually wlan0
  dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h






                                [ Read 2 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Save the file and exit. The content of this config file is just to specify the range of IP address that can be assigned to devices connected to the wireless access point.

With this done, we will be able to give an identity to devices on our network.

The next set of steps will help us configure the access point host software, setup the ssid, select the encrytpion etc.

Step 6: Configure *hostapd* for SSID and Password

We need to edit the *hostapd* config file(run *sudo nano /etc/hostapd/hostapd.conf*) to add the various parameters for the wireless network being setup including the ssid and password. Its should be noted that the password (passphrase) should be between 8 and 64 characters. Anything lesser won’t work.

interface=wlan0

driver=nl80211

ssid=piNetwork

hw_mode=g

channel=7

wmm_enabled=0

macaddr_acl=0

auth_algs=1

ignore_broadcast_ssid=0

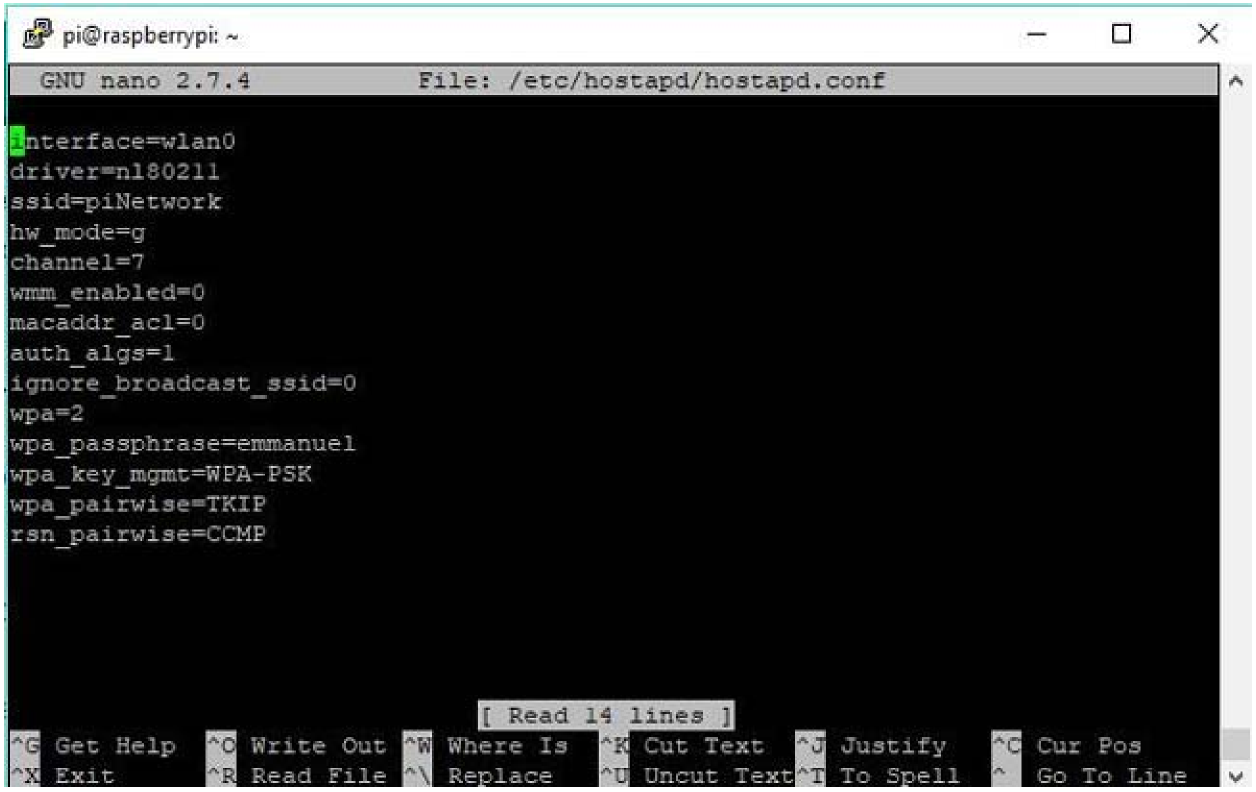
wpa=2

wpa_passphrase=emmanuel # use a very secure password and not this

wpa_key_mgmt=WPA-PSK

wpa_pairwise=TKIP

rsn_pairwise=CCM



```
  GNU nano 2.7.4              File: /etc/hostapd/hostapd.conf

interface=wlan0
driver=nl80211
ssid=piNetwork
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=emmanuel
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP




                              [ Read 14 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text^T To Spell    ^  Go To Line
```

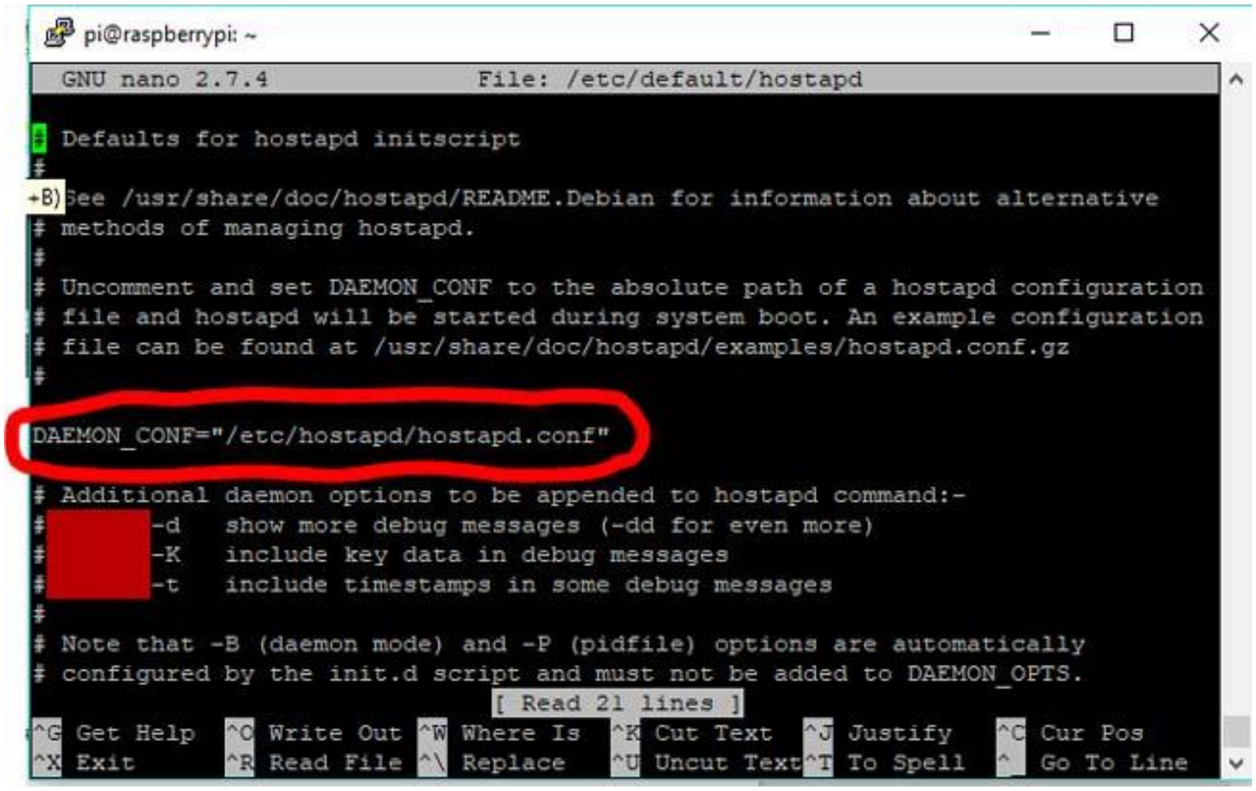
Feel free to change the ssid and password to suit your needs and desire.

Save the config file and exit.

After the config file has been saved, we need to point the hostapd software to where the config file has been saved. To do this, run;

sudo nano /etc/default/hostapd

find the line with daemon_conf commented out as shown in the image below.



```
pi@raspberrypi: ~
  GNU nano 2.7.4              File: /etc/default/hostapd

# Defaults for hostapd initscript
#
+B)See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#

DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
# [redacted] -d   show more debug messages (-dd for even more)
#           -K   include key data in debug messages
#           -t   include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
                               [ Read 21 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

Uncomment the DAEMON_CONF line and add the line below in between the quotes in front of the “equal to” sign.

/etc/hostapd/hostapd.conf

Step 7: Fire it up

Since we disabled the two software initially, to allow us configure them properly, we need to restart the system after configuration to effect the changes.

Use;

sudo systemctl start hostapd

sudo systemctl start dnsmasq

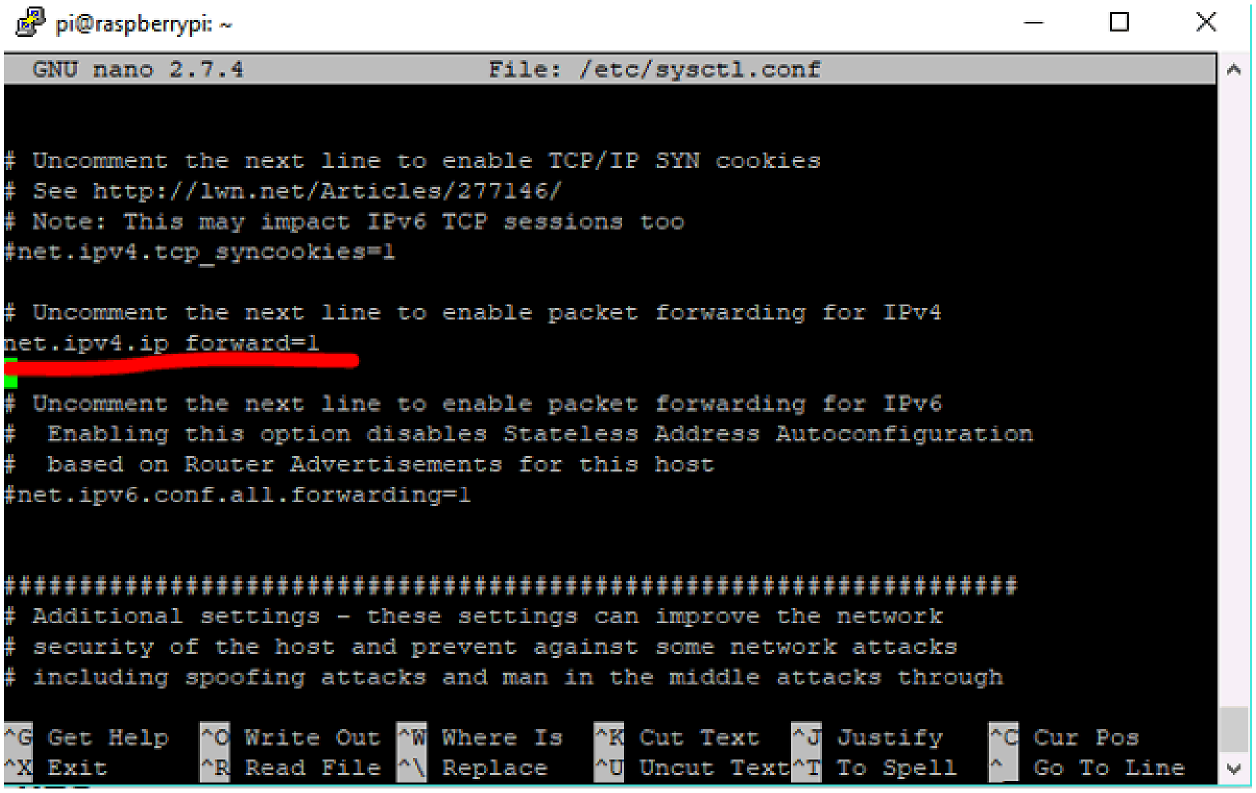
Step 8: Routing and masquerade for outbound traffic

We need to add routing and masquerade for outbound traffic.

To do this, we need to edit the config file of the *systemctl* by running:

sudo nano /etc/sysctl.conf

Uncomment this line ***net.ipv4.ip_forward=1***(highlighted in the image below)



```
  GNU nano 2.7.4                File: /etc/sysctl.conf


# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1


###################################################################
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text^T To Spell    ^  Go To Line
```

Save the config file and exit using ctrl+x followed by y.

Next we move to masquerading the outbound traffic. This can be done by making some changes to the iptable rule. To do this, run the following commands:

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

then save the Iptables rule using:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Step 9: Create Wireless Access Point on startup:

For most wireless access point application, it is often desired that the access point comes up as soon as the system boots. To implement this on the raspberry pi, one of the easiest ways is to add instructions to run the software in the *rc.local* file so we put commands to install the iptable rules on boot in the rc.local file.

To edit the rc.local file, run:

```
sudo nano /etc/rc.local
```

and add the following lines at the bottom of the system, just before the exit 0 statement

```
iptables-restore < /etc/iptables.ipv4.nat
```

Step 9: Reboot! and Use

At this stage, we need to reboot the system to effect all the changes and test the wireless access point starting up on boot with the iptables rule updated.

Reboot the system using:

```
sudo reboot
```

As soon as the system comes back on, you should be able to access the wireless access point using any Wi-Fi enabled device and the password used during the setup.

Accessing the Internet from the Raspberry Pi’s Wi-Fi Hotspot

Oh yes, so I will be adding a bonus tutorial to show how the wireless access point created can be used to provide internet connection for the devices connected to it.

The internet access distributed to the devices is provided via the Ethernet port on the Pi which can be connected to a router or any similar devices.

To implement this, we need to put a “bridge” in between the wireless device and the Ethernet device on the Raspberry Pi (the wireless access point) to pass all traffic between the two interfaces. To set this up, we will use the *bridge-utils*software. Install *hostapd* and *bridge-utils*. While we have installed *hostapd* before, run the installation again to clear all doubts.

```
sudo apt-get install hostapd bridge-utils
```

Next, we stop hostapd so as to configure the software.

```
sudo systemctl stop hostapd
```

When a bridge is created, a higher level construct is created over the two ports being bridged and the bridge thus becomes the network device. To prevent conflicts, we need to stop the allocation of IP addresses by the DHCP client running on the Raspberry Pi to the eth0 and wlan0 ports. This will be done by editing the config file of the dhcpcd client to include ***denyinterfaces wlan0*** and ***denyinterfaces eth0*** as shown in the image below.

The file can be edited by running the command;

```
sudo nano /etc/dhcpcd.conf
```

Note: *From this point on, ensure you don’t disconnect the Ethernet cable from your PC if you are running in headless mode as you may not be able to connect via SSH again since we have disabled the Ethernet port. If working with a monitor, you have nothing to fear.*

Next, we create a new bridge called br0

```
sudo brctl addbr br0
```

Next, we connect the ethernet port (eth0) to the bridge (br0) using;

```
sudo brctl addif br0 eth0
```

Next, we edit the interfaces file using ***sudo nano /etc/network/interfaces*** so various devices can work with the bridge. Edit the interfaces file to include the information below;

```
#Bridge setup

auto br0

iface br0 inet manual

bridge_ports eth0 wlan0
```

Lastly we edit the hostapd.conf file to include the bridge configuration. This can be done by running the command: ***sudo nano /etc/hostapd.conf*** and editing the file to contain the information below. Note the bridge was added below the wlan0 interface and the driver line was commented out.

