

Node Web Server with Python control for Raspberry Pi

June 16, 2021

Table of Contents

Introduction.....	2
Useful resources.....	3
GPIO Wiring.....	4
Initial setup of Raspberry Pi.....	5
Find the IP address of your Pi.....	6
Install Node JS.....	7
Overview.....	7
Installing node.js.....	8
The onoff Module.....	10
Installing socket.io.....	11
socket.io.js client side javascript file.....	11
Allow web server to run on a port below 1024.....	11
RaspberryPi Zero W considerations.....	12
Retrieve and running the webserver code.....	13
Testing the communication interface with netcat.....	15

Introduction

This demo project shows you how to create a web server on your Raspberry Pi using node-js and use it as a front end webserver for another program (e.g. Python, C, etc) or even another computer or specialize controller. It is up to your other program to updated data to all clients as it occurs and to control all other functions such as changing the state of the GPIO pins.

We will use a simple UDP socket connection to communicate with our other program. If this other program is running on the same Pi, we will use a simple loopback connection (127.0.0.1) to communicate between programs. Nothing about this example program is secure. If you are passing sensitive data back and forth, you should consider upgrading the web service to https and use an ssh connection to communicate to another computer/controller. Setting up a secure web server is beyond the scope of this demo program.

You should watch my first you tube video at <https://youtu.be/TVxQROFPjy0> on how to install node JS and running the GPIO demo.

This demo project assume you have a basic knowledge of the Raspberry Pi and are well familiar with the command line, and installing the Raspberry Pi OS on a Raspberry Pi. So we will not be covering those topics in this demo project. If you are unfamiliar with these basic concepts, refer to the literally thousands of tutorials and videos that others have made on getting started with the Raspberry Pi.

For this demo, we will be using the following hardware and software.

- Raspberry Pi OS 2020-12-02-raspbian-buster-armhf-full.zip
- Node.JS v12.20.1 LTS or any version of Node.JS v12.x
- A Raspberry Pi 4 Model B with 4GB of memory and a 16GB or larger SD card
- A HDMI display, keyboard, and mouse is advisable at least for the initial setup.
- Either a WiFi or LAN network connection to your Pi

Other version of software and hardware will probably work however you may have to make some adjustments to this procedure. In particular, I had issues with Node.js v14.x and had to revert back to v12.x to resolve those issue. If you have issues, I found that <https://stackoverflow.com> to be a valuable resource in resolving issues.

There are literally thousands of ways that you could implement this project. There are packages such a JQuery, Express, React, Angular, PHP, and Wordpress that can aid in web development. However, I found that just sticking to plain vanilla html, css, javascript, and node.js was the easiest to learn and implement.

