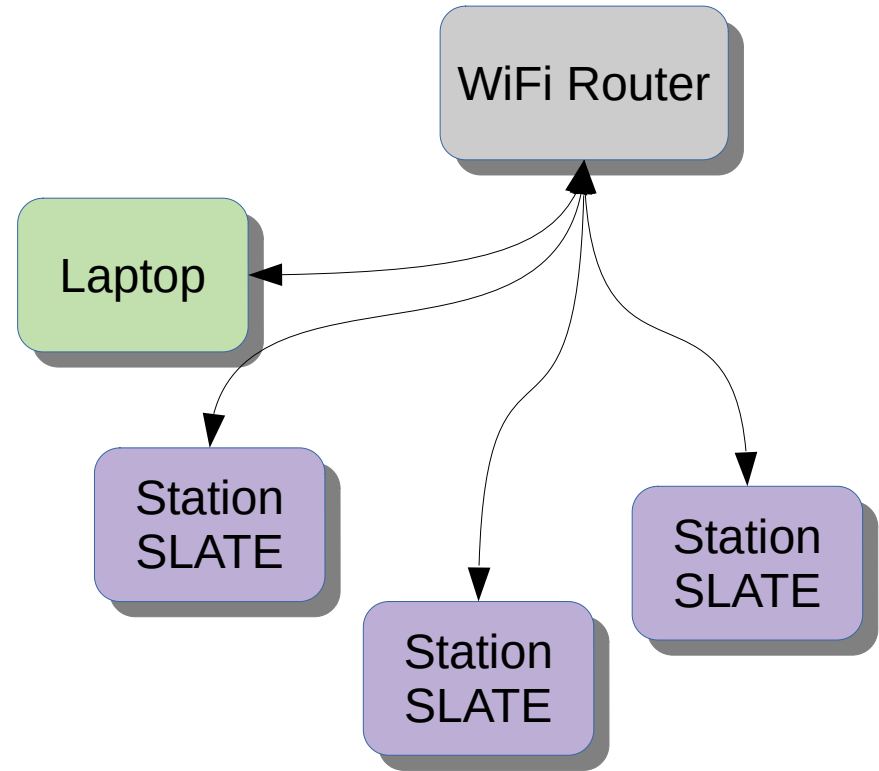


Intro to Internet of Things

Introduction

- Internet of Things also known as **IoT** is a term used for connecting any type of device to the internet to be controlled and/or monitored such as appliances, lighting, locks, etc.
- WiFi routers are everywhere and in many homes. This lesson will explain how multiple SLATEs can be connected to a WiFi router and how to access them through another computer.



- WiFi routers give computers and devices an IP address when the devices connect to the router. The router runs a DHCP server program. DHCP is Dynamic Host Configuration Protocol. It is configured with a range of IP addresses it can send to devices that connect and request an IP address.
- Many WiFi routers are configured by default with an address range of 192.168.0.100 to 192.168.0.254 or 192.168.1.100 to 192.168.1.254. This leaves the addresses below 100 to be used as static IP addresses where the device can be manually configured with an IP address.

Creating a Web Server

- The client and server will use HTTP which is **H**yper **T**ext **T**ransfer **P**rotocol.
- HTTP is stateless. The client has to connect for each data transfer and then disconnect.
- The client sends a request to the server and the server responds.
- There are two request methods, GET and POST. GET will be used here.
- GET requests data from a specified resource and can be used to perform an action. It is the simpler method of communicating with the server.
- Web browsers require a URL to identify a website. It is the website name or address.
- A GET request can be typed into the web browser URL location. Example:
- 192.168.x.x/red/1 (insert SLATE IP address)
- The /red identifies a resource in the web server such as a directory or file.
- This web server will use it to execute a function.

Webserver Module

- The **microdot** module will be used to implement a webserver on the SLATE.
- **Microdot** is a minimalistic python web framework.
- It simplifies creating a webserver.
- Each webpage request triggers a function to perform some operation and respond to the web browser.
- Details can be found at:
<https://github.com/miguelgrinberg/microdot/tree/main>

Simple Webserver

- Connect the RED LED to digital pin 15.
- Connect the GREEN LED to digital pin 13.
- The webserver will control the LEDs.
- Download the **microdot.py** module from canvas and install it into the SLATE.

Setting up as WiFi Host

- In Thonny, open the file connect.py.
- Save it as host.py.
- Delete the lines so that all that remains is shown.
- Save it.
- This sets up the SLATE to be the access point and removes the need for a UDP packet to be sent to start the program.

```
import network
import socket

def connect():
    ap = network.WLAN(network.AP_IF)
    ap.config(essid='ssid', password='password')
    ap.active(True)
```

Web Server Index Page

- For the web server, and **index.html** page needs to be created.
- This is the page that is served when accessing the webserver.
- The page has four buttons to turn on and off the two LEDs.
- The bottom button will open a new web page that displays sensor data.
- We will first focus on the LED buttons.