```
interface=wlan0

bridge=br0

#driver=nl80211

ssid=NameOfNetwork

hw_mode=g

channel=7

wmm_enabled=0

macaddr_acl=0

auth_algs=1

ignore_broadcast_ssid=0

wpa=2
```

```
wpa_passphrase=AardvarkBadgerHedgehog

wpa_key_mgmt=WPA-PSK

wpa_pairwise=TKIP

rsn_pairwise=CCMP
```

With this done, save the config file and exit.

To effect the changes made to the Raspberry Pi, **reboot** the system. Once it comes back up, you should now be **able to access the internet by connecting to the Wireless access point created by the Raspberry Pi**. This of course will only work if internet access is available to the pi via the Ethernet port.

While this project can be used to extend Wi-Fi around the house or office or an entire compound, there are several applications I find very interesting and useful like the raspberry pi as a home automation hub so several Wi-Fi enabled home automation devices can connect to the internet using the raspberry pi’s wireless access point. Do you have any other cool Idea, to which this can be applied, feel free to share via the comment section to inspire others.

Testing Raspberry Pi Wireless Access Point:

To test these instructions, use a mobile phone or any other device capable of connecting to a WiFi hotspot network, you should see the name pop up. You can then connect to it by using that terrible password we specified “emmanuel”. Be sure to use a more secure password when implementing. I only used that password to make things easier to follow.

Also note, it might take a while for the Wireless access point to become visible after reboot as the Pi needs to boot up before the network activities start.

Practical 6 : Controlling Raspberry Pi with Telegram

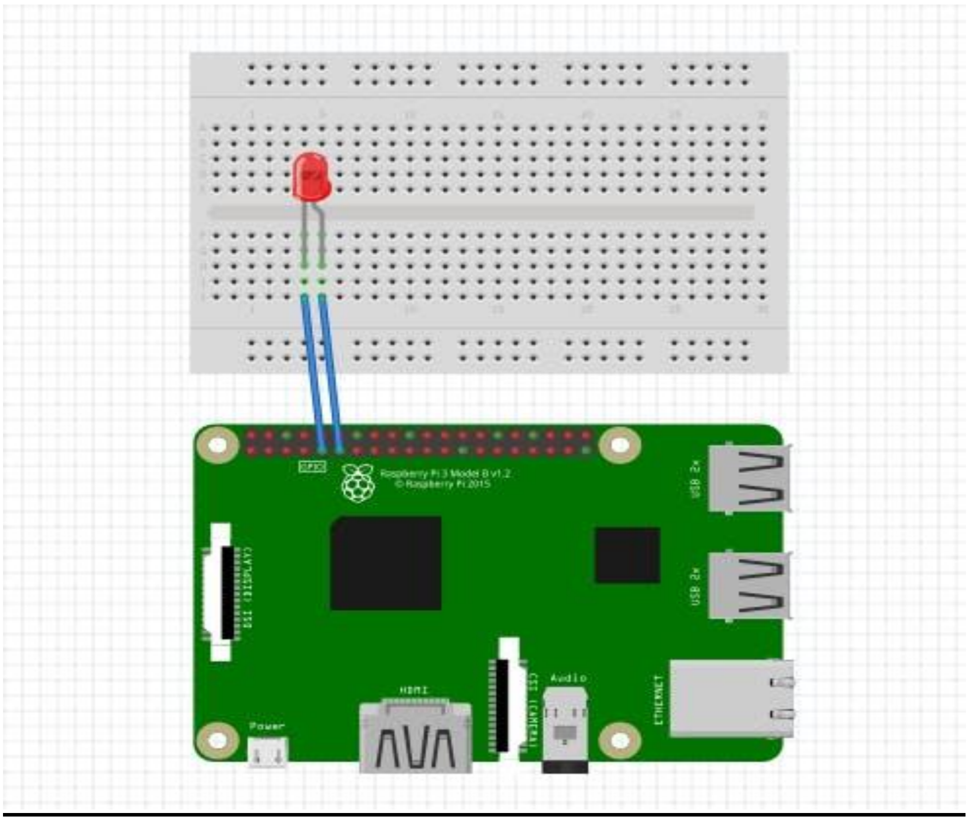
Hardware Requirements:

1. Mobile with Telegram and Registered .
2. LED
3. Breadboard
4. Connecting wires.

Software Requirements:

1. Python
2. Telegram APP
3. RASPBIAN OS

Circuit Diagram:



Setting up Telegram:

Code:

Step 1: Install Telegram on Your Phone, Obviously

Go to App Store (iPhone) or Play Store (Android), download and install Telegram on your phone.

Now, *you* can use Telegram. Not yet for the Raspberry Pi. Telegram reserves a special kind of accounts for machines, called **bot accounts**. As the owner of your own Pi, *you* have to obtain a bot account for it.

Step 2: Text /newbot to BotFather
Open Telegram on your phone, search for a user called **BotFather**. As the name implies, he is the Father of All Bots.As you may have guessed, he is not of our own species, but is actually a machine. He accepts special commands, because he does not understand plain English very well.

To obtain a bot account, text him **/newbot**. (you need the slash '/' in front) He will then ask a couple of questions. I call my bot "Led". You will see why in a few moments. But you can give it any name you want.

At the end of process, you will be given a**token**, something like 123456789:ABCdefGhIJKlmNo-PQRsTUVwxyZ. This token represents the bot account. You are going to put this token on the Pi.

Step 3: Install Telepot on Raspberry Pi

Enter the Pi, via SSH or a USB-TTL serial cable. Install **telepot**, a Python package that enables the Pi to speak Telegram Bot API.

On the command line, run these two commands:
sudo apt-get install python-pip
sudo pip install telepot

Step 4: Test Token

On the command line, type **python** to enter the Python interpreter.
In the Python interpreter, enter these three lines;

```
import teleport
bot = telepot.Bot('*** copy bot token from botfather ***')
bot.getMe()
```

You should keep bot token secret too. Having the token means having access to the bot account.
If the last command, **getMe()**, returns a dictionary describing the bot account (as in the screenshot), all is good. Type **exit()** to leave the Python interpreter.

If not, you have copied the wrong token. Type **exit()** to leave the Python interpreter. Then type **python** to come in again, and repeat those three lines of code.

Step 5: Python code?
Create a python file name telegram.py in raspberry pi and copy the following code.

```
import sys
import time
import random
import datetime
import telepot
import RPi.GPIO as GPIO
from telepot.loop import MessageLoop

red=23# connect red led at pin 23 of raspberry pi

now=datetime.datetime.now()
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

GPIO.setup(red,GPIO.OUT)
GPIO.output(red,0)

def action(msg):
   chat_id = msg['chat']['id']
   command = msg['text']
   print ('Got command: %s' %command)
   if 'On' in command:
      message="Turn On"
      message=message+" red"
      GPIO.output(red,1)
      bot.sendMessage(chat_id,message)

   if 'Off' in command:
      message="Turn Off"
      message=message+" red"
      GPIO.output(red,0)
      bot.sendMessage(chat_id, message)
```

bot = telepot.Bot('626665131:AAHsNzQbqSj9GZ9-w2t4I')**#paste your bot tokan here.**

```
print(bot.getMe())
MessageLoop(bot,action).run_as_thread()
print ('I am listening...')

while 1:
    time.sleep(10)
```

Step 6: Run It and Text It

Assuming you have named the file you have just saved "telegram.py", to run the bot, type

sudopython /home/pi/telegram.py

Open Telegram on your phone, search for your bot using its name or username. Text it **On** or **Off**, and see how it responds.

Practical 7: Fingerprint Sensor interfacing with Raspberry Pi

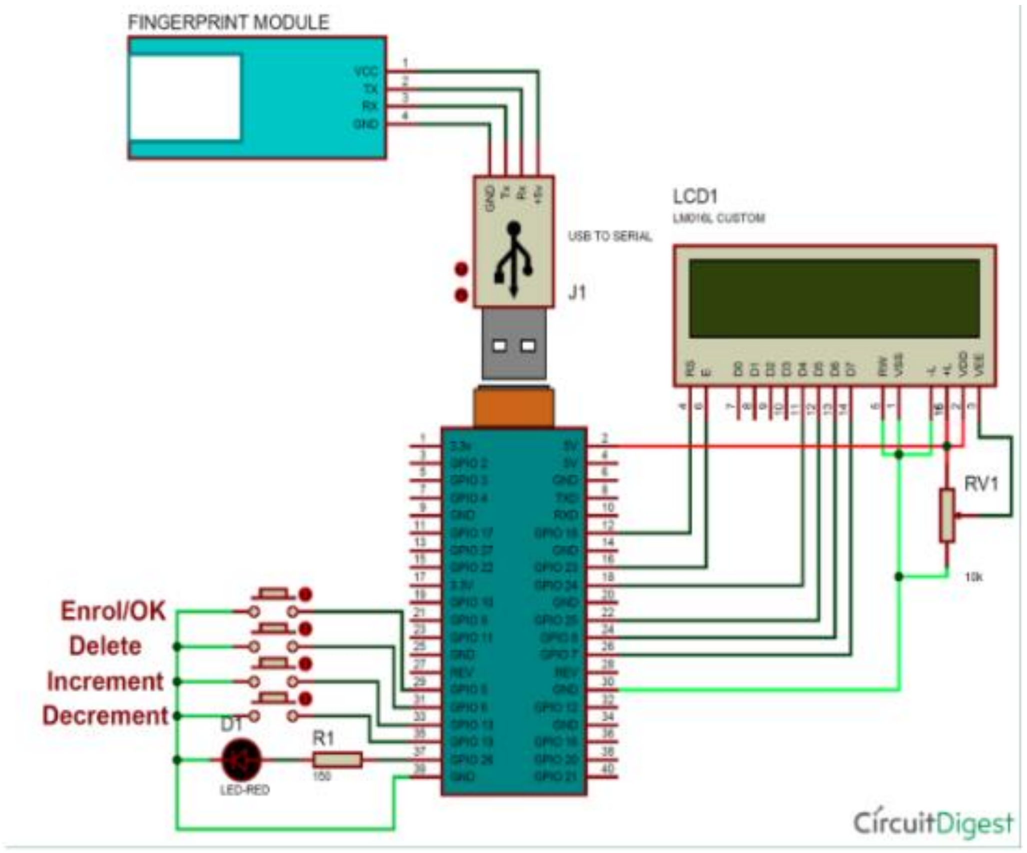
Hardware Requirements:

1. Fingerprint sensor
2. USB to TTL/UART converter
3. Connecting wires
4. Push Buttons
5. 16x2 LCD
6. LED
7. Breradboard

Software Requirements:

1. Python
2. Raspbian OS
3. pyfingerprint repo from github

Circuit Diagram:



<u>Code:</u>

```
import time
from pyfingerprint.pyfingerprint import PyFingerprint
import RPi.GPIO as gpio
RS =18
EN =23
D4 =24
D5 =25
D6 =8
D7 =7
enrol=5
delet=6
inc=13
dec=19
led=26
HIGH=1
LOW=0
gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(RS, gpio.OUT)
gpio.setup(EN, gpio.OUT)
gpio.setup(D4, gpio.OUT)
gpio.setup(D5, gpio.OUT)
gpio.setup(D6, gpio.OUT)
gpio.setup(D7, gpio.OUT)
gpio.setup(enrol, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(delet, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(inc, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(dec, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(led, gpio.OUT)
try:
    f = PyFingerprint('/dev/ttyUSB0', 57600, 0xFFFFFFFF, 0x00000000)
    if ( f.verifyPassword() == False ):
        raise ValueError('The given fingerprint sensor password is wrong!')
except Exception as e:
    print('Exception message: ' + str(e))
    exit(1)
def begin():
  lcdcmd(0x33)
  lcdcmd(0x32)
  lcdcmd(0x06)
  lcdcmd(0x0C)
  lcdcmd(0x28)
  lcdcmd(0x01)
  time.sleep(0.0005)
```

```
def lcdcmd(ch):
  gpio.output(RS, 0)
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)

def lcdwrite(ch):
  gpio.output(RS, 1)
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
def lcdclear():
  lcdcmd(0x01)
```

```
def lcdprint(Str):
 l=0;
 l=len(Str)
 for i in range(l):
   lcdwrite(ord(Str[i]))
def setCursor(x,y):
   if y == 0:
      n=128+x
   elif y == 1:
      n=192+x
   lcdcmd(n)
def enrollFinger():
   lcdcmd(1)
   lcdprint("Enrolling Finger")
   time.sleep(2)
   print('Waiting for finger...')
   lcdcmd(1)
   lcdprint("Place Finger")
   while ( f.readImage() == False ):
      pass
   f.convertImage(0x01)
   result = f.searchTemplate()
```

```
positionNumber = result[0]
if ( positionNumber >= 0 ):
    print('Template already exists at position #' + str(positionNumber))
    lcdcmd(1)
    lcdprint("Finger ALready")
    lcdcmd(192)
    lcdprint("   Exists     ")
    time.sleep(2)
    return
print('Remove finger...')
lcdcmd(1)
lcdprint("Remove Finger")
time.sleep(2)
print('Waiting for same finger again...')
lcdcmd(1)
lcdprint("Place Finger")
lcdcmd(192)
lcdprint("   Again    ")
while ( f.readImage() == False ):
    pass
f.convertImage(0x02)
if ( f.compareCharacteristics() == 0 ):
    print ("Fingers do not match")
```

```
        lcdcmd(1)
        lcdprint("Finger Did not")
        lcdcmd(192)
        lcdprint("   Mactched   ")
        time.sleep(2)
        return
    f.createTemplate()
    positionNumber = f.storeTemplate()
    print('Finger enrolled successfully!')
    lcdcmd(1)
    lcdprint("Stored at Pos:")
    lcdprint(str(positionNumber))
    lcdcmd(192)
    lcdprint("successfully")
    print('New template position #' + str(positionNumber))
    time.sleep(2)
def searchFinger():
    try:
        print('Waiting for finger...')
        while( f.readImage() == False ):
            #pass
            time.sleep(.5)
            return
        f.convertImage(0x01)
        result = f.searchTemplate()
        positionNumber = result[0]
        accuracyScore = result[1]
        if positionNumber == -1 :
            print('No match found!')
            lcdcmd(1)
            lcdprint("No Match Found")
            time.sleep(2)
            return
        else:
            print('Found template at position #' + str(positionNumber))
            lcdcmd(1)
            lcdprint("Found at Pos:")
            lcdprint(str(positionNumber))
            time.sleep(2)
    except Exception as e:
        print('Operation failed!')
        print('Exception message: ' + str(e))
        exit(1)
def deleteFinger():
    positionNumber = 0
    count=0
```