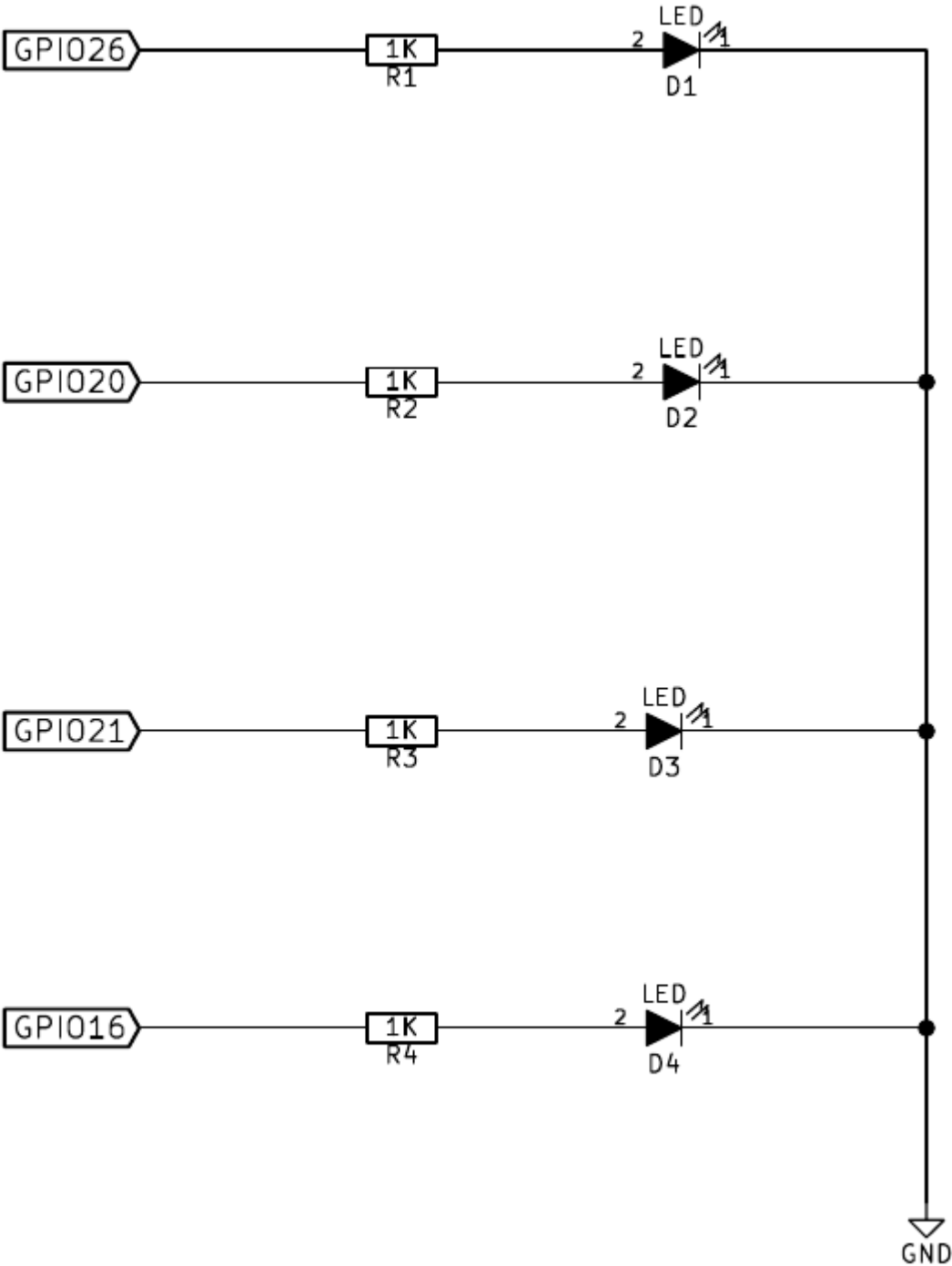
Useful resources

<https://nodejs.org/en/>
<https://www.w3schools.com/nodejs/default.asp>
<https://socket.io/docs/#What-Socket-IO-is-not>
<https://socket.io/get-started/chat>
<https://stackoverflow.com>
<https://nodejs.org/dist/v6.3.0/docs/api/dgram.html>

Videos from Traversy Media where particularly useful when learning web development. Here are just some of the videos I found useful:

HTML: <https://www.youtube.com/watch?v=UB1O30fR-EE>
JavaScript: <https://www.youtube.com/watch?v=hdI2bqOjy3c>
Node.js: <https://www.youtube.com/watch?v=fBNz5xF-Kx4>
JavaScript DOM: <https://www.youtube.com/watch?v=0ik6X4DJKCc>
CSS: <https://www.youtube.com/watch?v=yfoY53QXEnI>
AJAX: <https://www.youtube.com/watch?v=82hmvUYY6QA>