Web Server

Led Control

LED 1 on

LED 1 off

LED 1 state: 1

LED 2 on

LED 2 off

LED 2 state: 1

Get Sensors

Simple Web Server

- The index web page will be created as a large text string and put in a separate file.
- Save this text into a new file called **index.py**.
- This is a function that generates an html text to be sent to a browser.
- The function has two arguments, **red_state** and **green_state**.
- The arguments are used to update the values LEDS states in the html.

```
def get_html(red_state, green_state):  
    html = f"""  
        <!DOCTYPE html>  
        <html>  
        <head><title>Web Server</title></head>  
        <body>  
            <h1>Web Server</h1>  
            <h2>Led Control</h2>  
            <form action="/red/1">  
                <input type="submit" value="Red LED on" />  
            </form><br>  
            <form action="/red/0">  
                <input type="submit" value="Red LED off" />  
            </form>  
            <p>Red LED state: {red_state}</p><br>  
            <form action="/green/1">  
                <input type="submit" value="Green LED on" />  
            </form><br>  
            <form action="/green/0">  
                <input type="submit" value="Green LED off" />  
            </form>  
            <p>Green LED state: {green_state}</p><br>  
        </body>  
        </html>  
    """  
    return html
```

Simple Web Server

- Variable **html** is assigned the text in triple quotes. All text between the triple quotes is assigned to the variable. The triple quotes allows single quotes to be included and not interpreted.
- the **f** in front of the first set of quotes allows variables to be embedded in the text. The variables are surrounded by **{}**.
- The **f** in front of quotes is called f-string. It allows variables, expressions, and functions to be embedded directly in the string.

```
def get_html(red_state, green_state):  
    html = f"""  
        <!DOCTYPE html>  
        <html>  
        <head><title>Web Server</title></head>  
        <body>  
            <h1>Web Server</h1>  
            <h2>Led Control</h2>  
            <form action="/red/1">  
                <input type="submit" value="Red LED on" />  
            </form><br>  
            <form action="/red/0">  
                <input type="submit" value="Red LED off" />  
            </form>  
            <p>Red LED state: {red_state}</p><br>  
            <form action="/green/1">  
                <input type="submit" value="Green LED on" />  
            </form><br>  
            <form action="/green/0">  
                <input type="submit" value="Green LED off" />  
            </form>  
            <p>Green LED state: {green_state}</p><br>  
        </body>  
        </html>  
    """  
    return html
```

Simple Web Server

- The html **<form>** is used with an action attribute.
- The action specifies which LED and the state with 1 being on and 0 being off.
- The input type is **submit** which creates a button. value is the text in the button.
- When the button is pressed, the browser sends a GET request to the server.
- The URL generated would be
 - `http://192.168.4.1/red/1?`

```
def get_html(red_state, green_state):
    html = f"""
        <!DOCTYPE html>
        <html>
        <head><title>Web Server</title></head>
        <body>
            <h1>Web Server</h1>
            <h2>Led Control</h2>
            <form action="/red/1">
                <input type="submit" value="Red LED on" />
            </form><br>
            <form action="/red/0">
                <input type="submit" value="Red LED off" />
            </form>
            <p>Red LED state: {red_state}</p><br>
            <form action="/green/1">
                <input type="submit" value="Green LED on" />
            </form><br>
            <form action="/green/0">
                <input type="submit" value="Green LED off" />
            </form>
            <p>Green LED state: {green_state}</p><br>
        </body>
        </html>
    """
    return html
```

- Start a new program in Thonny. Name it `webserver.py`.
- At the start of the program, the needed modules are imported.
- Digital pins 13 and 15 are configured as outputs to control the LEDs.
- The WiFi is activated.
- The Microdot web server module is instantiated and named **app**.
- The first function sends the index html to the browser.
- The `@app.route()` is a decorator and the argument specifies the action to trigger the function.

```
from machine import Pin, I2C
from microdot import Microdot
import host
from index.py import get_html

red = Pin(15, Pin.OUT)
green = Pin(13, Pin.OUT)

host.connect()

app = Microdot()

@app.route('/')
async def index(request):
    html = get_html(red.value(), green.value())
    return html, 200, {'Content-Type': 'text/html'}
```

Connecting to Router

- A function for each LED is added.
- Each one has a decorator that specifies the action. Notice that it matches the action in the html except that the number is replaced with `<action>`.
- The function gets the action value as a string.
- The action variable value is checked and the LED is turned on or off.
- The html is updated with the current LED state and returned.
- Every action must return an html string or text string otherwise the webbrowser will wait until it times out.

```
@app.route('/red/<action>')
def control_led(request,action):
    if action == '1':
        red.value(1)
    elif action == '0':
        red.value(0)
    html = get_html(red.value(),green.value())
    return html,200,{'Content-Type': 'text/html'}

@app.route('/green/<action>')
def control_led(request,action):
    if action == '1':
        green.value(1)
    elif action == '0':
        green.value(0)
    html = get_html(red.value(),green.value())
    return html,200,{'Content-Type': 'text/html'}

app.run(port=80)
```

