Week 6: Homework 2: Blinking LED Websocket server

https://hc.labnet.sfbu.edu/~henry/npu/classes/embed/raspberry_pi/slide/exercise_raspberry_pi.htmlLinks to an external site.

Q19 ==> Blinking LED Websocket server

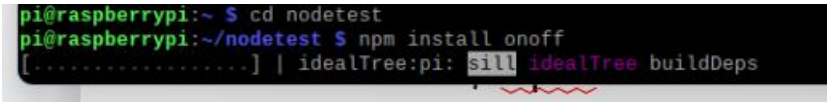
1. Blinking LED Websocket server
    - Please implement a Blinking LED websocket server on Raspberry Pi.
    - Reference
        - Quiz
        - Controlling LEDs by node.js
        - Implement WebSockets server using Node.js

Step 1: Install Package

Install websocket

$ npm install websocket

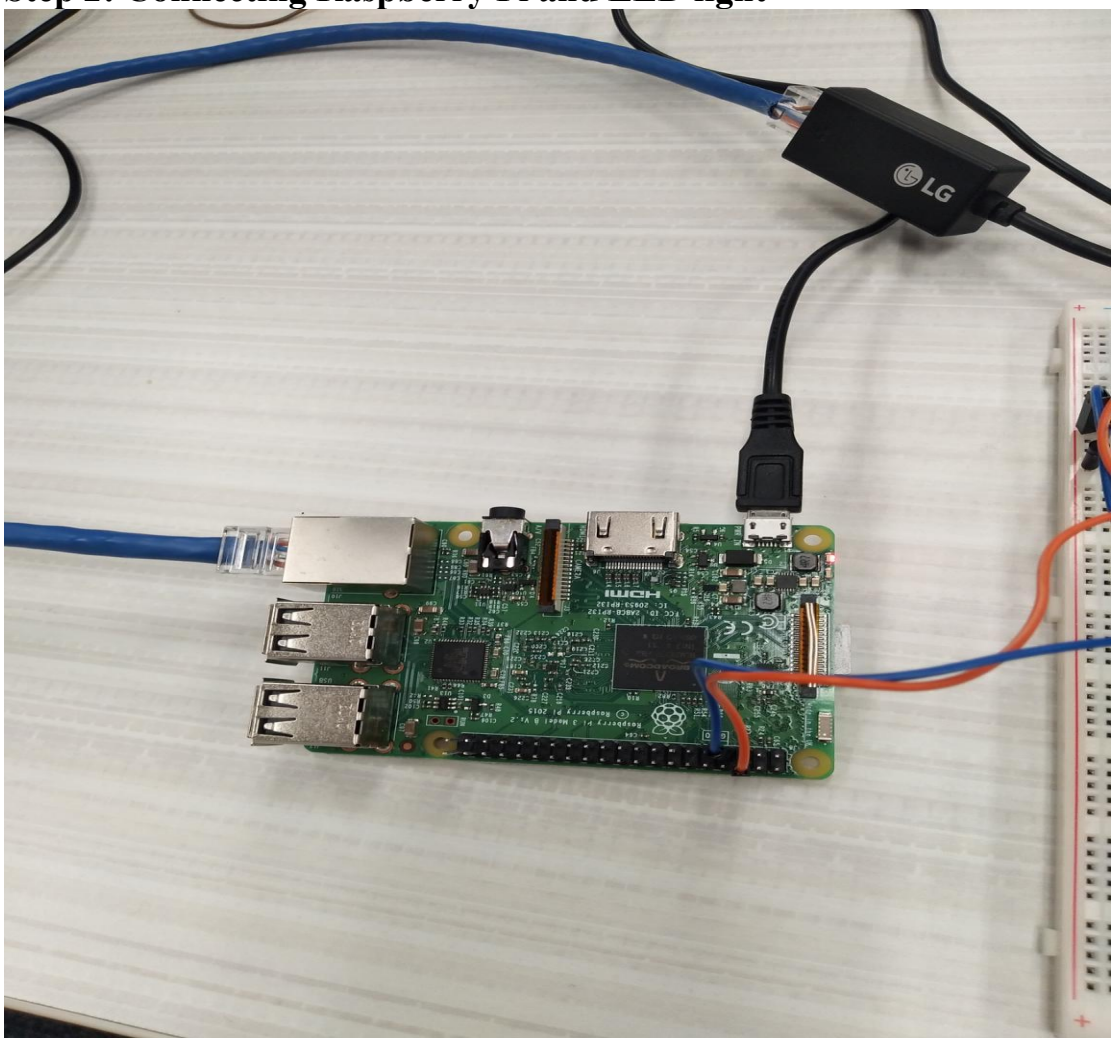
$ npm install onoff

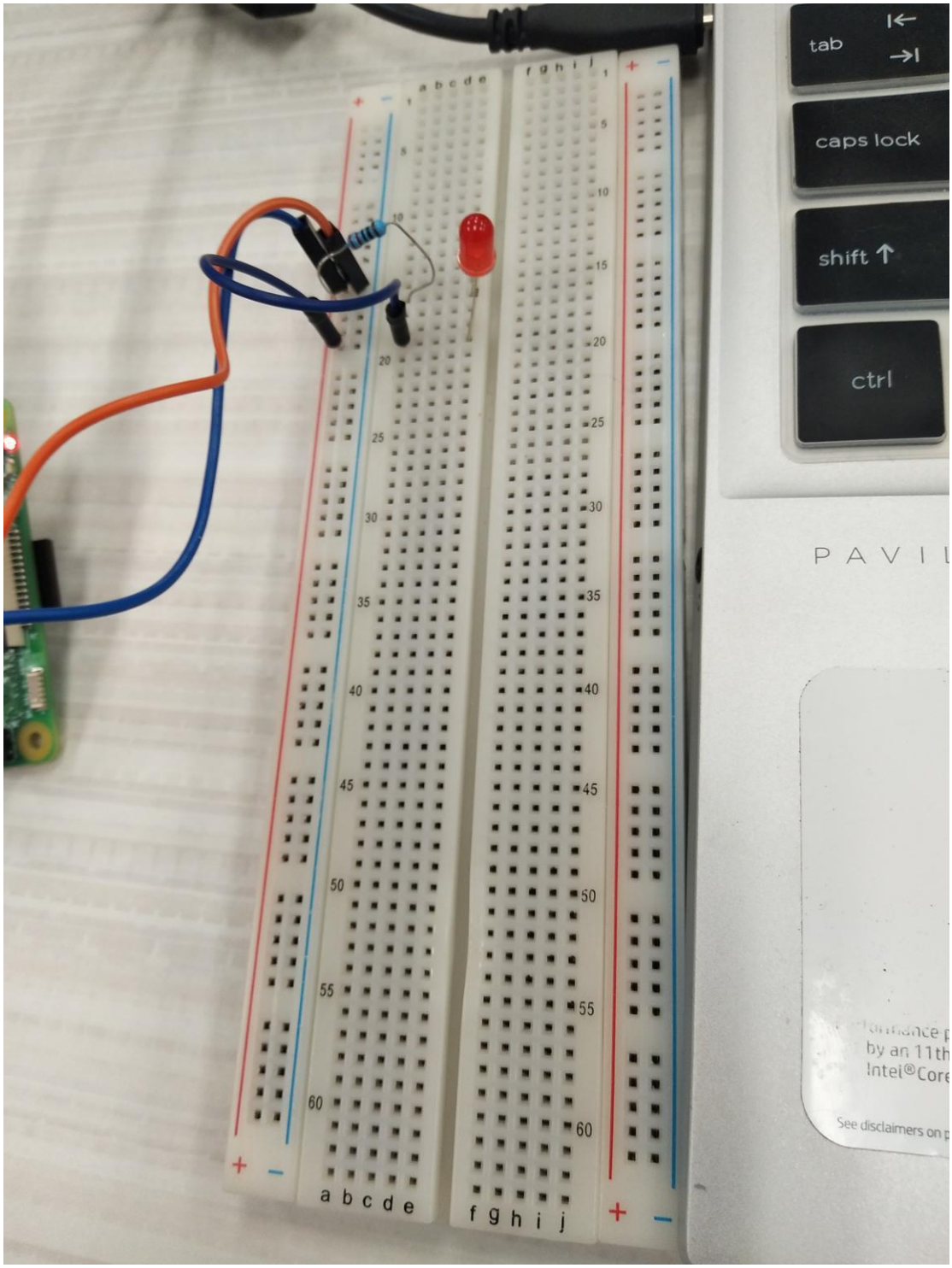


$ pigpiod -v
$ sudo apt-get update
$ sudo apt-get update
$ sudo apt-get install pigpio
$ npm install pigpio

```
added 17 packages, and audited 45 packages in 41s

found 0 vulnerabilities
pi@raspberrypi:~ $ cd nodetest
pi@raspberrypi:~/nodetest $ npm install onoff

up to date, audited 45 packages in 1m

found 0 vulnerabilities
pi@raspberrypi:~/nodetest $ pigpiod -v
79
pi@raspberrypi:~/nodetest $ sudo apt-get update
Get:1 http://raspbian.raspberrypi.org/raspbian bullseye InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian bullseye InRelease [23.6 kB]
Hit:3 https://deb.nodesource.com/node_10.x bullseye InRelease
Get:4 http://archive.raspberrypi.org/debian bullseye/main armhf Packages [311 kB]
Fetched 349 kB in 4s (91.8 kB/s)
Reading package lists... Done
pi@raspberrypi:~/nodetest $ sudo apt-get install pigpio
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
pigpio is already the newest version (1.79-1+rpt1).
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
pi@raspberrypi:~/nodetest $ npm install pigpio

added 1 package, and audited 46 packages in 36s

found 0 vulnerabilities
pi@raspberrypi:~/nodetest $
```