GPIO Wiring

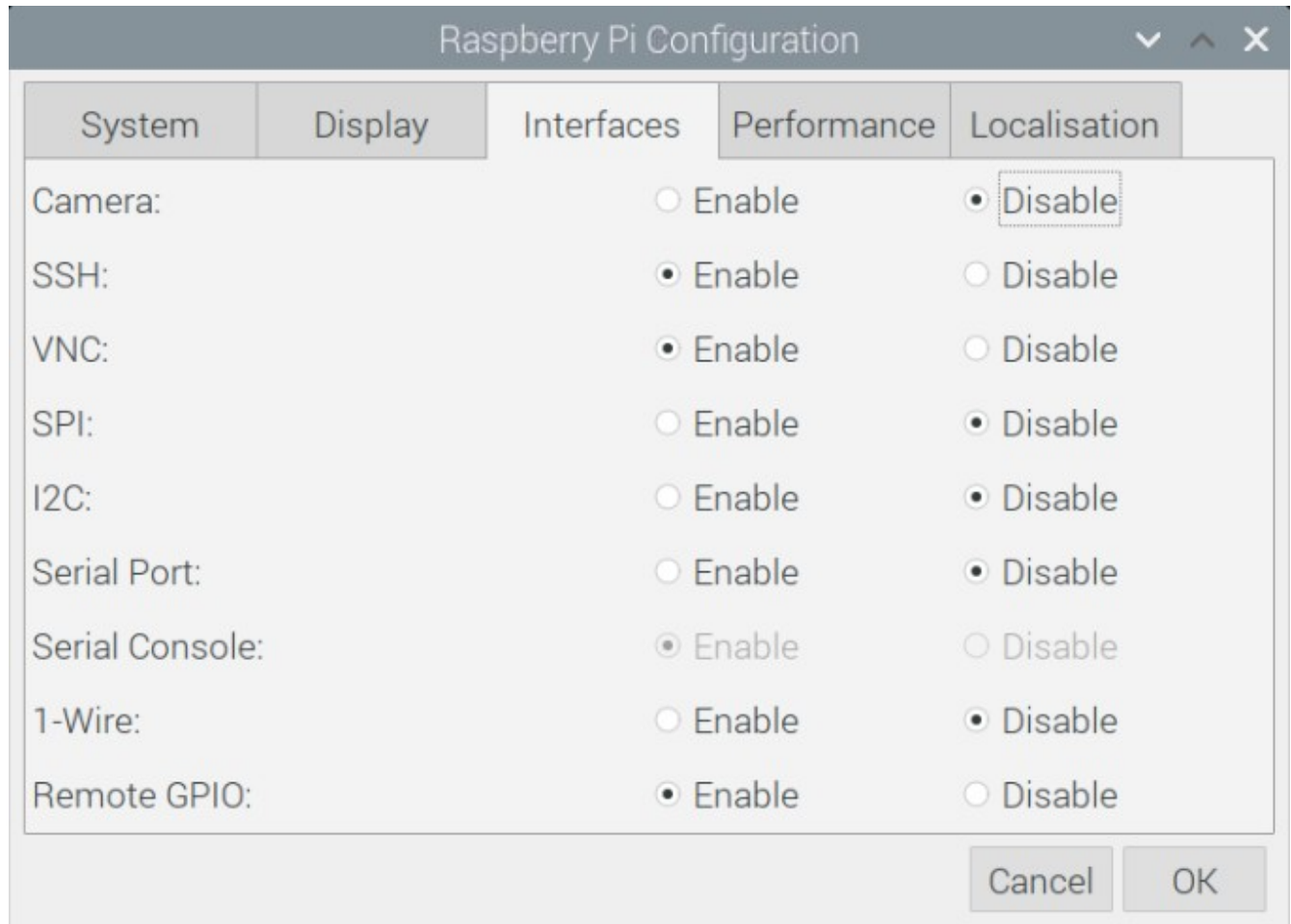


Initial setup of Raspberry Pi

Install a clean version Raspberry Pi OS on your Pi. I recommend you use the desktop with recommended software version. Refer to the introduction for the version I used. You can download the latest version of software at:

<https://www.raspberrypi.org/software/operating-systems/>

Once you go through the initial installation and update the Pi, click the menu button and go to "Preferences/Raspberry Pi Configuration" and enable "SSH" and "Remote GPIO". If you wish to have a remote Desktop to the Pi using RealVNC software, enable "VNC" too.



The screenshot shows the 'Raspberry Pi Configuration' window with the 'Interfaces' tab selected. The window has a title bar with a dropdown arrow, an up arrow, and a close button. Below the title bar are five tabs: 'System', 'Display', 'Interfaces', 'Performance', and 'Localisation'. The 'Interfaces' tab is active, showing a list of hardware interfaces with 'Enable' and 'Disable' radio buttons. The 'Camera' interface has 'Disable' selected and is highlighted with a dashed border. The 'SSH' and 'Remote GPIO' interfaces have 'Enable' selected. The other interfaces (VNC, SPI, I2C, Serial Port, Serial Console, 1-Wire) also have 'Disable' selected. At the bottom right are 'Cancel' and 'OK' buttons.