Connecting to Router

- The last line starts the web server and uses port 80 which is the standard port for a webserver.
- Run the webserver.py program.
- Open a web browser and enter the URL:
 - <http://192.168.4.1/>
- There may be a delay of up to a minute but the web server should respond and a web page should appear. Afterwards, the web server will respond quickly.
- Click on each button and verify the LEDs will turn on and off.
- Update the webserver.py and index.py files and add the Yellow LED.

```
@app.route('/red/<action>')
def control_led(request,action):
    if action == '1':
        red.value(1)
    elif action == '0':
        red.value(0)
    html = get_html(red.value(),green.value())
    return html,200,{'Content-Type': 'text/html'}

@app.route('/green/<action>')
def control_led(request,action):
    if action == '1':
        green.value(1)
    elif action == '0':
        green.value(0)
    html = get_html(red.value(),green.value())
    return html,200,{'Content-Type': 'text/html'}

app.run(port=80)
```

Displaying Sensor Data

- A button will be added to go to a new web page and display the sensor data.
- The temperature, air pressure, and relative humidity will be displayed and updated once a second.
- Open index.py and add the bold html text into the string. This adds a button to open a web page to display the sensor data.

```
def get_html(red_state, green_state):  
    html = f"""  
        <!DOCTYPE html>  
        <html>  
        <head><title>Web Server</title></head>  
        <body>  
            <h1>Web Server</h1>  
            <h2>Led Control</h2>  
            <form action="/red/1">  
                <input type="submit" value="Red LED on" />  
            </form><br>  
            <form action="/red/0">  
                <input type="submit" value="Red LED off" />  
            </form>  
            <p>Red LED state: {red_state}</p><br>  
            <form action="/green/1">  
                <input type="submit" value="Green LED on" />  
            </form><br>  
            <form action="/green/0">  
                <input type="submit" value="Green LED off" />  
            </form>  
            <p>Green LED state: {green_state}</p><br>  
            <form action="/sensors">  
                <input type="submit" value="Get Sensors"/>  
            </form>  
        </body>  
        </html>  
    """  
    return html
```

Displaying Sensor Data

- Create a new file and save the code to the right as **sensor_page.py**.
- This web page displays the sensor data as text and updates once a second.
- The `<meta>` line sets the refresh rate of the web page.

```
def get_sensor_page(p,h,t):  
    html =f"""  
        <!DOCTYPE html>  
        <html>  
        <head><title>Web Server</title></head>  
        <meta http-equiv="refresh" content="1">  
        <body>  
            <h1>Sensor Page</h1>  
            <h2>Pressure {p}</h2>  
            <h2>Humidity {h}</h2>  
            <h2>Temperature {t}</h2>  
        </body>  
        </html>  
    """  
    return html
```

- At the top of the webserver.py code add the highlighted code.
- The sensor modules are imported.
- The sensors_page module is imported.
- The I2C bus is initialized and the sensors are initialized.

```
from machine import Pin, I2C
from microdot import Microdot
import wifi
from index.py import get_html
from bmp280 import BMP280
import ahtx0
from sensors_page import get_sensor_page

red = Pin(15, Pin.OUT)
green = Pin(13, Pin.OUT)

i2c = I2C(sda=Pin(4), scl=Pin(5))
sensor = ahtx0.AHT10(i2c)
bmp = BMP280(i2c)

wifi.connect()

app = Microdot()

@app.route('/')
async def index(request):
    html = get_html(red.value(), green.value())
    return html, 200, {'Content-Type': 'text/html'}
```

Connecting to Router

- Moving to the bottom of the program, enter the function above the last line
- The functions gets the sensor values and calls the sensor function to create an html string to send back to the browser.
- Run the webserver program.
- Open the web page in a web browser using the URL:
 - <http://192.168.4.1/>
- Click on the sensor button and the sensor page should appear and the sensor values should update once a second.

```
@app.route('/sensors')
def read_sensors(request):
    p = bmp.pressure
    h = sensor.relative_humidity
    t = bmp.temperature
    html = get_sensor_page(p,h,t)
    return html,200,{'Content-Type': 'text/html'}

app.run(port=80)
```

Things to Try

- You can update the html strings to make the web pages nicer.
- You can ask AI to generate web pages with features.
- As a challenge, create a web page that displays two dials showing temperature and humidity that updates every second.
- Below was generated by AI by describing the features.
- Make sure to request that the html generated be self contained and only use an embedded style sheet and html.
- Require that the html be generated by a python function with two arguments for temperature and humidity.