```
lcdcmd(1)
lcdprint("Delete Finger")
lcdcmd(192)
lcdprint("Position: ")
lcdcmd(0xca)
lcdprint(str(count))
while gpio.input(enrol) == True:   # here enrol key means ok
    if gpio.input(inc) == False:
        count=count+1
        if count>1000:
            count=1000
        lcdcmd(0xca)
        lcdprint(str(count))
        time.sleep(0.2)
    elif gpio.input(dec) == False:
        count=count-1
        if count<0:
            count=0
        lcdcmd(0xca)
        lcdprint(str(count))
        time.sleep(0.2)
positionNumber=count
if f.deleteTemplate(positionNumber) == True :
        print('Template deleted!')
        lcdcmd(1)
        lcdprint("Finger Deleted");
        time.sleep(2)
begin()
lcdcmd(0x01)
lcdprint("FingerPrint ")
lcdcmd(0xc0)
lcdprint("Interfacing ")
time.sleep(3)
lcdcmd(0x01)
lcdprint("Edkits Electroni")
lcdcmd(0xc0)
lcdprint("cs Welcomes You ")
time.sleep(3)
flag=0
lcdclear()
while 1:
    gpio.output(led, HIGH)
    lcdcmd(1)
    lcdprint("Place Finger")
    if gpio.input(enrol) == 0:
        gpio.output(led, LOW)
    enrollFinger()
elif gpio.input(delet) == 0:
    gpio.output(led, LOW)
    while gpio.input(delet) == 0:
        time.sleep(0.1)
    deleteFinger()
else:
    searchFinger()
```

Practical 8: Raspberry Pi GPS Module Interfacing

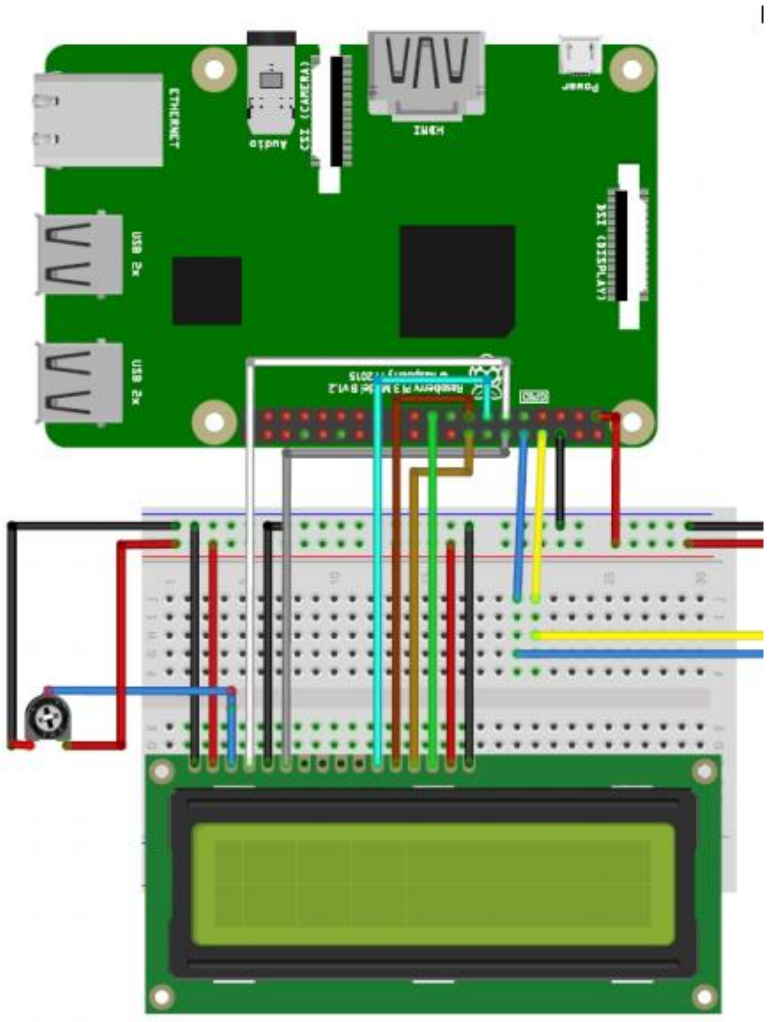
Hardware Requirements:

1. GPS module
2. USB to TTL converter
3. Connecting wires

Software Requirements:

1. Python
2. Raspbian OS
3. gpsd module via github

Circuit Diagram:



Code:

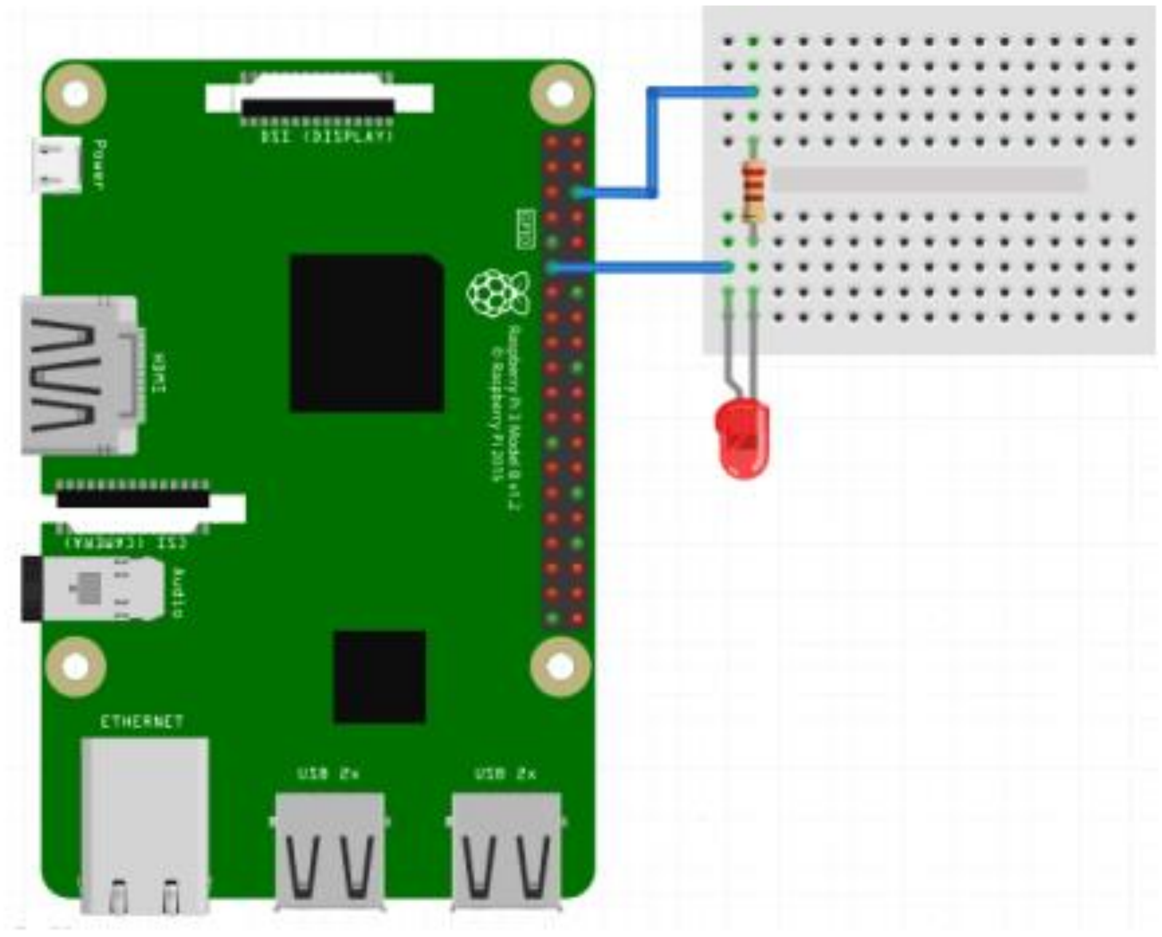
```
import serial              #import serial pacakge

from time import sleep

import webbrowser          #import package for opening link in browser

import sys                 #import system package

def GPS_Info():

    global NMEA_buff

    global lat_in_degrees

    global long_in_degrees

    nmea_time = []

    nmea_latitude = []

    nmea_longitude = []

    nmea_time = NMEA_buff[0]              #extract time from GPGGA string

    nmea_latitude = NMEA_buff[1]          #extract latitude from GPGGA string

    nmea_longitude = NMEA_buff[3]         #extract longitude from GPGGA string

   print("NMEA Time: ", nmea_time,'\n')

    print ("NMEA Latitude:", nmea_latitude,"NMEA Longitude:",
nmea_longitude,'\n')

   lat = float(nmea_latitude)             #convert string into float for calculation

    longi = float(nmea_longitude)         #convertr string into float for calculation

lat_in_degrees = convert_to_degrees(lat)    #get latitude in degree decimal format

    long_in_degrees = convert_to_degrees(longi) #get longitude in degree decimal
format
```

```
#convert raw NMEA string into degree decimal format

def convert_to_degrees(raw_value):

    decimal_value = raw_value/100.00

    degrees = int(decimal_value)

    mm_mmmm = (decimal_value - int(decimal_value))/0.6

    position = degrees + mm_mmmm

    position = "%.4f" %(position)

    return position

gpgga_info = "$GNGGA,"

ser = serial.Serial ("/dev/ttyUSB0")    #Open port with baud rate

GPGGA_buffer = 0

NMEA_buff = 0

lat_in_degrees = 0

long_in_degrees = 0

try:

    while True:

        received_data = (str)(ser.readline())                   #read NMEA string received

        GPGGA_data_available = received_data.find(gpgga_info)   #check for
NMEA GPGGA string

        if (GPGGA_data_available>0):

            GPGGA_buffer = received_data.split("$GNGGA,",1)[1]  #store data
coming after "$GPGGA," string
```

```
            NMEA_buff = (GPGGA_buffer.split(','))               #store comma separated
data in buffer

            GPS_Info()                                    #get time, latitude, longitude

            print("lat in degrees:", lat_in_degrees," long in degree: ", long_in_degrees,
'\n')

            map_link = 'http://maps.google.com/?q=' + lat_in_degrees + ',' +
long_in_degrees    #create link to plot location on Google map

            print("<<<<<<<<press ctrl+c to plot location on google maps>>>>>>\n")
#press ctrl+c to plot on map and exit

            print("------------------------------------------------------------\n")

except KeyboardInterrupt:

    webbrowser.open(map_link)                                #open current position
information in google map