System	Display	Interfaces	Performance	Localisation
Camera:		<input type="radio"/> Enable		<input checked="" type="radio"/> Disable
SSH:		<input checked="" type="radio"/> Enable		<input type="radio"/> Disable
VNC:		<input checked="" type="radio"/> Enable		<input type="radio"/> Disable
SPI:		<input type="radio"/> Enable		<input checked="" type="radio"/> Disable
I2C:		<input type="radio"/> Enable		<input checked="" type="radio"/> Disable
Serial Port:		<input type="radio"/> Enable		<input checked="" type="radio"/> Disable
Serial Console:		<input checked="" type="radio"/> Enable		<input type="radio"/> Disable
1-Wire:		<input type="radio"/> Enable		<input checked="" type="radio"/> Disable
Remote GPIO:		<input checked="" type="radio"/> Enable		<input type="radio"/> Disable

Cancel OK

Find the IP address of your Pi

Open a terminal on the Pi and execute "ifconfig" to get the IP address of your pi. You will use that later to SSH into the Pi and to access the web server. If you wish, you can assign a static IP to you Pi however I usually prefer to give the Pi a reserved DHCP in my router's DHCP server.

ifconfig

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.18.105 netmask 255.255.255.0 broadcast 192.168.18.255
    RX packets 5779 bytes 523040 (510.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7188 bytes 4134507 (3.9 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

In this case, we are connected by the LAN port and our IP address is 192.168.18.105. So we will use that IP address for the rest on this demo. So where you see 192.168.18.105, you want to replace it with the IP address of your Pi.

Install Node JS

Overview

Node.js provides a method to run Javascript on your web server. Normally when people refer to javascript, they are talking about javascript that runs in the client's web browser. We will use node.js to implement our web server using javascript that runs on the server side instead of the client side. This has an advantage in that we don't have to learn a different programming language on the server side code. Both the client and server side uses javascript programming.

There is really no installation required for node other than unzipping the archive and placing it in your /usr/local folder. Your webserver files can be located in any other folder and usually in your home folder.

Node.js also supports websockets natively which is a big plus for it over an Apache2/PHP implementation. However, you do need to download the code from nodejs.org as the node.js package in the Raspberry Pi repository is usually too old to run the npm (node packet manager).

We will install Socket.io as that provides an easy interface to web sockets. Generally you want to include a reference socket.io.js in your html file. When you web client first connects to the pi, socket.io establishes a long-polling connection for the client, then socket.io tries to upgrade to better transports on the client side, like WebSockets.

Installing node.js

Blue text refers to commands that you execute in a terminal on the pi and green text is the response you get back from the pi.

To install node.js on a raspberry pi, open a terminal and execute:

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade
```

Next we need to determine what ARM processor the Pi is using. Execute:

```
uname -m
armv7l
```

So we are using an ARM7 processor. Go to <https://nodejs.org/en/download/> and download ARMv7 version of Node.js. In this case, we will be installing node-v12.20.1-linux-armv7l. As of May 31, 2021, the latest versions of Node still do not work on the Pi. Node v12.x is still the best version to use for the Raspberry Pi 4B. You can download that version at <https://nodejs.org/dist/latest-v12.x/> and modify the commands below for that version.

If you are using a Raspberry Pi Zero, see the "RaspberryPi Zero W considerations" section of this document. I have successfully run this project on a Raspberry Pi Zero W however you need to install a different version of NodeJS and install a couple of extra packages.

```
cd ~/Downloads
wget https://nodejs.org/dist/latest-v12.x/node-v12.22.1-linux-armv7l.tar.xz
```

```
ls -l
node-v12.22.1-linux-armv7l.tar.xz
```

Next extract the files from the archive:

```
tar -xvf node-v12.22.1-linux-armv7l.tar.xz
```

```
ls -l
total 12084
drwxr-xr-x 6 pi pi      4096 Apr  6 10:09 node-v12.22.1-linux-armv7l
-rw-r--r-- 1 pi pi 12368700 Apr  6 10:10 node-v12.22.1-linux-armv7l.tar.xz
```

Then open the directory with the node files and confirm they have been extracted.

```
cd node-v12.22.1-linux-armv7l
ls -l
total 184
drwxr-xr-x 2 pi pi   4096 Apr  6 10:09 bin
-rw-r--r-- 1 pi pi 54147 Apr  6 10:09 CHANGELOG.md
drwxr-xr-x 3 pi pi   4096 Apr  6 10:09 include
drwxr-xr-x 3 pi pi   4096 Apr  6 10:09 lib
-rw-r--r-- 1 pi pi 81255 Apr  6 10:09 LICENSE
-rw-r--r-- 1 pi pi 28766 Apr  6 10:09 README.md
drwxr-xr-x 5 pi pi   4096 Apr  6 10:09 share
```


Copy files to your /usr/local directory

```
sudo cp -R * /usr/local/
```

Check if node and npm have been installed correctly.

```
node -v
```

```
v12.22.1
```

```
npm -v
```

```
6.14.12
```

Next we need to install a few modules using the node package manager (npm). But first we need to create a directory for our node files and initialize the npm. That will create a json file that will be needed when we install packages.