Step 2: Connecting Raspberry Pi and LED light





Step 3: Code Source:

websocket_blink.js

```
#!/usr/bin/env node

// - Node.js installation
```

```
// - The module websocekt is installed by this command

//    npm install -g websocket

// - If you want to use a formal protocol, you need to replace

//      var WebSocketServer = require(

//
'C:/Users/Henry/AppData/Roaming/npm/node_modules/websocket').server;

//   with

//       var WebSocketServer = require("ws").Server

// - Please refer Zhomart Mukhamejanov's example if you want to deploy
server.js on Heroku

// - Please refer Vidit Mody's use of ws protocol

//var WebSocketServer =
require('C:/Users/Henry/AppData/Roaming/npm/node_modules/websock
et').server;

var WebSocketServer = require('websocket').server;

var http = require('http');



var server = http.createServer(function(request, response) {

    console.log((new Date()) + ' Received request for ' + request.url);

    response.writeHead(404);

    response.end();

});

server.listen(8080, function() {
```

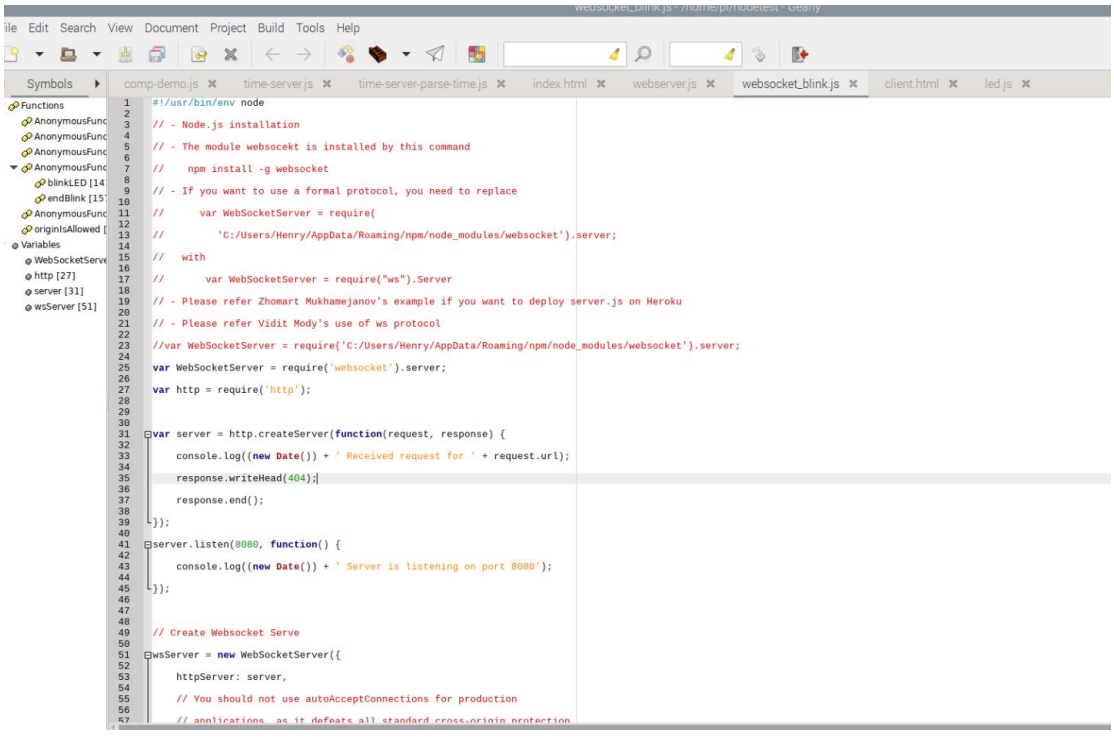
```
    console.log((new Date()) + ' Server is listening on port 8080');
});

// Create Websocket Serve
wsServer = new WebSocketServer({
    httpServer: server,
    // You should not use autoAcceptConnections for production
    // applications, as it defeats all standard cross-origin protection
    // facilities built into the protocol and the browser.  You should
    // *always* verify the connection's origin and decide whether or not
    // to accept it.
    autoAcceptConnections: false
});

function originIsAllowed(origin) {
  // Put logic here to detect whether the
  // specified origin (i.e., client) is allowed.
  return true;
}

wsServer.on('request', function(request) {
```