    sys.exit(0)
```

Practical 9: IoT based Web Controlled Home Automation using Raspberry Pi

Circuit Diagram:



Code:

#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.gzde:

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

WebIOPidoes 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

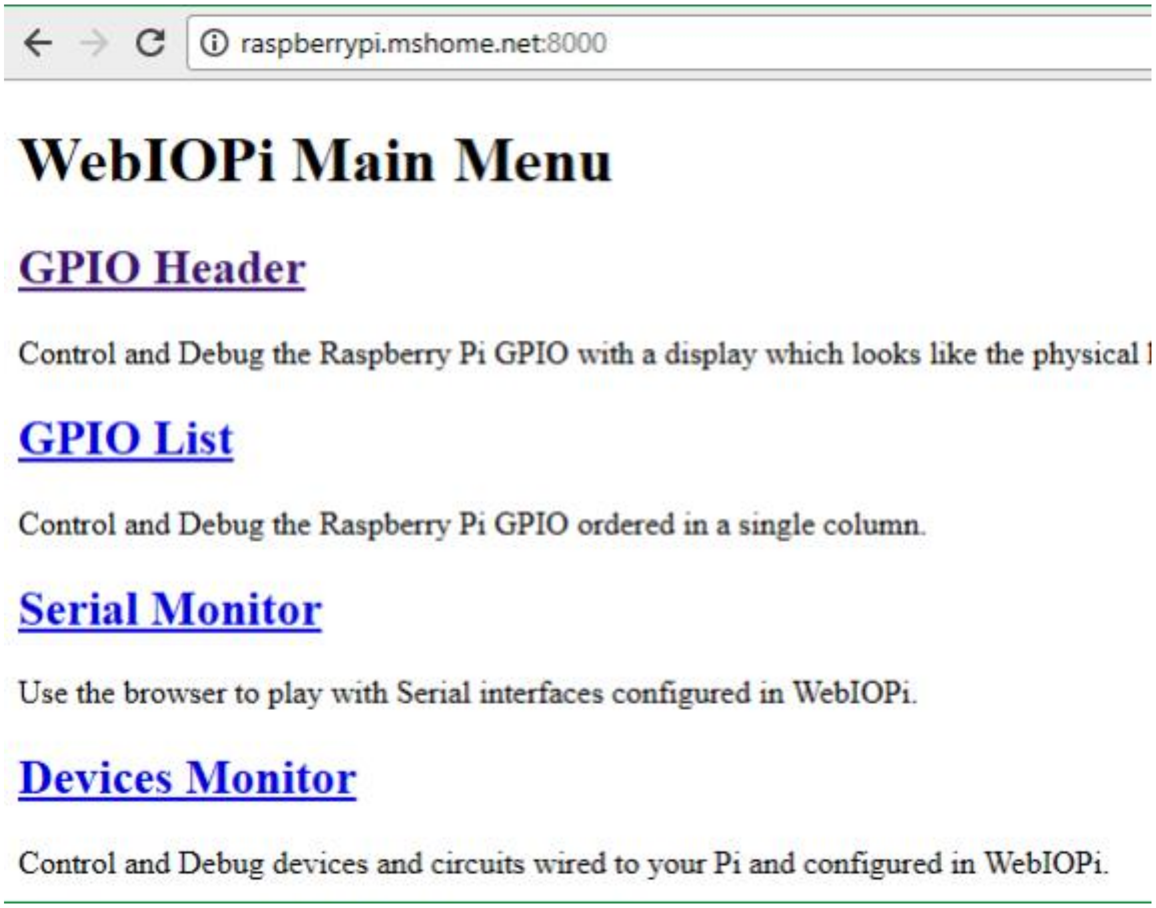
#Password is raspberry

#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.



raspberrypi.mshome.net:8000

WebIOPi Main Menu

GPIO Header

Control and Debug the Raspberry Pi GPIO with a display which looks like the physical

GPIO List

Control and Debug the Raspberry Pi GPIO ordered in a single column.

Serial Monitor

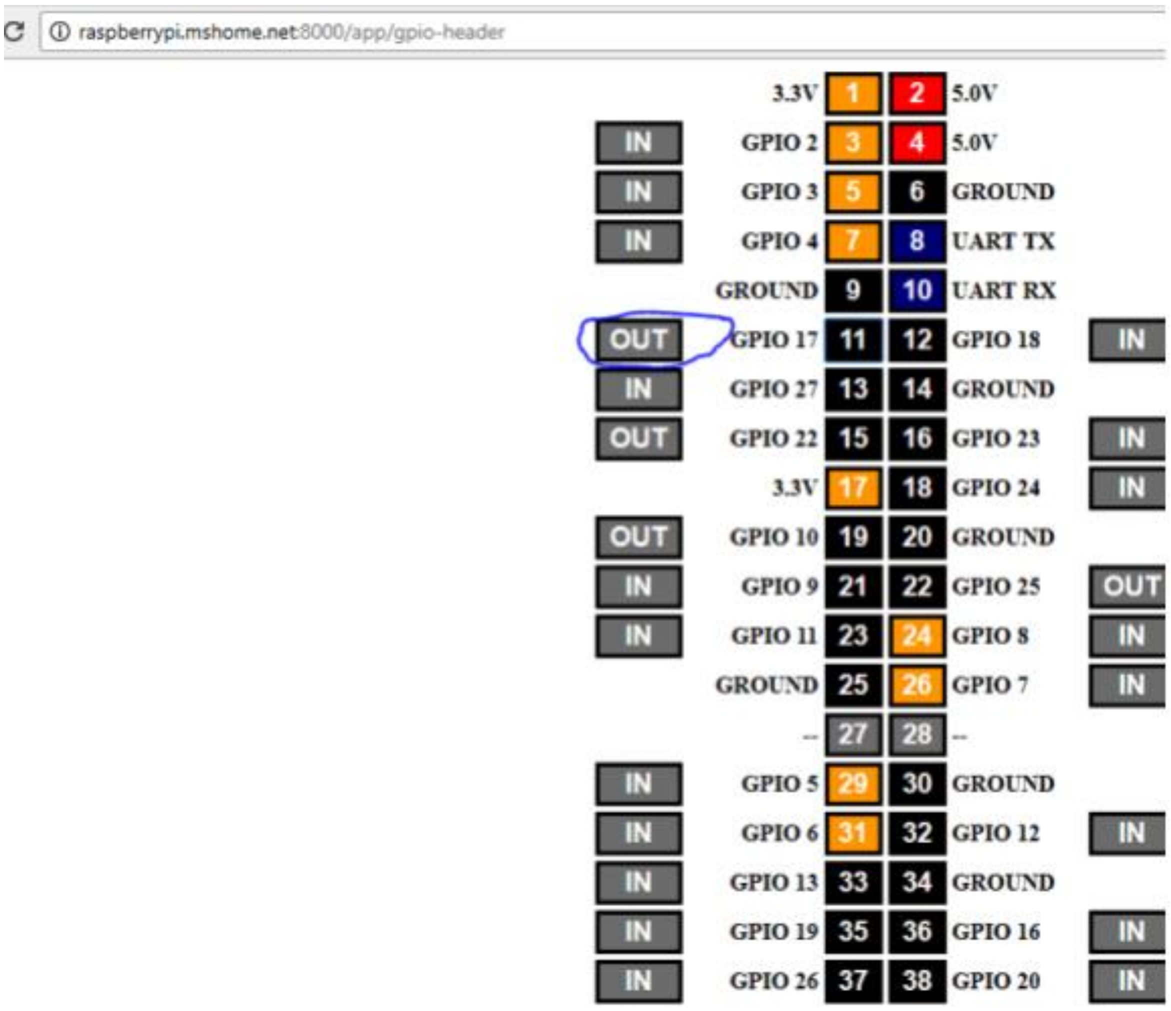
Use the browser to play with Serial interfaces configured in WebIOPi.

Devices Monitor

Control and Debug devices and circuits wired to your Pi and configured in WebIOPi.

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

output



raspberrypi.mshome.net:8000/app/gpio-header

	3.3V	1	2	5.0V	
IN	GPIO 2	3	4	5.0V	
IN	GPIO 3	5	6	GROUND	
IN	GPIO 4	7	8	UART TX	
	GROUND	9	10	UART RX	
OUT	GPIO 17	11	12	GPIO 18	IN
IN	GPIO 27	13	14	GROUND	
OUT	GPIO 22	15	16	GPIO 23	IN
	3.3V	17	18	GPIO 24	IN
OUT	GPIO 10	19	20	GROUND	
IN	GPIO 9	21	22	GPIO 25	OUT
IN	GPIO 11	23	24	GPIO 8	IN
	GROUND	25	26	GPIO 7	IN
	--	27	28	--	
IN	GPIO 5	29	30	GROUND	
IN	GPIO 6	31	32	GPIO 12	IN
IN	GPIO 13	33	34	GROUND	
IN	GPIO 19	35	36	GPIO 16	IN
IN	GPIO 26	37	38	GPIO 20	IN

.

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.

Practical 10: Visitor Monitoring with Raspberry Pi and Pi Camera

Hardware Requirements:

1.Raspberry Pi

2. Pi camera

3. 16x2 LCD

4. DC Motor

5. IC L293D

6. Buzzer

7. LED

8. Bread Board

9. Resistor (1k,10k)

10. Capacitor (100nF)

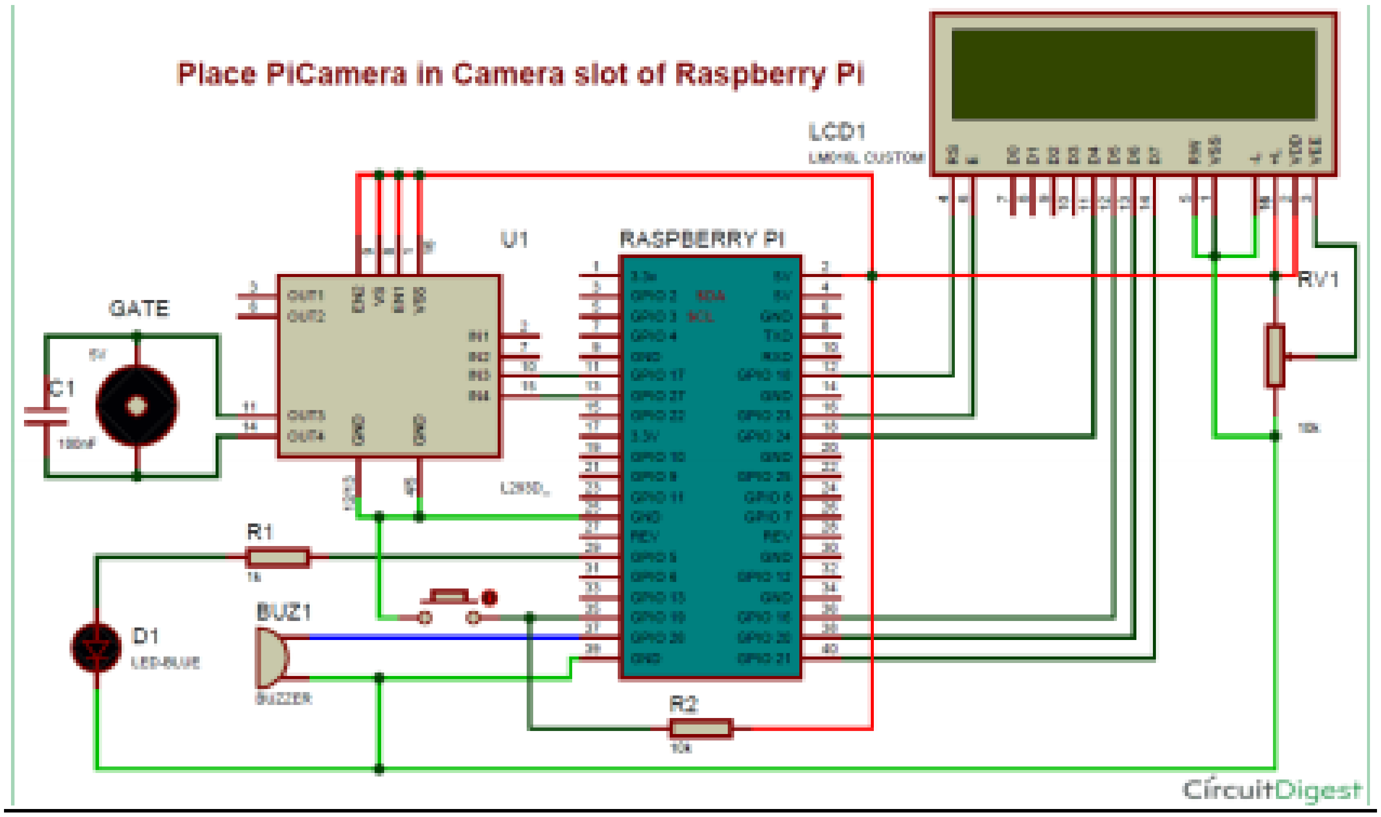
11. Push Button

12. Connecting wires

13. 10k Pot

14.Power Supply

Circuit Diagram:



Code:

```
import RPi.GPIO as gpio

import picamera

import time

m11=17

m12=27

led=5

buz=26

button=19

HIGH=1
```

```
LOW=0
gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(led, gpio.OUT)
gpio.setup(buz, gpio.OUT)
gpio.setup(m11, gpio.OUT)
gpio.setup(m12, gpio.OUT)
gpio.setup(button, gpio.IN)
gpio.output(led, 0)
gpio.output(buz, 0)
gpio.output(m11, 0)
gpio.output(m12, 0)
data=""
def capture_image():
    print("Please Wait...")
    data=time.strftime("%d_%b_%Y\%H:%M:%S")
    camera.start_preview()
    time.sleep(5)
    print(data)
    camera.capture('/home/pi/Desktop/Visitors/%s.jpg'%data)
    camera.stop_preview()
    print("Image Captured Successfully")
    time.sleep(2)
```

```
def gate():
    print(" Welcome ")
    gpio.output(m11,1)
    gpio.output(m12,0)
    time.sleep(1.5)
    gpio.output(m11,0)
    gpio.output(m12,0)
    time.sleep(3)
    gpio.output(m11,0)
    gpio.output(m12,1)
    time.sleep(1.5)
    gpio.output(m11,0)
    gpio.output(m12,0)
    print(" Thankyou ")
    time.sleep(2)
print("Visitor Monitoring")
print("   Using RPI  ")
time.sleep(3)
camera = picamera.PiCamera()
camera.rotation=180
camera.awb_mode='auto'
camera.brightness=55
```

```
time.sleep(2)
while 1:
  print("Please Press Button")
  print (" to open the gate ")
  gpio.output(led, 1)
  if gpio.input(button)==0:
    gpio.output(buz, 1)
    gpio.output(led, 0)
    time.sleep(0.5)
    gpio.output(buz, 0)
    capture_image()
    gate()
    time.sleep(0.5)



camera.rotation=180
camera.rotation=180
```

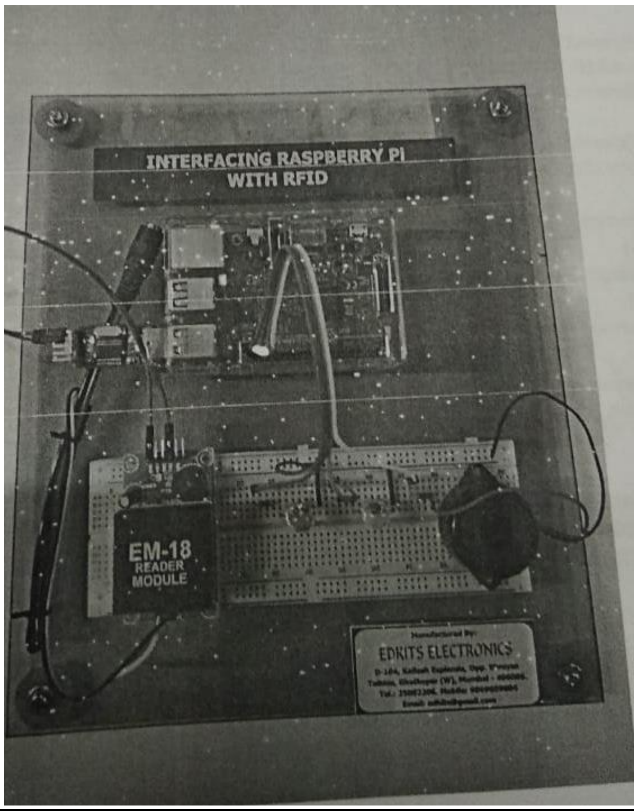
Practical 11:Interfacing Raspberry Pi with RFID.

Hardwa• Raspberry Pi 3 Model B

1. EM-18 RFID Reader Module
2. RS232 – to – USB Adapter (as my RFID Reader has only RS232 Output)
3. Few RFID Cards or RFID Tags
4. Power Supply for RFID Reader (my RFID Reader has a 5V Regulator)
5. 5V Supply for Raspberry Pi and RS232-USB Adapter
6. Connecting Wires
7. 680Ω Resistor (1/4 Watt)
8. 1.5KΩ Resistor (1/4 Watt)re Requirements:

Software Requirements:

Circuit Diagram:



To Get ID Code:

```
import serial


def read_rfid ():
    ser = serial.Serial("/dev/ttyUSB0")
    ser.baudrate = 9600
    data = ser.read(12)
    ser.close ()
    return data


  id = read_rfid ()
  print id
```

Code:

```
import RPi.GPIO as GPIO
import time
import serial
GPIO.setmode(GPIO.BOARD)
greenLED = 37
redLED = 35
buzzer = 33
```

```
GPIO.setup(greenLED, GPIO.OUT)
GPIO.setup(redLED, GPIO.OUT)
GPIO.setup(buzzer, GPIO.OUT)

GPIO.setup(greenLED, False)
GPIO.setup(redLED, False)

GPIO.output(buzzer, True)
time.sleep(0.1)
GPIO.output(buzzer, False)
time.sleep(0.1)
GPIO.output(buzzer, True)
time.sleep(0.1)
GPIO.output(buzzer, False)
time.sleep(0.1)

def read_rfid ():
    ser = serial.Serial("/dev/ttyUSB0")
    ser.baudrate = 9600
    data = ser.read(12)
    ser.close()
    return data
```

```
try:

    while True:
     id = read_rfid ()
     print (id)
 if id=="400037052656":
         print("Access Granted")
         GPIO.output(greenLED, True)
         GPIO.output(redLED, False)
         GPIO.output(buzzer, False)
         time.sleep(2)
     else:
         print("Access Denied")
         GPIO.output(greenLED, False)
         GPIO.output(redLED, True)
         GPIO.output(buzzer, True)
         time.sleep(2)
         GPIO.output(greenLED, False)
         GPIO.output(redLED, False)
         GPIO.output(buzzer, False)
finally:
    GPIO.cleanup()
```

Practical 12 : Building Google Assistant with Raspberry Pi.

Hardware requirements:

1. Raspberry pi
2. USB microphone
3. Speaker

Software Requiremnets:

1. Raspbian OS
2. Google API
3. Python

Code:

Testing your Audio for Google Assistant

1. Before we get into all the hard work of setting up our Google Assistant and setting up the required API .we will first test to ensure our audio is working. At this stage, you must have your USB microphone and speakers attached to your Raspberry Pi. Once you are sure both are connected to the Raspberry Pi, we can test to make sure that the speakers are working correctly by running the following command. speaker-test -t wav You should hear sound from your speakers. This sound will be a person speaking. If you do not hear anything coming from your speaker's double check they are plugged in correctly and are turned up.

2. Now, let's test our microphone by making a recording, to do this we will run the following command on your Raspberry Pi. This command will make a short 5-second recording. arecord --format=S16_LE --duration=5 --rate=16000 --file-type=raw out.raw If you receive an error when running this command make sure that you have your microphone plugged in, this command will only succeed if it can successfully listen to your microphone.

3. With our recording done we can now run the following command to read in our raw output file and play it back to our speakers. Doing this will allow you to test

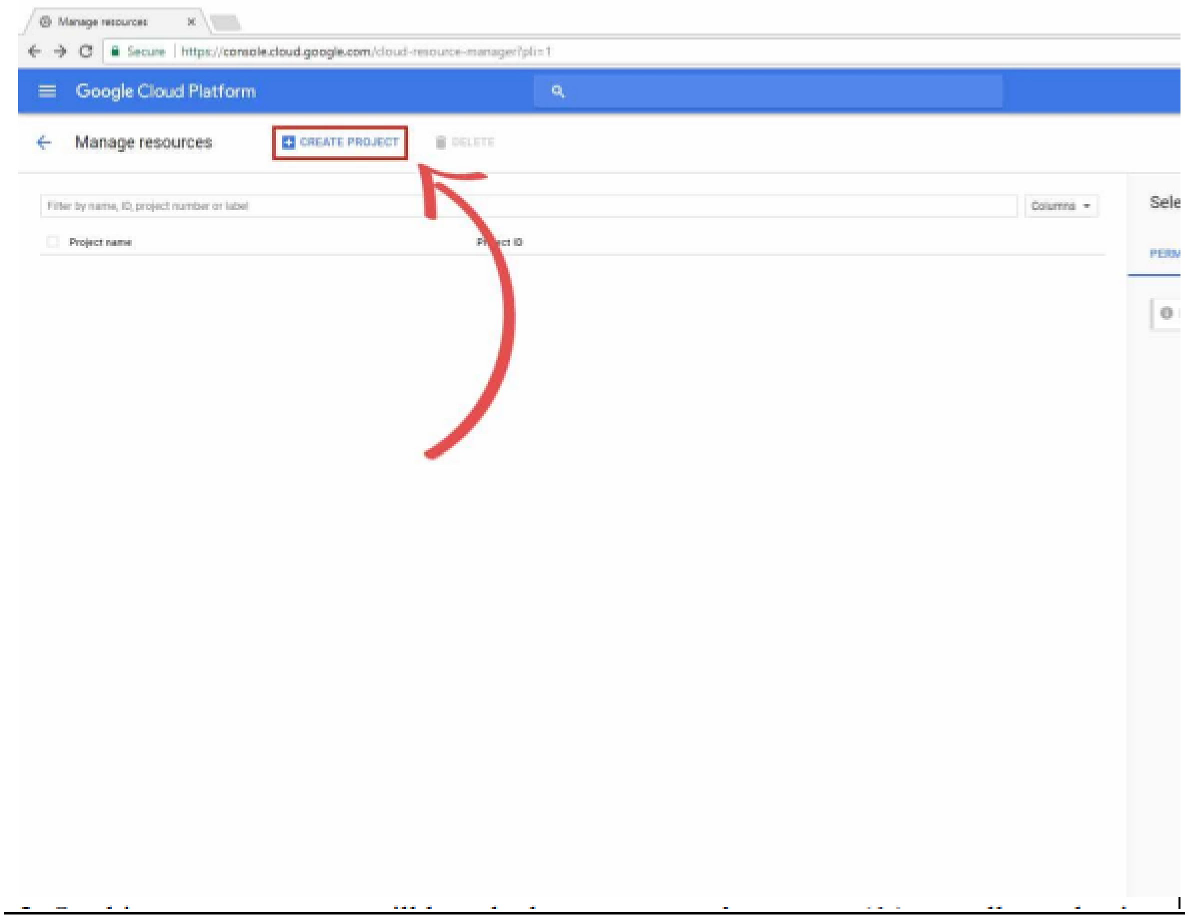
the playback volume and also listen to the recording volume. Doing this is a crucial task as you don't want your Raspberry Pi picking up every little noise but you also don't want it being able to barely hear you when you say "Ok Google". aplay --format=S16_LE --rate=16000 out.raw

4. If you find the playback volume or recording volume is either too high or too low, then you can run the following command to launch the mixer. This command will allow you to tweak the volumes for certain devices. Alsamixer

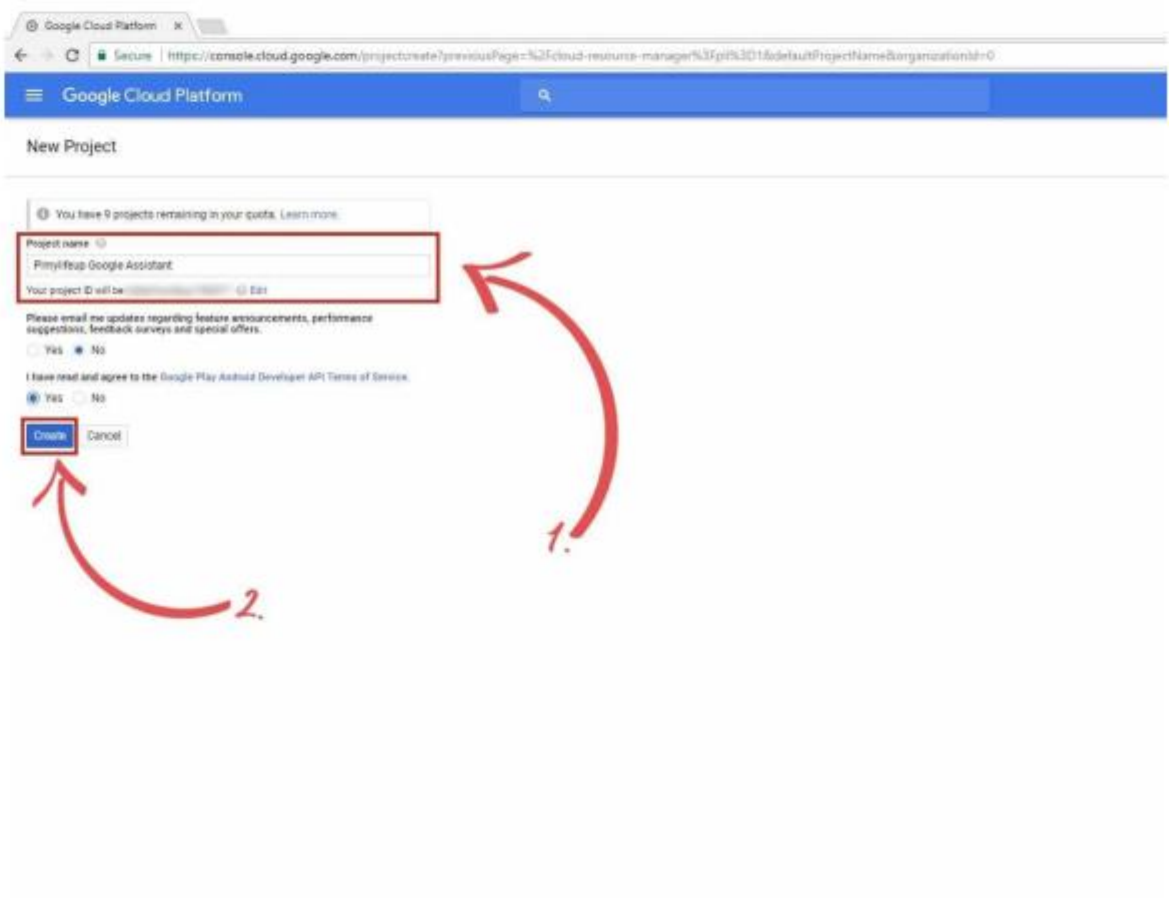
Once you have confirmed that your microphone and speakers are working correctly, you can move onto setting up your very own Raspberry Pi Google Assistant. Registering for the Google API

1. Before we get started with setting up the Google Assistant code on the Raspberry Pi itself, we must first register and setup oAuth access to Google's Assistant API. To do this, you will need to have a Google account already. Once you have your Google account ready, go to the following web address. https://console.cloud.google.com/project

2. Once you have logged into your account, you will be greeted with the following screen. On here you will want to click the "Create Project" link as shown in our screenshot.



3. On this next screen, you will be asked to enter a project name (1.) as well as selecting the two radial boxes. For the project name just set something relevant to what you plan on doing. For instance, we set ours to “Pimylifeup Google Assistant” For the two radial boxes, we selected ‘No‘ to wanting email updates and ‘Yes‘ to agree to their terms of service. Finally, you will need to press the “Create” button.


Google Cloud Platform
Secure
https://console.cloud.google.com/projectcreate?previousPage=%2Fcloud-resource-manager%3Fpli%3D1&defaultProjectName&organizationId=0
Google Cloud Platform
New Project

You have 9 projects remaining in your quota. Learn more

Project name: Pimylifeup Google Assistant

Your project ID will be [redacted] Edit

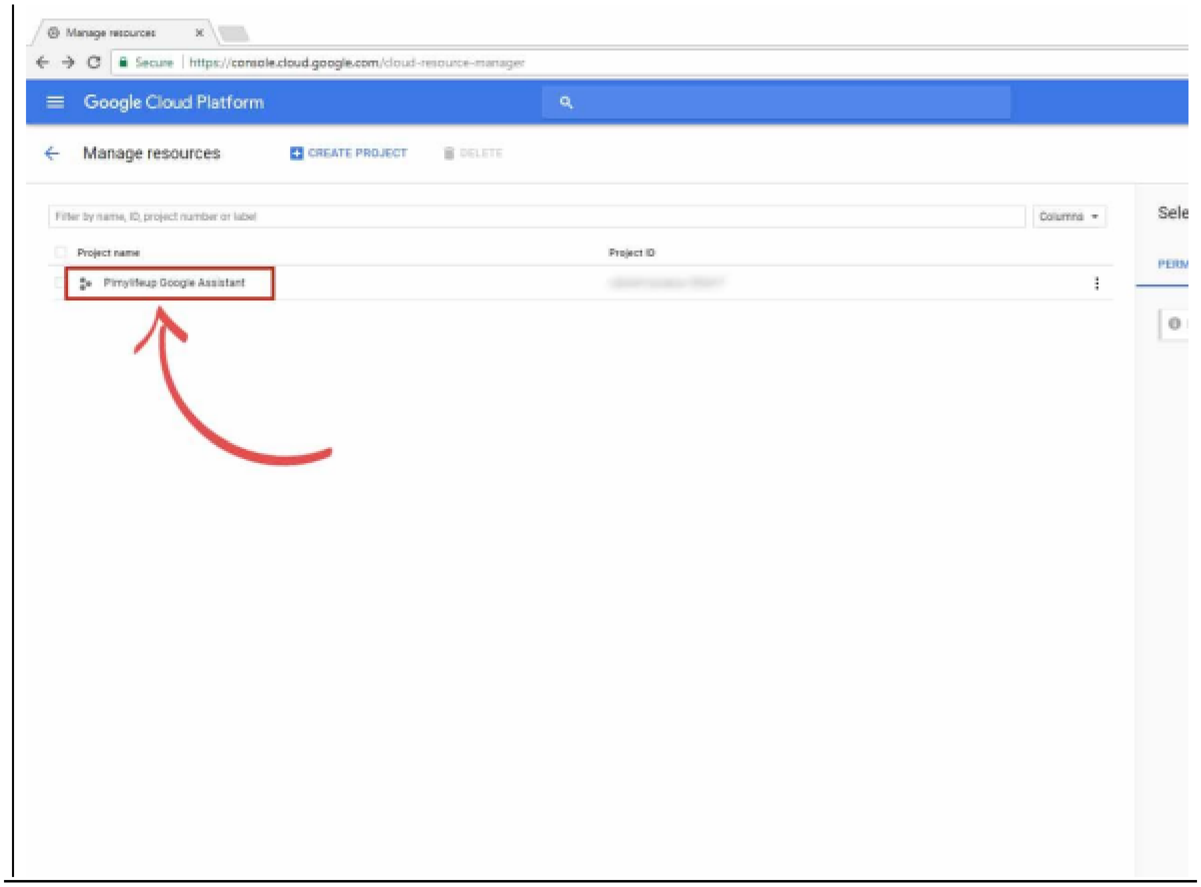
Please email me updates regarding feature announcements, performance suggestions, feedback surveys and special offers.

Yes [✓ No]

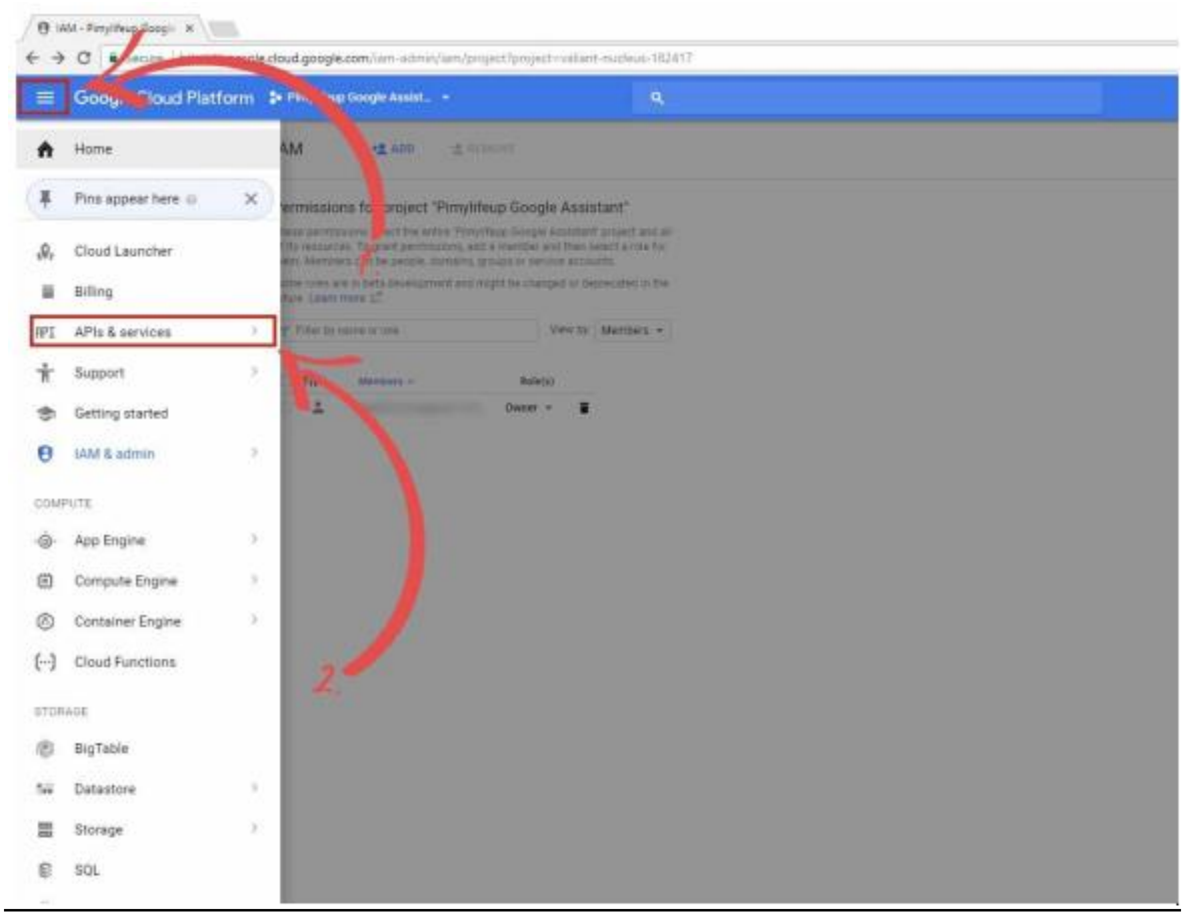
I have read and agree to the Google Play Android Developer API Terms of Service.