```
mkdir ~/webserver/
```

```
cd ~/webserver
```

```
npm init
```

For the most part, you can use the defaults when asked a question by npm init. Add a description for your project. It is okay to leave the repository blank.

The onoff Module

Since we are not controlling the GPIO from node on this example, it is not necessary to install the onoff module. However you can leave it installed if you already have installed it.

Installing socket.io

To install socket.io, open a terminal on the Raspberry Pi and execute:

```
npm install socket.io --save
+ socket.io@3.0.5
added 23 packages from 74 contributors and audited 29 packages in 3.547s
found 0 vulnerabilities
```

socket.io.js client side javascript file

In your html, include a reference to:

```
<script src="/socket.io/socket.io.js"></script>
```

While `"/socket.io/socket.io.js"` looks like an invalid url, it works because you wrap your HTTP server in Socket.IO and it intercepts requests for `/socket.io/socket.io.js` and sends the appropriate response automatically. If you want to inspect socket.io.js, you can download it to your current directory (assuming your web server is running on port 80) by executing the following:

```
wget 192.168.18.105:80/socket.io/socket.io.js
```

Allow web server to run on a port below 1024

If you want to run the node web server on a port lower than 1024 without running node as root (generally a bad idea to run something as root), you need to run following from a terminal on the pi to give node access to the lower ports without running it as root. You only need to execute this one.

```
sudo apt update
sudo apt install libcap2-bin
sudo setcap cap_net_bind_service=+ep /usr/local/bin/node
```

You can later check if a program has extra privileges by executing:

```
getcap /usr/local/bin/node
/usr/local/bin/node = cap_net_bind_service+ep
```

To see all files in a directory and subdirectories that have elevated privileges, execute:

```
getcap -r /usr/
/usr/local/bin/node = cap_net_bind_service+ep
```

RaspberryPi Zero W considerations

If you are using a RaspberryPi zero-W, it will report:

```
uname -m  
armv6l
```

Support for armv6l has been dropped in later version of nodes. If you wish to use a Raspberry Pi zero, you will have to use node v11.x instead. You can download this version at:

<https://nodejs.org/dist/latest-v11.x/>

When you run

```
npm install socket.io --save
```

You will likely get errors like:

```
npm WARN ws@7.4.6 requires a peer of bufferutil@^4.0.1 but none is installed. You must install peer dependencies yourself.  
npm WARN ws@7.4.6 requires a peer of utf-8-validate@^5.0.2 but none is installed. You must install peer dependencies yourself.
```

If you do, run the following commands:

```
npm install --save-optional utf-8-validate  
npm install --save-optional bufferutil  
npm install socket.io --save
```

Otherwise, the setup is exactly the same for a Raspberry Pi Zero W and this example program will run on it too. I have successfully run this project on the Zero W using the following versions

RaspberryPi OS: 2021-05-07-raspbian-buster-armhf-lite
Node.JS: node-v11.15.0-linux-armv6l

If you are using the headless lite version of RaspberryPi OS like I did on the Zero W, you will need to do everything from the command line which requires a little more skill.

Retrieve and running the webserver code

Open a terminal or ssh connection on your Pi.

```
ssh -X pi@192.168.18.105
cd ~/Downloads
ls -l
total 12080
drwxr-xr-x 6 pi pi      4096 Jan  4 05:27 node-v12.20.1-linux-armv7l
-rw-r--r-- 1 pi pi 12362220 Jan  4 05:27 node-v12.20.1-linux-armv7l.tar.xz

rm -r main.zip
wget https://github.com/StevesRandomProjects/NodeWebServerForPython/archive/refs/heads/main.zip
ls -l
total 12088
-rw-r--r-- 1 pi pi      8314 Jan 17 18:28 main.zip
-rw-r--r-- 1 pi pi 12362220 Jan  4 05:27 node-v12.20.1-linux-armv7l.tar.xz

unzip main.zip
ls -l
total 12092
-rw-r--r-- 1 pi pi     13795 Jun 16 01:18 main.zip
drwxr-xr-x 6 pi pi      4096 Apr  6 10:09 node-v12.22.1-linux-armv7l
-rw-r--r-- 1 pi pi 12368700 Apr  6 10:10 node-v12.22.1-linux-armv7l.tar.xz
drwxr-xr-x 3 pi pi      4096 Jun 16 01:16 NodeWebServerForPython-main