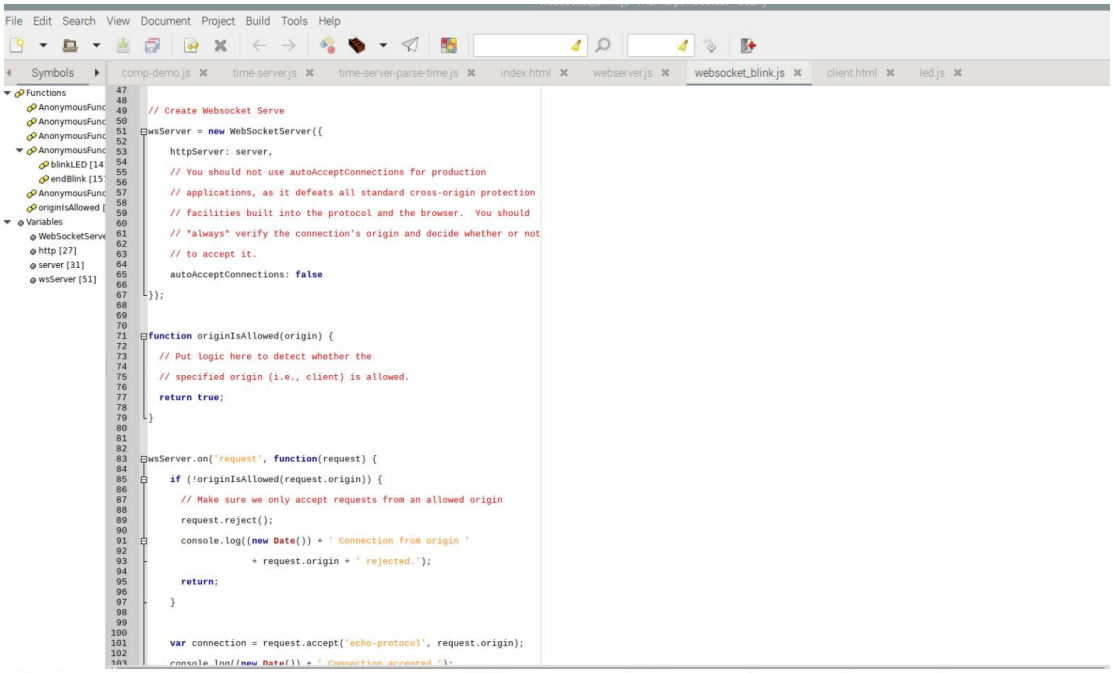
```
if (!originIsAllowed(request.origin)) {
  // Make sure we only accept requests from an allowed origin
  request.reject();
  console.log((new Date()) + ' Connection from origin '
              + request.origin + ' rejected.');
  return;
}

var connection = request.accept('echo-protocol', request.origin);
console.log((new Date()) + ' Connection accepted.');

////////////////////////////////////////////////////
// Case 1: rerceive message from the client
////////////////////////////////////////////////////
connection.on('message', function(message) {
    if (message.type === 'utf8') {
        console.log('Received Message: ' + message.utf8Data);
        connection.sendUTF(message.utf8Data);
    }
    else if (message.type === 'binary') {
        console.log('Received Binary Message of '
                    + message.binaryData.length + ' bytes');
```

```
            connection.sendBytes(message.binaryData);

        }

        var Gpio = require('onoff').Gpio; //include onoff to interact with the
GPIO

        var LED = new Gpio(4, 'out'); //use GPIO pin 4, and specify that it is
output

        var blinkInterval = setInterval(blinkLED, 250); //run the blinkLED
function every 250ms


        function blinkLED() { //function to start blinking

          if (LED.readSync() === 0) { //check the pin state, if the state is 0
(or off)

            LED.writeSync(1); //set pin state to 1 (turn LED on)

          } else {

            LED.writeSync(0); //set pin state to 0 (turn LED off)

          }

        }


        function endBlink() { //function to stop blinking

          clearInterval(blinkInterval); // Stop blink intervals

          LED.writeSync(0); // Turn LED off

          LED.unexport(); // Unexport GPIO to free resources

        }
```

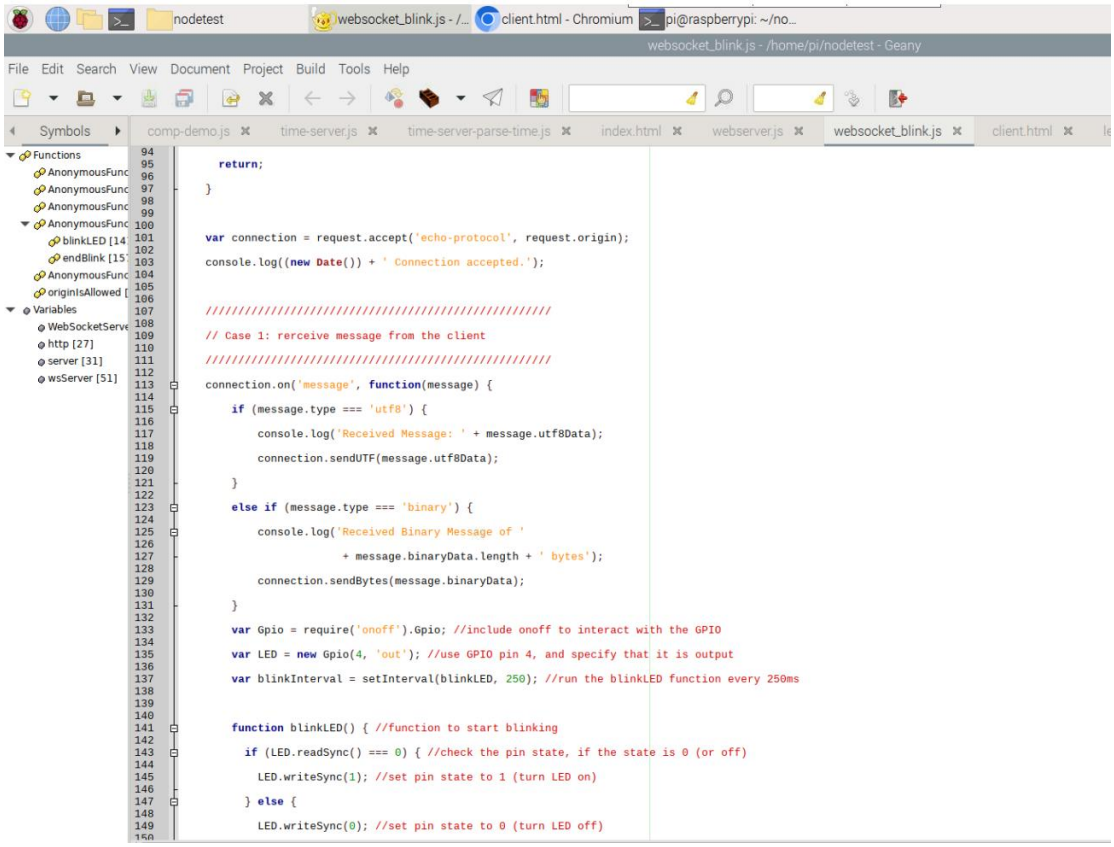
```
        setTimeout(endBlink, 5000); //stop blinking after 5 seconds
    });


    ////////////////////////////////////////////////////
    // Case 2: close the connection
    ////////////////////////////////////////////////////
    connection.on('close', function(reasonCode, description) {
        console.log((new Date()) + ' Peer '
                + connection.remoteAddress + ' disconnected.');
    });
});
```