[✓ Yes] No

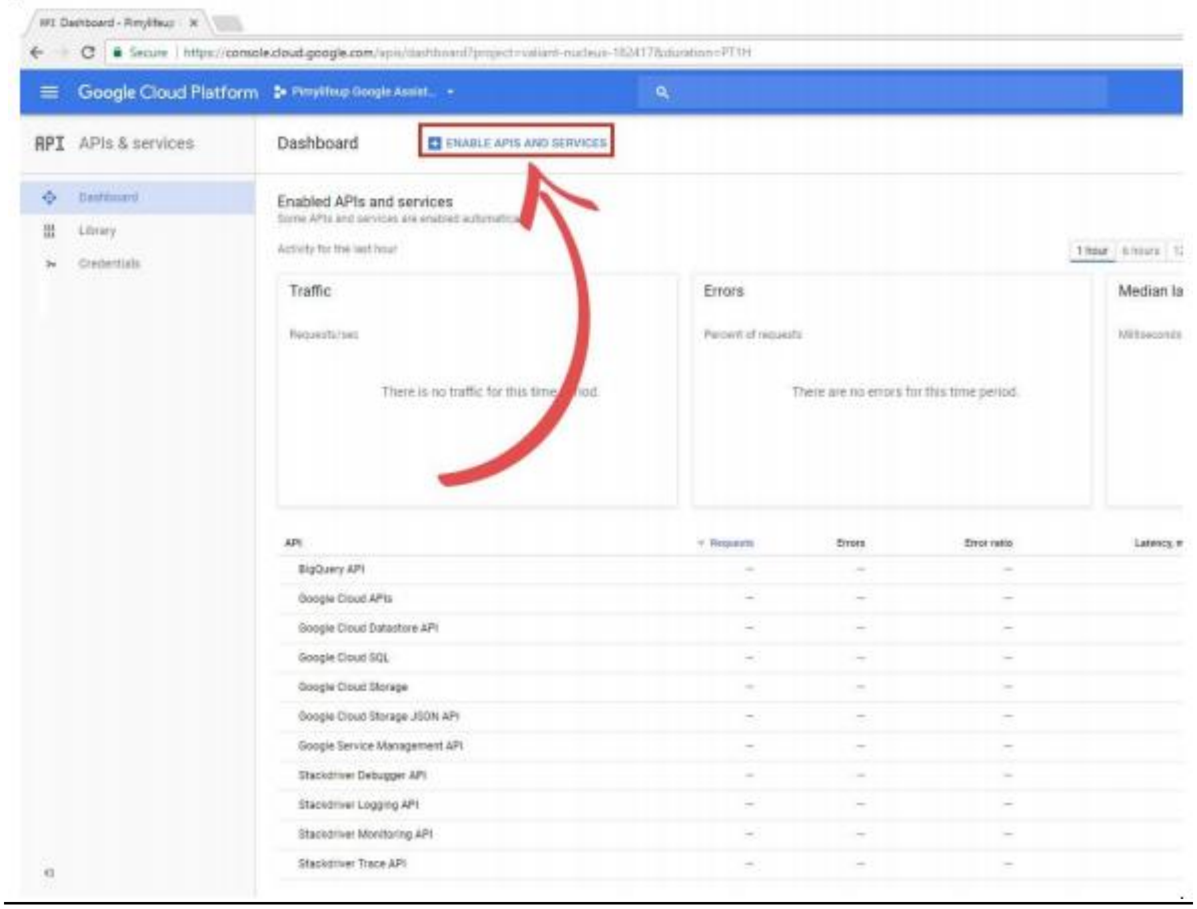
4. The project creation process can take some time, so be patient. You should receive a notification in the top right-hand corner of the screen when it's complete. If it doesn't automatically appear after some time, try refreshing the page. Once it has appeared, click the project name to select it as shown below


Manage resources
Secure | https://console.cloud.google.com/cloud-resource-manager
Google Cloud Platform
Manage resources
CREATE PROJECT
DELETE
Filter by name, ID, project number or label
Columns
Project name
Project ID
Pimylifeup Google Assistant

5. Now on this screen click the hamburger icon (1.) in the top right-hand corner to bring out the side menu. Then on the side menu, you will want to select “API’s and Service” (2.). This screen is where we will create all the authentication details that we need and also where we will enable the Google Assistant API.

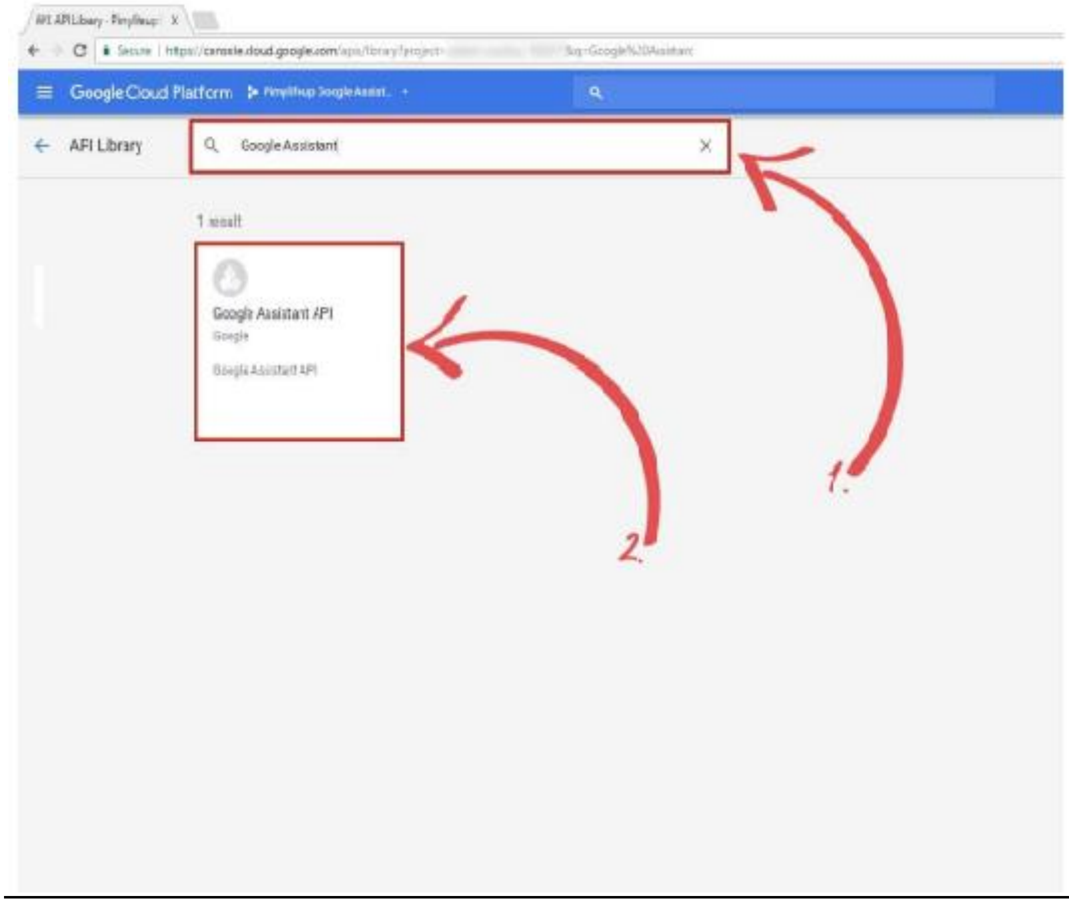


6. Now to get to the more interesting part of enabling the correct API, we need to click the “Enable APIS and SERVICES” link like as shown below:

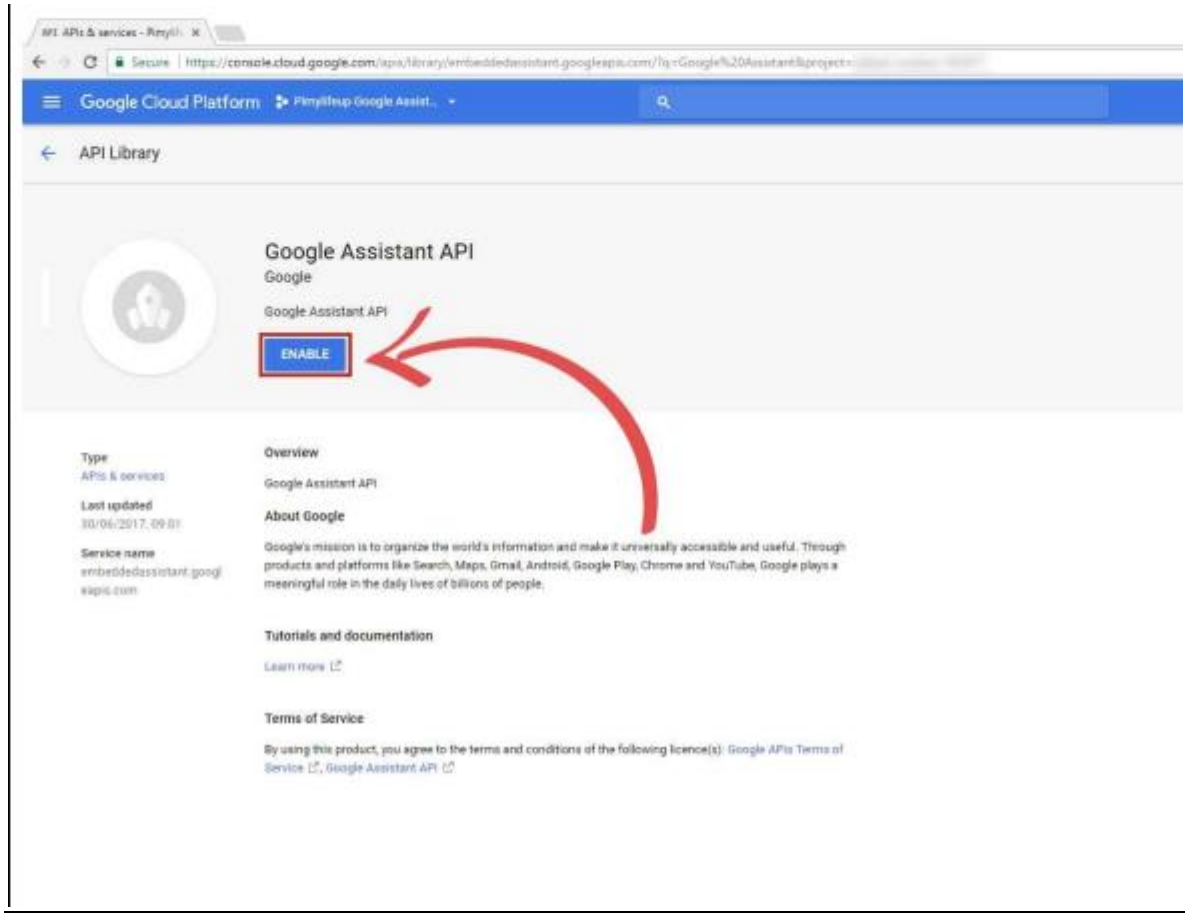


API	Requests	Errors	Error ratio
BigQuery API	–	–	–
Google Cloud APIs	–	–	–
Google Cloud Datastore API	–	–	–
Google Cloud SQL	–	–	–
Google Cloud Storage	–	–	–
Google Cloud Storage JSON API	–	–	–
Google Service Management API	–	–	–
Stackdriver Debugger API	–	–	–
Stackdriver Logging API	–	–	–
Stackdriver Monitoring API	–	–	–
Stackdriver Trace API	–	–	–

7. Finally, let’s search up the API that we are after. To do this type in “Google Assistant” in the search box. (1.) Once the results have shown up, you need to click the box that has “Google Assistant API” (2.) written on it, as this will take us to the screen to activate it.



8. On this screen, you need to click the “Enable” button as shown below.



Google Assistant API

Google

Google Assistant API

Type

APIs & services

Last updated

30/06/2017, 09:01

Service name

embeddedassistant.googleapis.com

Overview

Google Assistant API

About Google

Google's mission is to organize the world's information and make it universally accessible and useful. Through products and platforms like Search, Maps, Gmail, Android, Google Play, Chrome and YouTube, Google plays a meaningful role in the daily lives of billions of people.

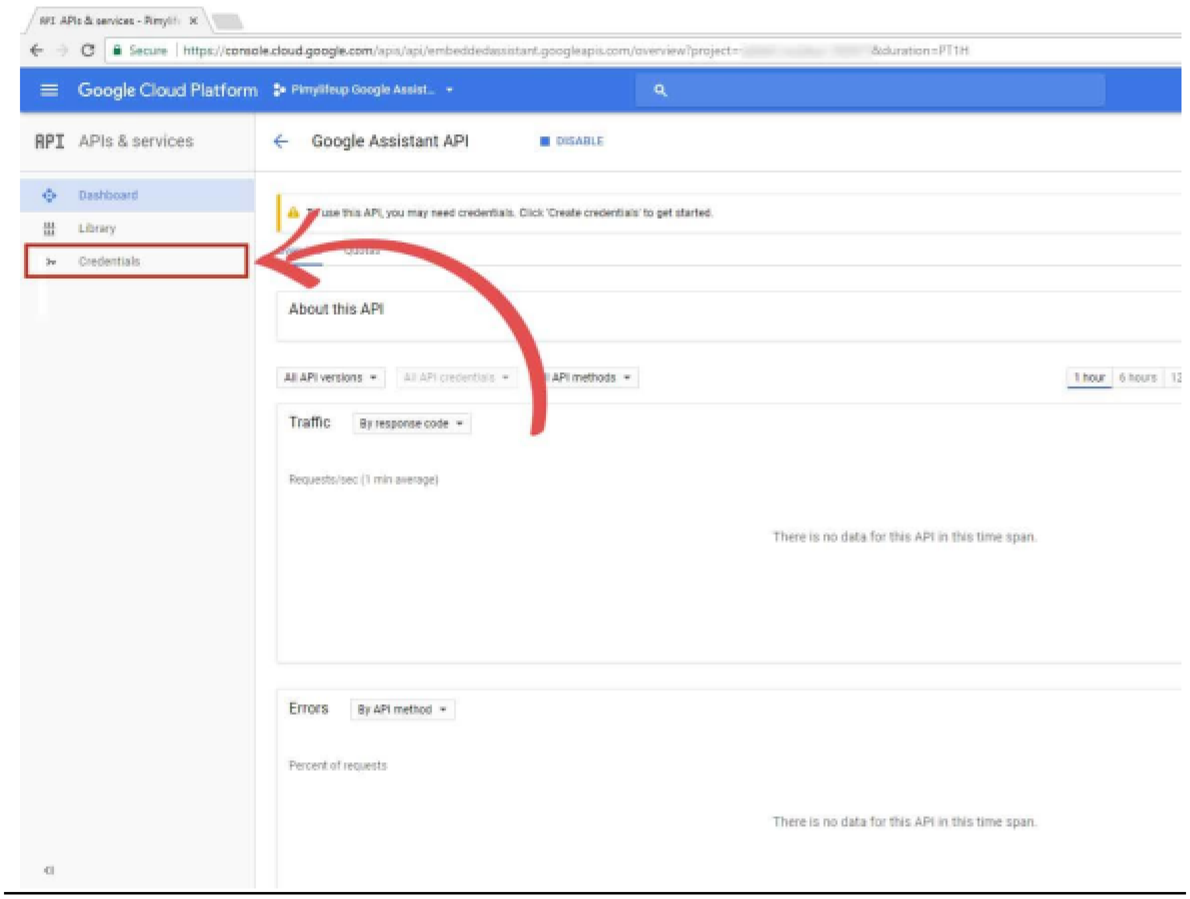
Tutorials and documentation

Learn more

Terms of Service

By using this product, you agree to the terms and conditions of the following licence(s): Google APIs Terms of Service, Google Assistant API

9. You should now be taken back to the “APIs & services” page, here you want to click “Credentials” in the sidebar.

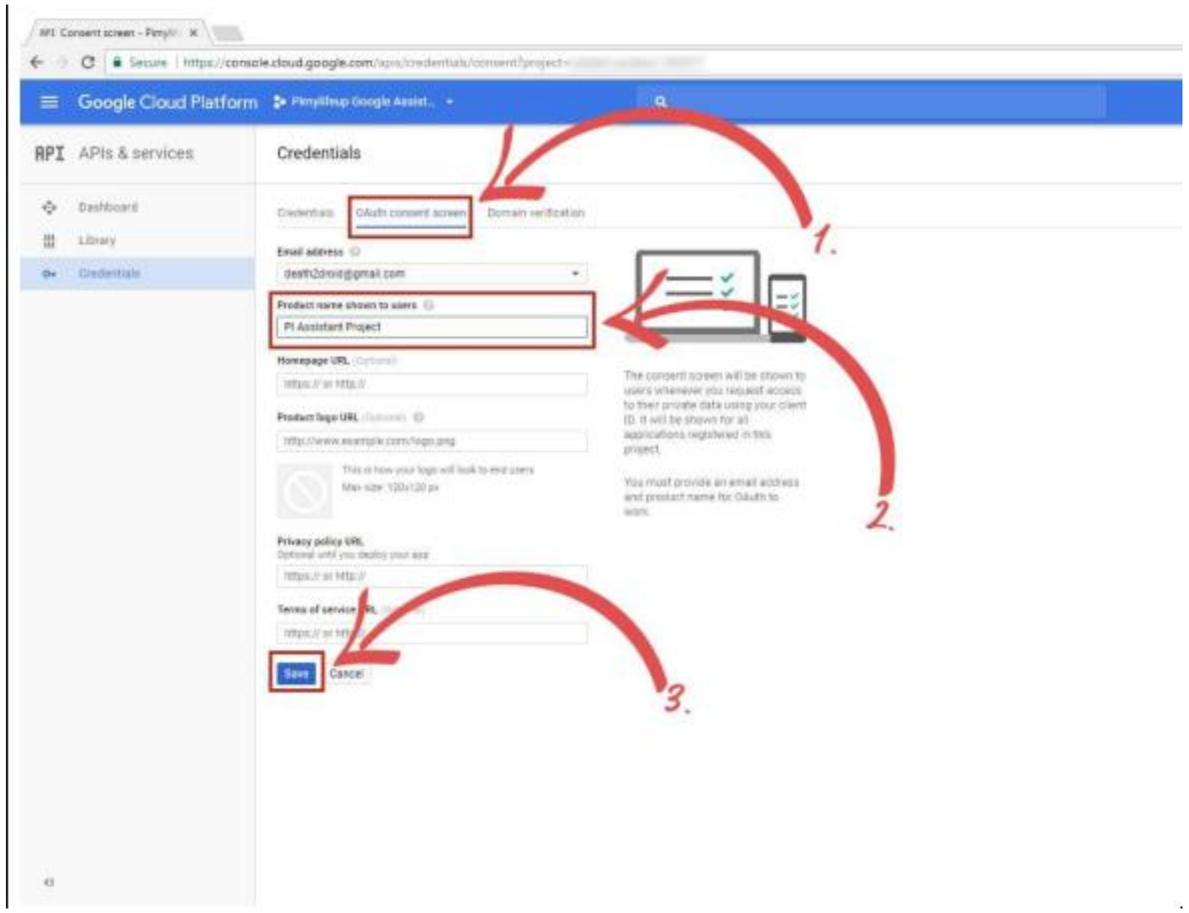


To use this API, you may need credentials. Click 'Create credentials' to get started.

There is no data for this API in this time span.

There is no data for this API in this time span.

10. Now that we are on the "Credentials" screen we need to switch to the "OAuth consent screen" tab (1.) Afterward, you will need to enter a name in the "Project name shown to users" field (2.), for our tutorial we named this "Pi Assistant Project". If you use your name, make sure you don't use the word Google in it as it will automatically refuse you. Once you have entered the Product name of your choice, you can click the "Save" button (3.)



Email address
death2droid@gmail.com

Product name shown to users
PI Assistant Project

Homepage URL (Optional)
https:// or http://

Product logo URL (Optional)
http://www.example.com/logo.png

This is how your logo will look to end users
Max size: 120x120 px

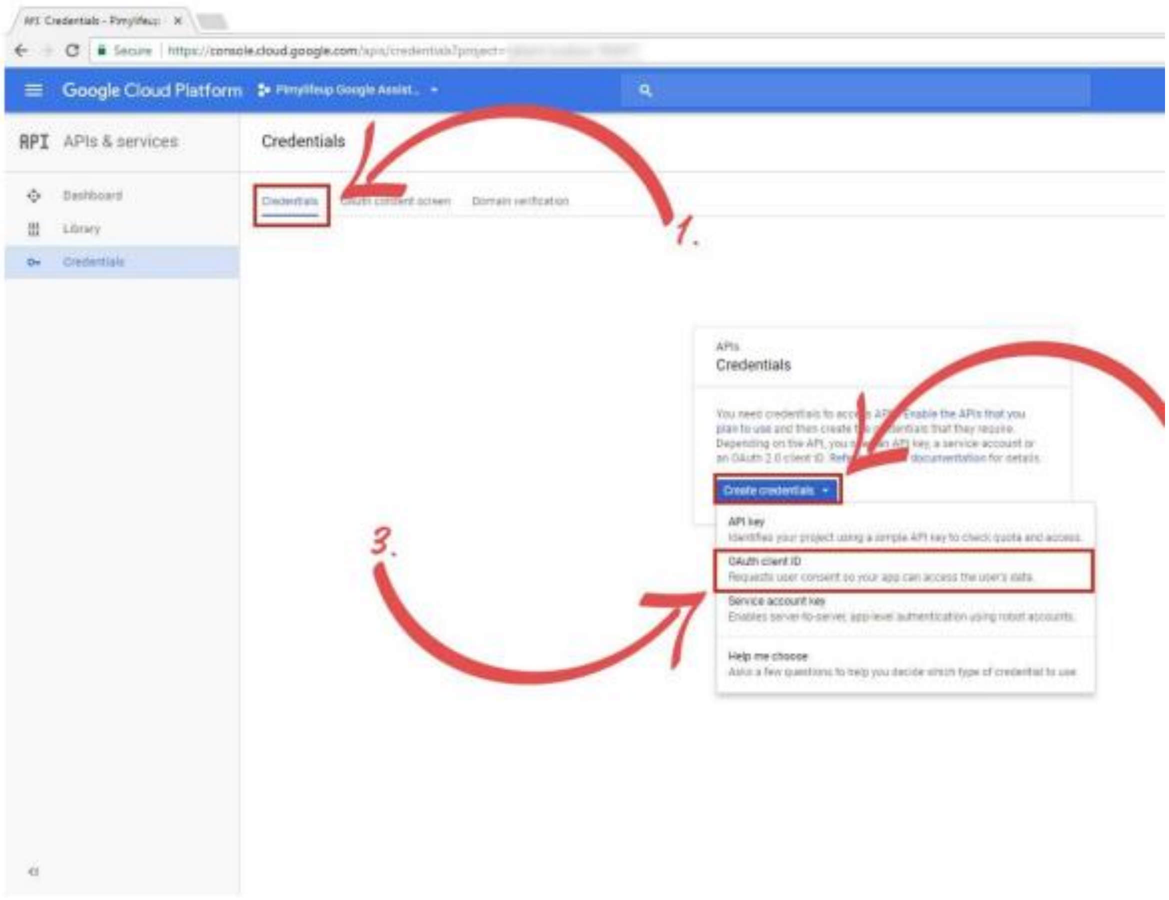
Privacy policy URL
Optional until you deploy your app
https:// or http://

Terms of service URL (Optional)
https:// or http://

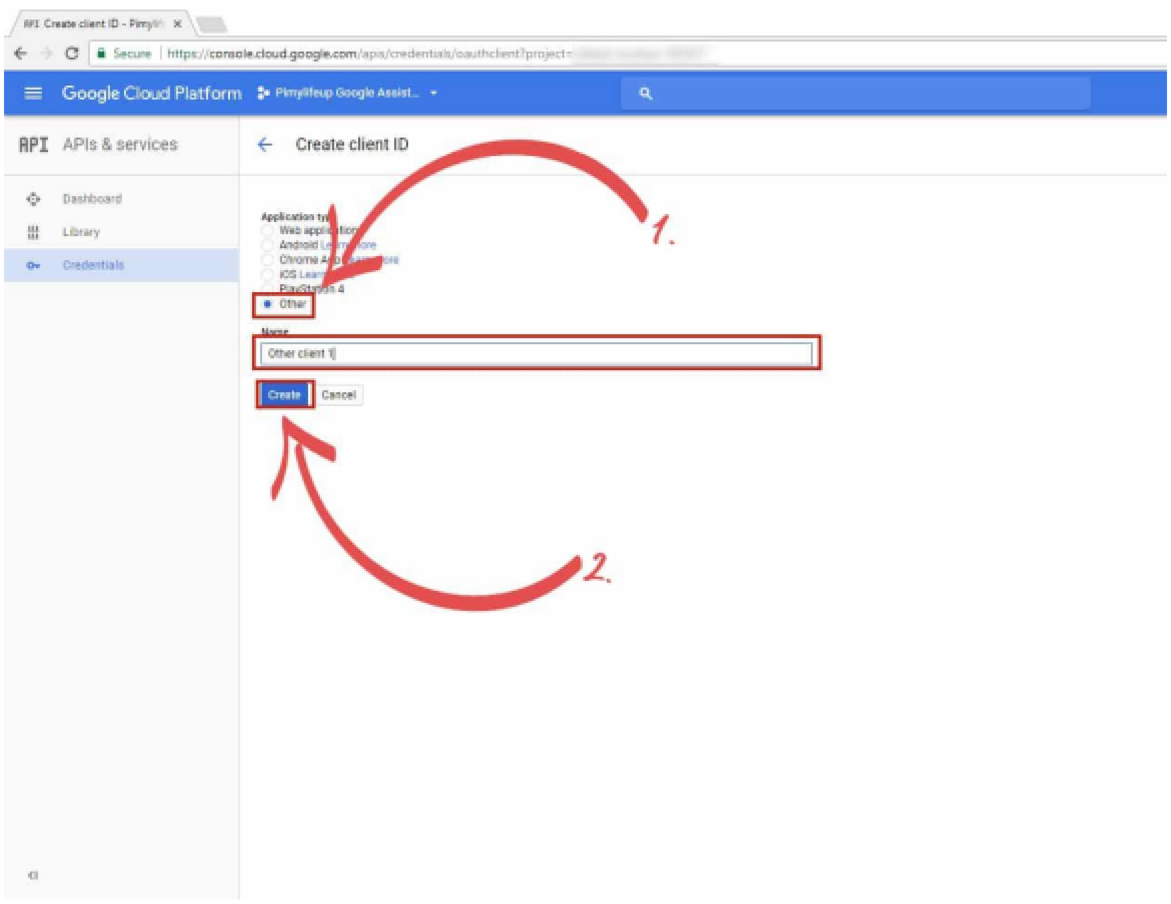
The consent screen will be shown to users whenever you request access to their private data using your client ID. It will be shown for all applications registered in this project.

You must provide an email address and product name for OAuth to work.

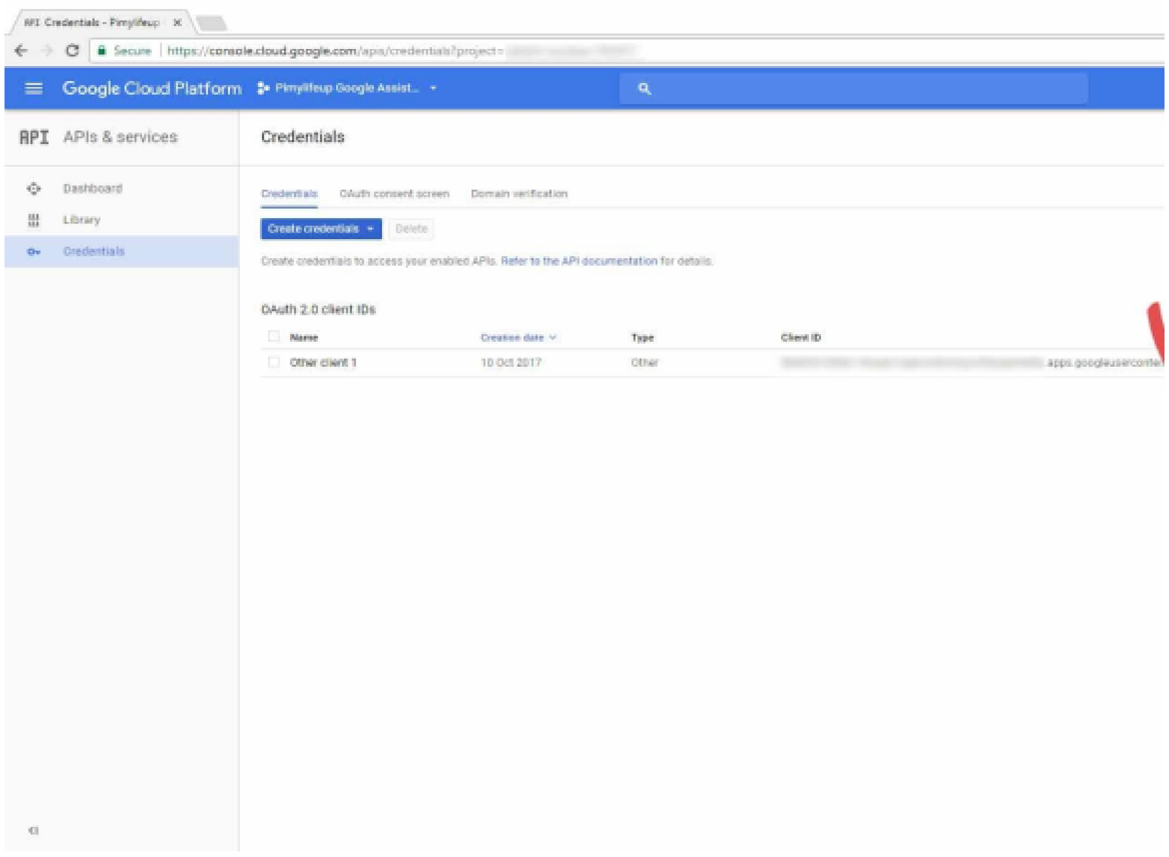
11. With the OAuth consent screen now setup we can create some credentials for it. To do this make sure you go back to the “Credentials” tab (1.). On this screen click the “Create credentials” button (2.) which will open up a dropdown menu. In this drop-down menu click “OAuth client ID” (3.).