cd NodeWebServerForPython-main
ls -l
total 12104
-rw-r--r-- 1 pi pi     13795 Jun 16 01:18 main.zip
drwxr-xr-x 6 pi pi      4096 Apr  6 10:09 node-v12.22.1-linux-armv7l
-rw-r--r-- 1 pi pi 12368700 Apr  6 10:10 node-v12.22.1-linux-armv7l.tar.xz
drwxr-xr-x 3 pi pi      4096 Jun 16 01:16 NodeWebServerForPython-main

mkdir ~/webserver/webgui
cp -R * ~/webserver/webgui/
cd ~/webserver/webgui

ls -l
total 28
-rw-r--r-- 1 pi pi  1077 Jun 16 01:25 LICENSE
drwxr-xr-x 4 pi pi  4096 Jun 16 01:25 public
-rw-r--r-- 1 pi pi   98 Jun 16 01:25 README.md
-rw-r--r-- 1 pi pi  5774 Jun 16 01:25 txrx.py
-rw-r--r-- 1 pi pi  7484 Jun 16 01:25 webserver.js
```

You should modify the webserver.js code with the IP address of your computer (line 18).

```
geany ~/webserver/webgui/webserver.js
const RemoteAddress1 = '192.168.18.34'
```

to point to your computer instead of you want to run a UDP connection from your computer. Also be sure to let port 3000 pass the firewall on your computer.

And finally, this command will start our webserver
`node webserver.js`

Your webserver should now be running. Open a web browser and go to the IP address of your Pi. In my case, that is:

<http://192.168.18.105/>

It won't do anything in terms of controlling the LEDs or displaying feedback because that function has been delegated to the Python program which is not running yet.

To stop the web server, press Ctrl-C in the terminal window. However don't do that yet because we need to check a few more things out.

Testing the communication interface with netcat

To test the UDP computer link, open a UDP connection from your computer. I am running Linux Mint 19.2 and for me, all I have to do is open a terminal and execute

```
netcat -u 192.168.18.105 3000 -p 3000
```

If you are running this command on the raspberry pi, use this instead

```
netcat -u 127.0.0.1 3000 -p 3001
```

You won't start receiving data from the Pi until you send something to the pi. So just press the enter key to open the communication port.

As you press buttons in your web browser, you will see the JSON values for those button presses in your terminal. For example, if I press and release the "GPIO26" toggle switch button, I will see

```
{"GPIO26T":1}
```

```
{"GPIO26T":0}
```

```
{"GPIO26T":1}
```

 is sent when you press the button

```
{"GPIO26T":0}
```

 is sent when you release the button.

Change the feedback of GPIO26 to on, send this through the UDP connection

```
{"GPIO26T":1}
```

To turn the feedback back off, send this

```
{"GPIO26T":0}
```

I have added a lot of other stuff to the webpage to demo other html widgets such as volume slide, analog gauge, input selection button with different feedback, and text boxes. None of those are programmed to do anything in this demo program however you can test them with netcat. Just press one of the buttons and your UDP comlink will display what its JSON value is.

You will notice that the json values will begin with

```
{"Dx", y}
```

 when you press a button where Dx is the button ID in the html file and y is either a 1 (indicating the button is pressed) or 0 indicating the button is released.

```
{"Ax", y}
```

 when you press the slider where Ax is the slider ID in the html file and y is a number between 1 and 100. y indicates the slider value.

```
{"Sx","some text"}
```

 indicating a text entry box.

If you are running a project with many elements, I find it is better to use some type of naming scheme where you can parse the json values in your external program and quickly execute the code for that button. Otherwise, you will have to do a bunch of "if else" statements which can slow down the response time.

To stop the UDP connection on your computer, press Ctrl-C in the terminal window.

Running the Python GPIO demo program

To run the python demo program on your pi, ssh into your pi with another terminal, and execute the txrx.py demo python3 program.

```
ssh pi@192.168.18.105  
cd ~/webserver/webgui  
python3 txrx.py
```

You will now see the volume slider toggling between 20 and 80 in your web browser. That is mainly to demonstrate that you can have a process running in the background making changes to the web page without requiring some type of user input. And the GPIO buttons will work just like the GPIO example with the all Node JS implementation. The only difference is you are using Node JS as a web server interface and all the logic behind it is handled by the python program.

As you press buttons on the web page that are not programmed, you will see the corresponding JSON values for those buttons in your python console. You can add logic to your python program to react to these button presses.

To stop the python demo program, press Ctrl-C twice.