Screenshot:

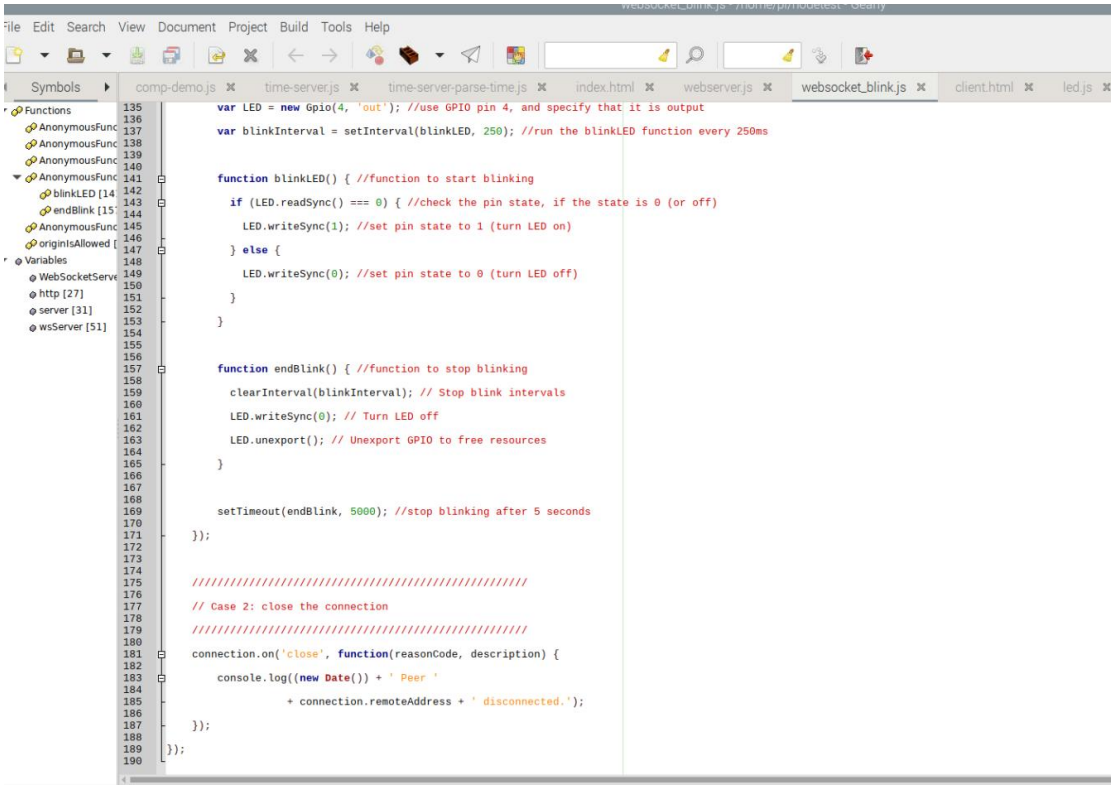




```
// Create Websocket Serve
wsServer = new WebSocketServer({
    httpServer: server,
    // You should not use autoAcceptConnections for production
    // applications, as it defeats all standard cross-origin protection
    // facilities built into the protocol and the browser.  You should
    // *always* verify the connection's origin and decide whether or not
    // to accept it.
    autoAcceptConnections: false
});

function originIsAllowed(origin) {
  // Put logic here to detect whether the
  // specified origin (i.e., client) is allowed.
  return true;
}

wsServer.on('request', function(request) {
    if (!originIsAllowed(request.origin)) {
      // Make sure we only accept requests from an allowed origin
      request.reject();
      console.log((new Date()) + ' Connection from origin '
                  + request.origin + ' rejected.');
      return;
    }

    var connection = request.accept('echo-protocol', request.origin);
    console.log((new Date()) + ' Connection accepted.');
```



```
      return;
    }

    var connection = request.accept('echo-protocol', request.origin);
    console.log((new Date()) + ' Connection accepted.');

    ////////////////////////////////////////////////////
    // Case 1: rerceive message from the client
    ////////////////////////////////////////////////////
    connection.on('message', function(message) {
        if (message.type === 'utf8') {
            console.log('Received Message: ' + message.utf8Data);
            connection.sendUTF(message.utf8Data);
        }
        else if (message.type === 'binary') {
            console.log('Received Binary Message of '
                        + message.binaryData.length + ' bytes');
            connection.sendBytes(message.binaryData);
        }
        var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO
        var LED = new Gpio(4, 'out'); //use GPIO pin 4, and specify that it is output
        var blinkInterval = setInterval(blinkLED, 250); //run the blinkLED function every 250ms


        function blinkLED() { //function to start blinking
          if (LED.readSync() === 0) { //check the pin state, if the state is 0 (or off)
            LED.writeSync(1); //set pin state to 1 (turn LED on)
          } else {
            LED.writeSync(0); //set pin state to 0 (turn LED off)
```