12. Now while creating your client ID, set the "Application type" to "Other" (1.). You can also decide to change the "Name" for the client id, in our case we just left it set to the default name. Finally, press the "Create" Button (2.)



13. Now we need to download the credentials file for our newly created oAuth credential. To do this click the download button as shown in the screenshot below. Keep this somewhere safe, as we will the text inside the file to the Raspberry Pi. (Of course, unless you downloaded it directly to your Pi)



Create credentials to access your enabled APIs. Refer to the API documentation for details.

OAuth 2.0 client IDs

Name	Creation date	Type	Client ID
Other client 1	10 Oct 2017	Other	[redacted] apps.googleusercontent

14. Finally, we need to go to the URL displayed below, on here you will need to activate the following activity controls to ensure that the Google Assistant API works correctly. • Web & App Activity • Location History • Device Information • Voice & Audio Activity https://myaccount.google.com/activitycontrols Downloading and setting up Google Assistant

1. Now that we have setup your Google account with the Google Assistant API there are a few things we need to do. Before we begin, we should update the Raspberry Pi's operating system to ensure that we are running the latest available software. To do this run the following two commands on the Raspberry Pi. sudo apt-get update sudo apt-get upgrade

2. Once the Raspberry Pi has finished updating, we can then proceed with setting up everything we need for running the Google Assistant API.On your Raspberry Pi, we will be creating a file where we will store the credentials we downloaded earlier on our computer. To do this run the following two commands to create a

folder and begin writing a file to store the credentials in. mkdir ~/googleassistant nano ~/googleassistant/credentials.json

3. Within this file, we need to copy the contents of the credentials file that we downloaded to your computer. You can open the .json file in any text editor and press CTRL + A then CTRL + C to copy the contents. Now in your SSH window, right click and click “Paste”.

4. Once you have copied the contents of your credentials over to our nano session, we can then save the file by pressing Ctrl + X then Y and then finally hitting Enter.

5. Now with the credentials file now saved safely to our Raspberry Pi we will start installing some of the dependencies we rely on. Run the following command to install Python3 and the Python 3 Virtual Environment to our Raspberry Pi. sudo apt-get install python3-dev python3-venv

6. We can now enable python3 as our virtual environment variable by running the following command on our Raspberry Pi. python3 -m venv env

7. With that now enabled we can go ahead and ensure that we have installed the latest versions of pip and the setuptools. To do this, we will run the following command on the Raspberry Pi. env/bin/python -m pip install --upgrade pip setuptools --upgrade

8. To get into this new Python environment that we have set up we should run the following command in terminal. source env/bin/activate

9. Now that we have all the packages we need to install the Google Assistant Library, to do this we will run the following command to utilize pip to install the latest version of the Python package. python -m pip install --upgrade google-assistant-library Getting the Google Assistant Running

1. Now that we have set up all the prerequisites to running the Google Assistant software on our Raspberry Pi we can finally get up to running it. To do this, we must first install the Google authorization tool to our Raspberry Pi. This package will allow us to authenticate our device and give ourselves the rights to be able to

make Google Assistant queries for your Google Account. Run the following command on the Raspberry Pi to install the Python authorization tool. python -m pip install --upgrade google-auth-oauthlib[tool]

2. With the Google Authentication library now installed, we need to run it. To do this, we will be running the following command on our Raspberry Pi. This command will generate a URL you will need to go to in your web browser so be prepared. google-oauthlib-tool --client-secrets ~/googleassistant/credentials.json --scope https://www.googleapis.com/auth/assistant-sdk-prototype --save --headless