client.html

```
<!DOCTYPE HTML>

<html>

<head>

<script type="text/javascript">


function WebSocketTest()

{

 if ("WebSocket" in window)

 {

    alert("WebSocket is supported by your Browser!");

    // Let us open a web socket
```

```
    // - Error if use this line
    //     var ws = new WebSocket("ws://localhost:8080");
    // - Use this line if the browser would like to communicate with
    //   the server where client.html is downloaded.
    //     var ws = new WebSocket("ws://" + location.host, 'echo-protocol');
    //   Refer Zhomart Mukhamejanov's example
    // - Websocket allows connection from any source, but first
    //   connection should be http request, they call it
    //   "Websocket handshake". For example, you can access
    //      http://npu-socket.herokuapp.com/
    //   then it is possible to write like this
    //       var ws = new WebSocket("wss://npu-socket.herokuapp.com",
'echo-protocol');
    var ws = new WebSocket("ws://localhost:8080", 'echo-protocol');


    // - The readonly attribute readyState represents the state
    //   of the connection. It can have the following values:
    //   + A value of 0 indicates that the connection has
    //     not yet been established.
    //   + A value of 1 indicates that the connection is
    //     established and communication is possible.
    //   + A value of 2 indicates that the connection is going
```

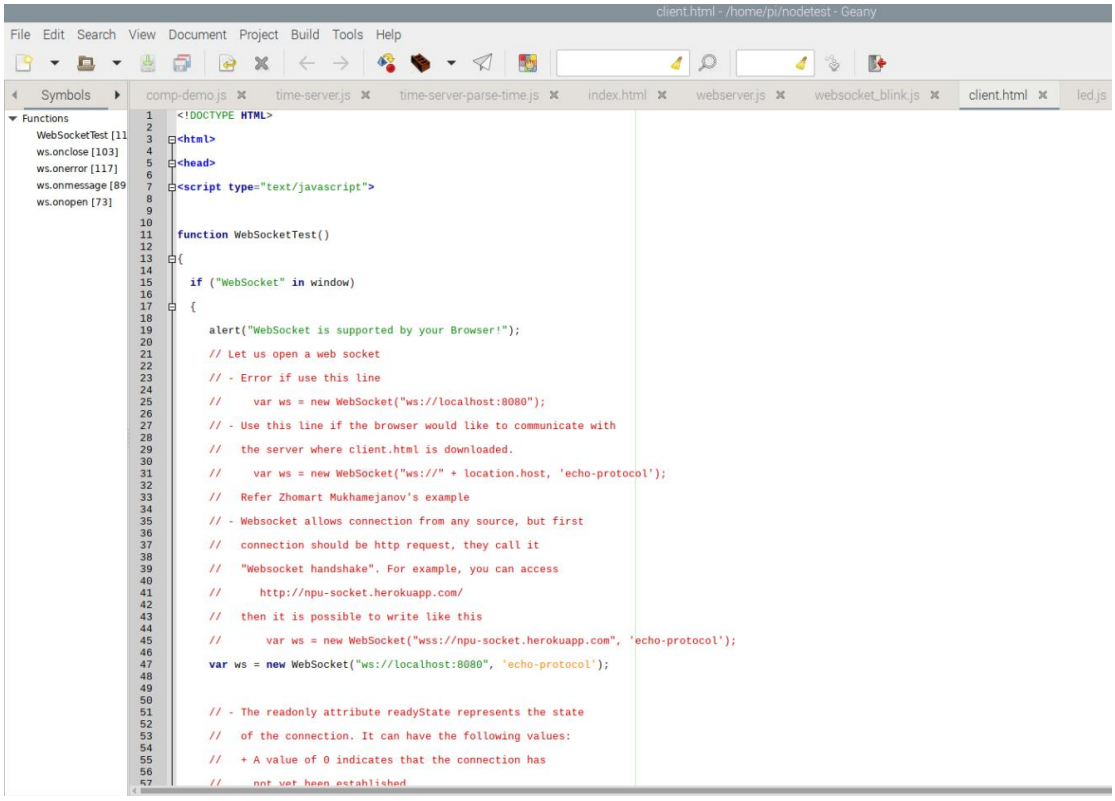
```
//     through the closing handshake.
//   + A value of 3 indicates that the connection has been
//     closed or could not be opened.
// - The open event occurs when socket connection is established.
ws.onopen = function()
{
   // Web Socket is connected, send data using send()
   ws.send("Please blink LED...");
   alert("Message is sent...");
};

// The message event occurs when client receives data from server.
ws.onmessage = function (evt)
{
   var received_msg = evt.data;
   alert("Message is received...");
};

// The close event occurs when connection is closed.
ws.onclose = function()
{
```