3. You will now be presented with the text “Please visit this URL to authorize this application:” followed by a very long URL. Make sure you copy this URL entirely to your web browser to open it.

4. On this screen login to your Google account, if you have multiple accounts make sure you select the one you set up your API key with. Afterward, you should be presented with a screen with the text “Please copy this code, switch to your application and paste it there” followed by a long authentication code. Copy the authentication code and paste it back into your terminal session and press enter. If the authentication was accepted you should see the following line appear on your command line: “credentials saved: /home/pi/.config/google-oauthlib-tool/credentials.json”

5. Finally, we have finished setting up everything we need to ruin the Google Assistant sample on our Raspberry Pi. All that is left to do now is to run it. We can run the software by running the following command on your Raspberry Pi. google-assistant-demo

6. Say “Ok Google” or “Hey Google“, followed by your query. The Google Assistant should give you a response. If the Assistant software doesn’t respond to your voice, then make sure that you have correctly setup your microphone and speakers by checking all cables.

Practical 13: Installing Windows 10 IoT Core on Raspberry Pi

Hardware Requirements:

1. Raspberry Pi 3.

2. 5V 2A microUSB power supply.

3. 8GB or larger Class 10 microSD card with full-size SD adapter.

4. HDMI cable.

5. Access to a PC.

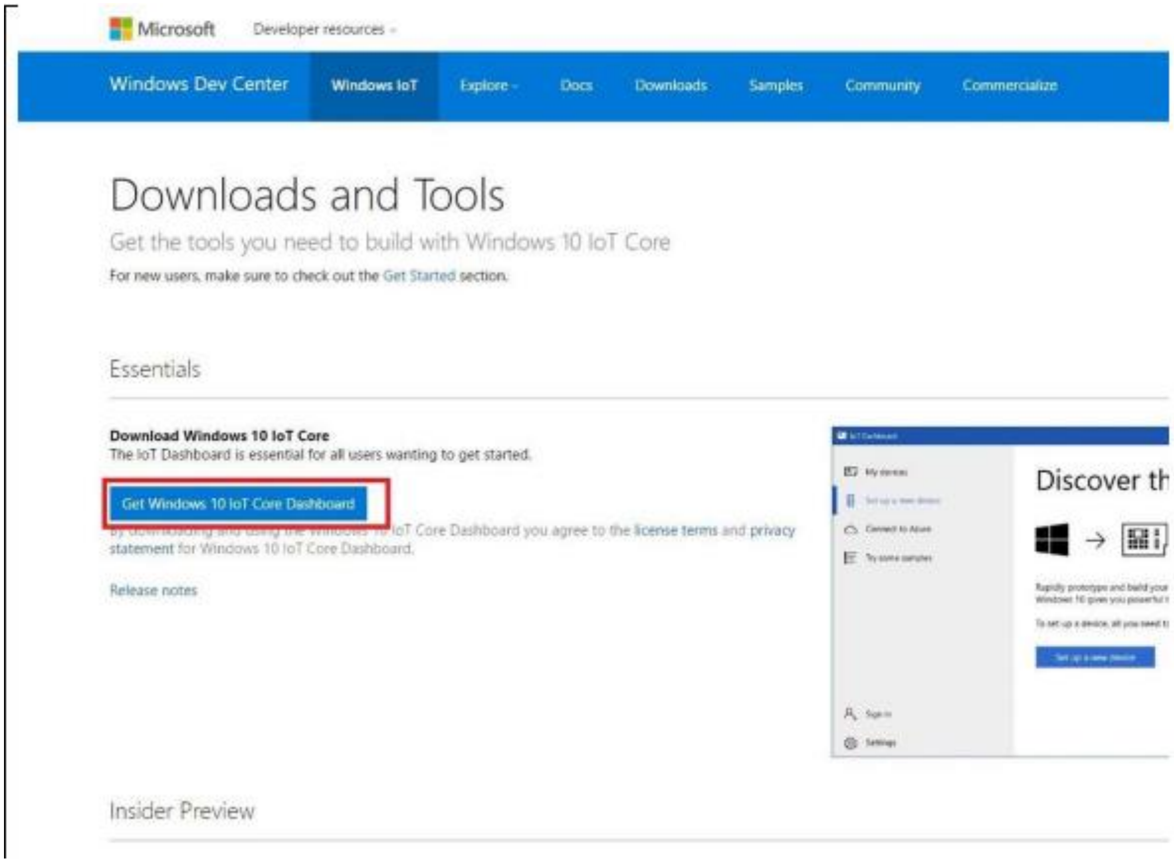
6. USB WiFi adapter (older models of Raspberry Pi) or Ethernet cable.

Software Requirements:-.

Code:

Go to the Windows 10 developer center.

Click Get Windows 10 IoT Core Dashboard to download the necessary application



Downloads and Tools

Get the tools you need to build with Windows 10 IoT Core

For new users, make sure to check out the Get Started section.

Essentials

Download Windows 10 IoT Core
The IoT Dashboard is essential for all users wanting to get started.

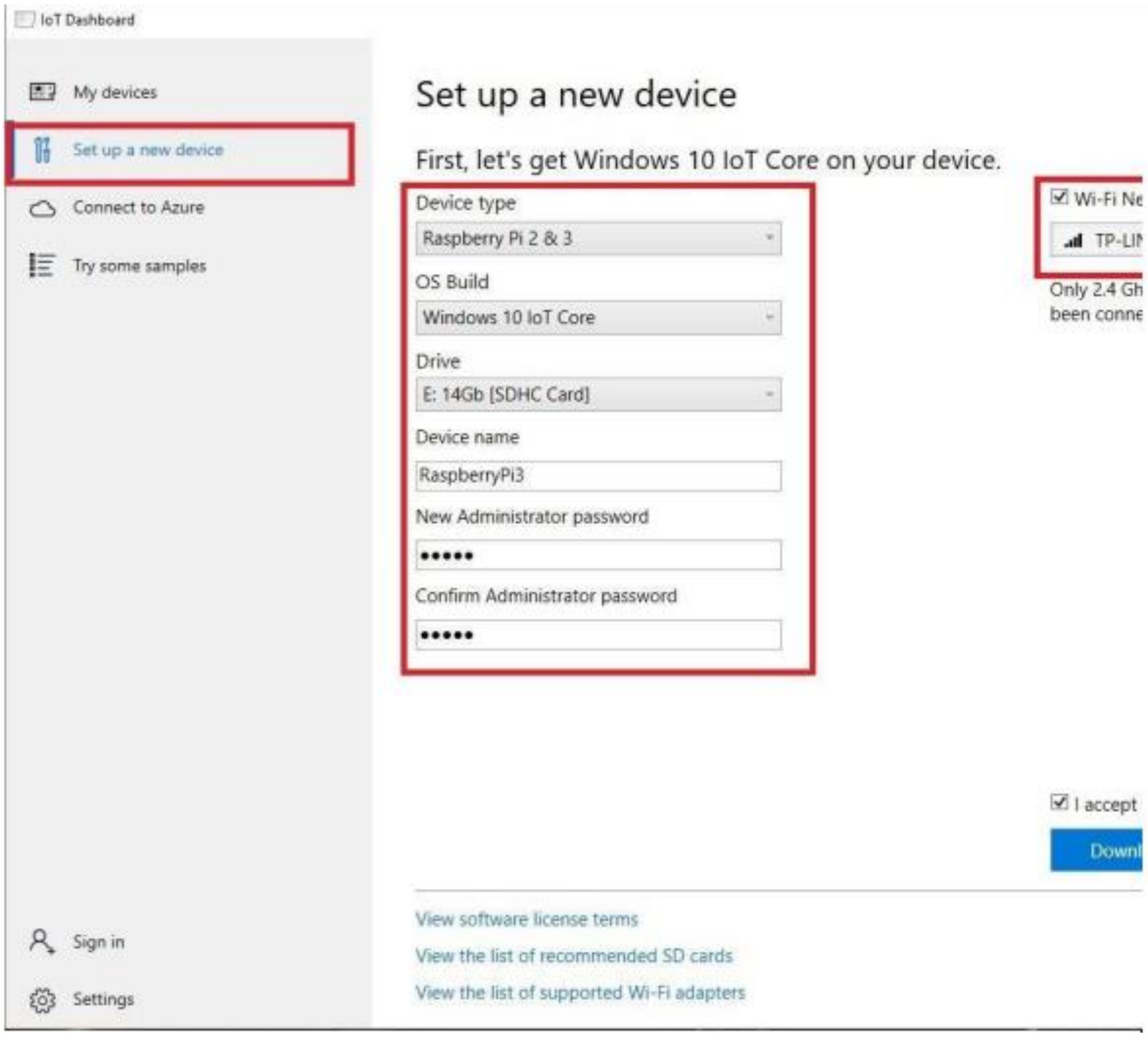
By downloading and using the Windows 10 IoT Core Dashboard you agree to the license terms and privacy statement for Windows 10 IoT Core Dashboard.

Release notes

Insider Preview

Install the application and open it.

Select set up a new device from the sidebar. Select the options as shown in the image below. Make sure you select the correct drive for your microSD card and give your device a name and admin password.


IoT Dashboard

My devices
Set up a new device
Connect to Azure
Try some samples
Sign in
Settings

Set up a new device

First, let's get Windows 10 IoT Core on your device.

Device type: Raspberry Pi 2 & 3

OS Build: Windows 10 IoT Core

Drive: E: 14Gb [SDHC Card]

Device name: RaspberryPi3

New Administrator password: •••••

Confirm Administrator password: •••••

☑ Wi-Fi Ne

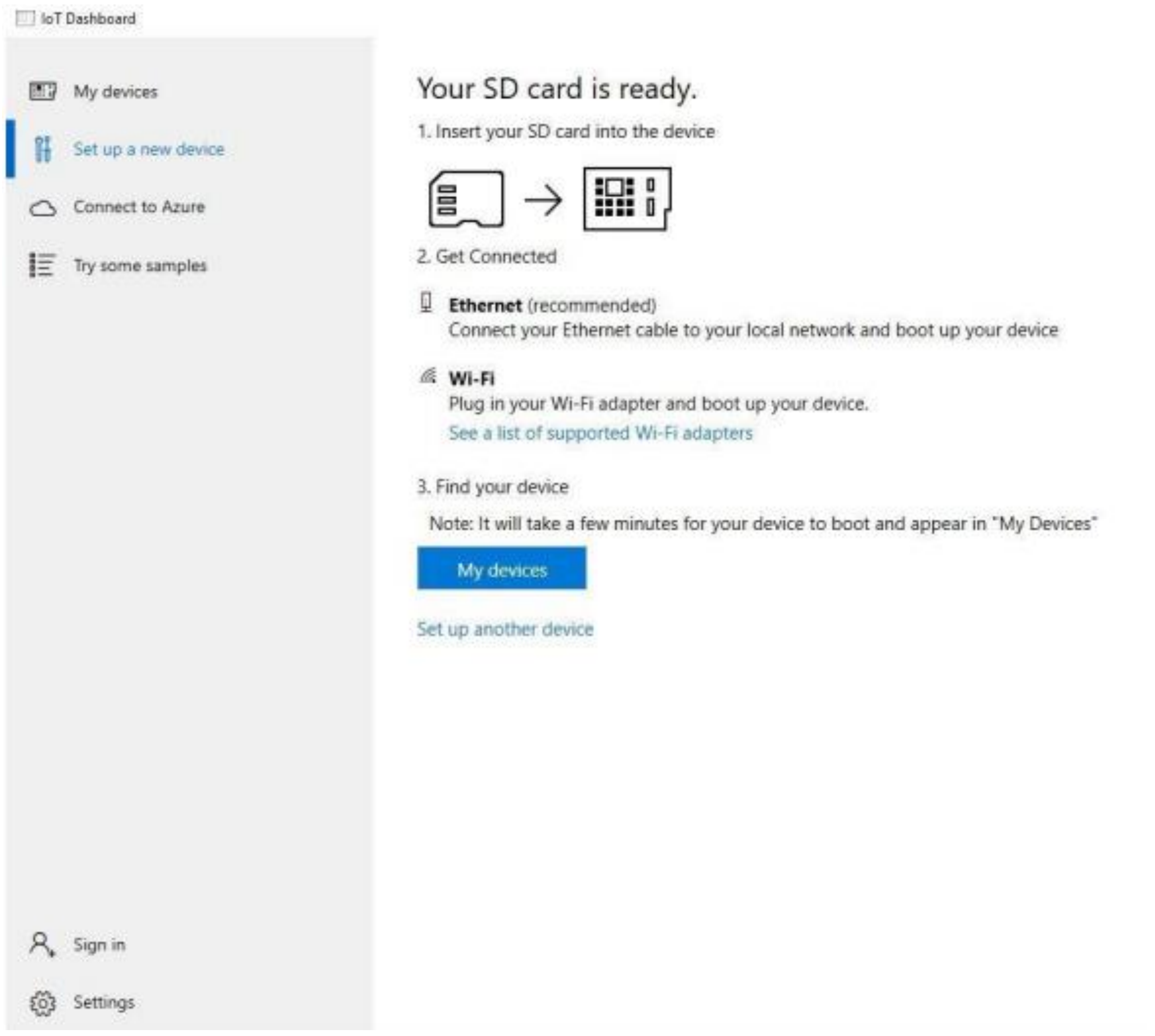
TP-LIN

Only 2.4 Gh
been conne

☑ I accept

Downl
View software license terms
View the list of recommended SD cards
View the list of supported Wi-Fi adapters

Select the WiFi network connection you want your Raspberry Pi to connect to, if required. Only networks your PC connects to will be shown. Click download and install. The application will now download the necessary files from Microsoft and flash them to your microSD card. It'll take a little while, but the dashboard will show you the progress.


IoT Dashboard

Your SD card is ready.

1. Insert your SD card into the device

2. Get Connected

Ethernet (recommended)
Connect your Ethernet cable to your local network and boot up your device

Wi-Fi
Plug in your Wi-Fi adapter and boot up your device.
See a list of supported Wi-Fi adapters

3. Find your device

Note: It will take a few minutes for your device to boot and appear in "My Devices"

Once the image has been installed on the microSD card, it's time to eject it from your PC and go over to the Raspberry Pi. First connect up the micro USB cable and power supply,HDMI cable and USB WiFi adapter or Ethernet cable. Connect the HDMI cable to your chosen display, insert the microSD card into the Raspberry Pi and power it up.