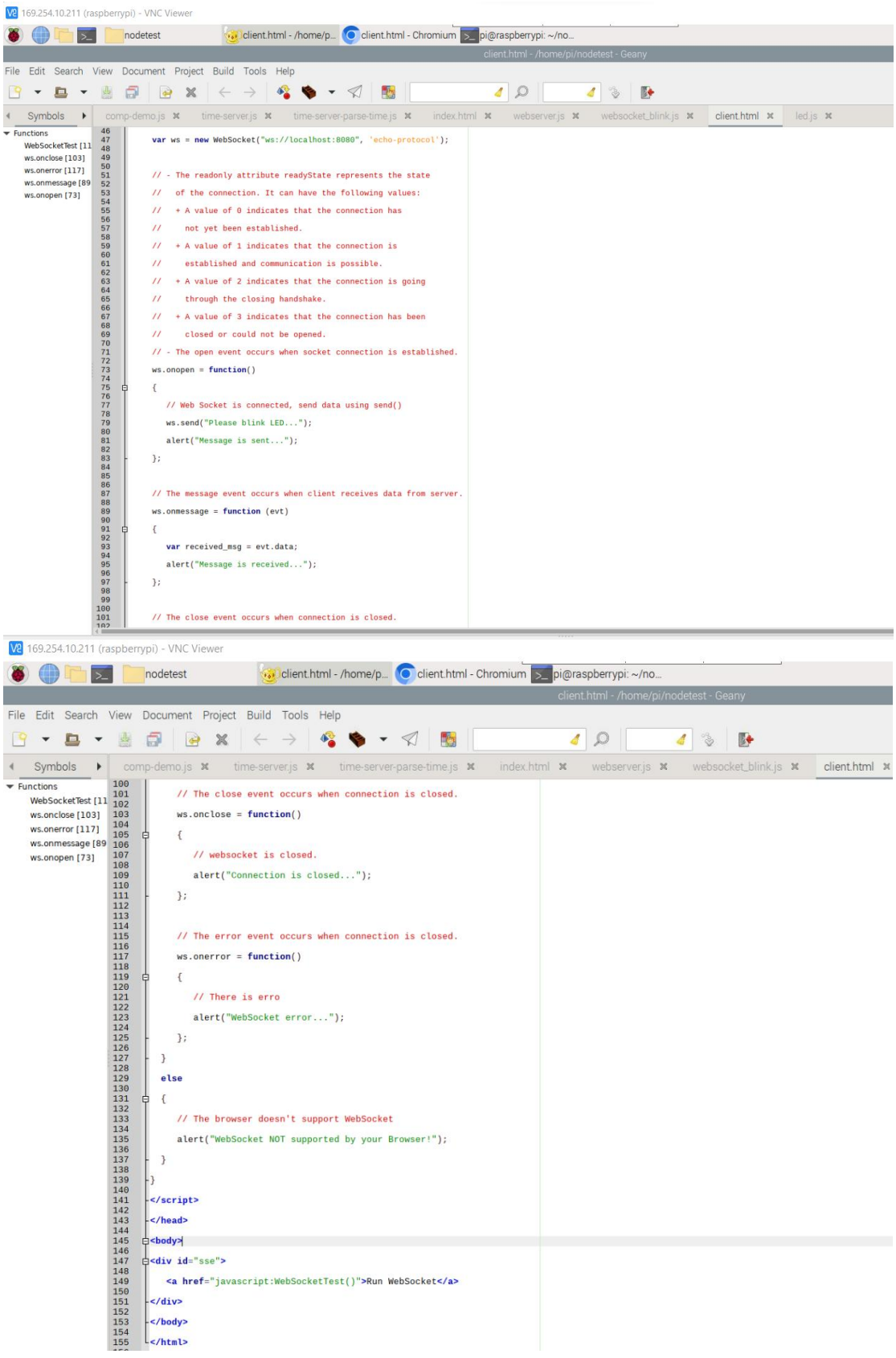
```
      // websocket is closed.
      alert("Connection is closed...");
    };

    // The error event occurs when connection is closed.
    ws.onerror = function()
    {
      // There is erro
      alert("WebSocket error...");
    };
  }
  else
  {
    // The browser doesn't support WebSocket
    alert("WebSocket NOT supported by your Browser!");
  }
}
</script>
</head>
<body>
<div id="sse">
  <a href="javascript:WebSocketTest()">Run WebSocket</a>
```

</body>

</html>

Screenshot:

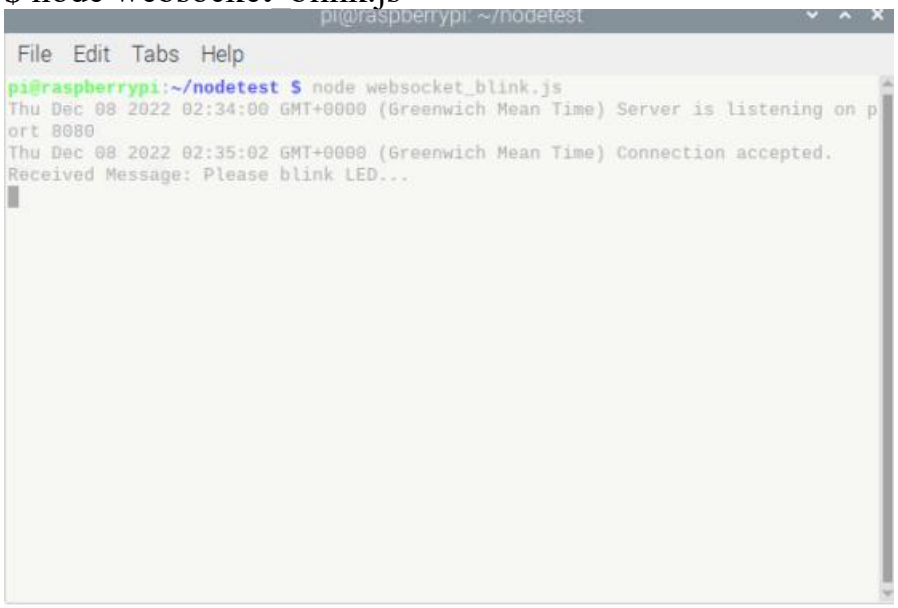




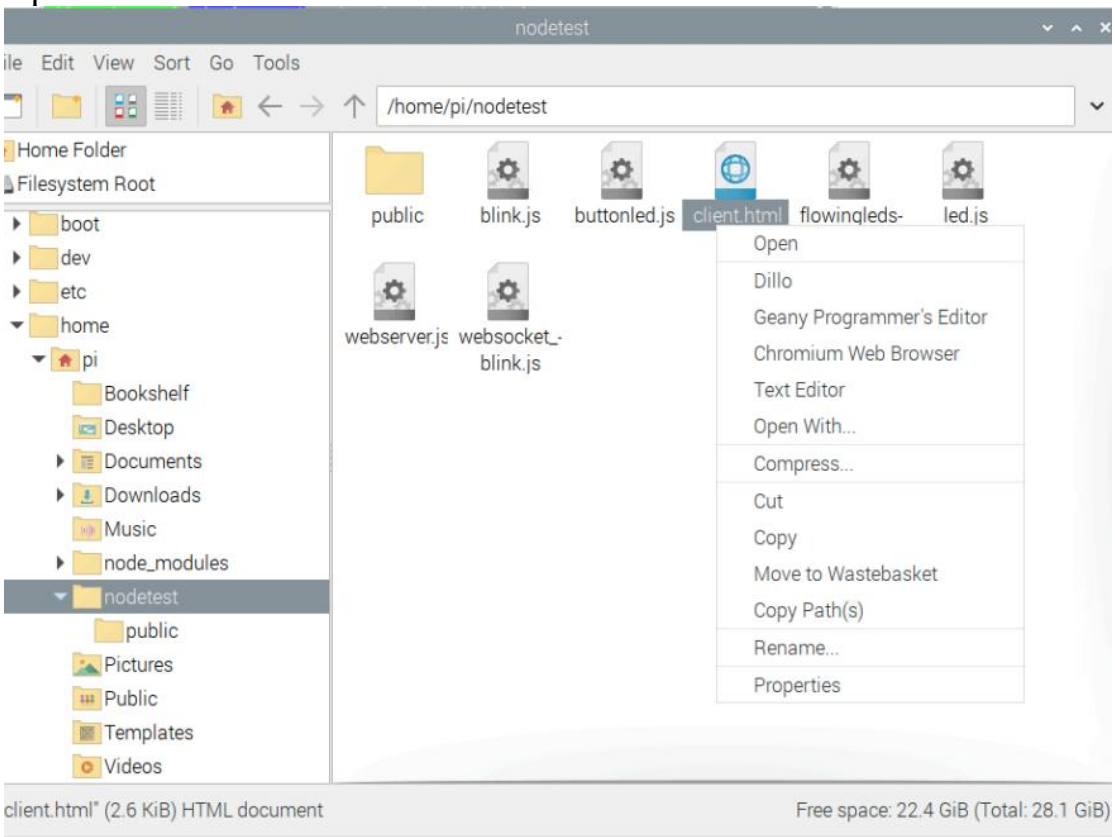
Step 3: Run codes

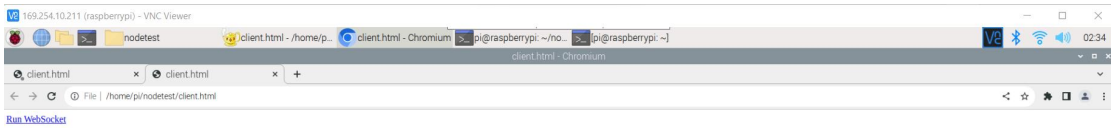
$ cd nodetest

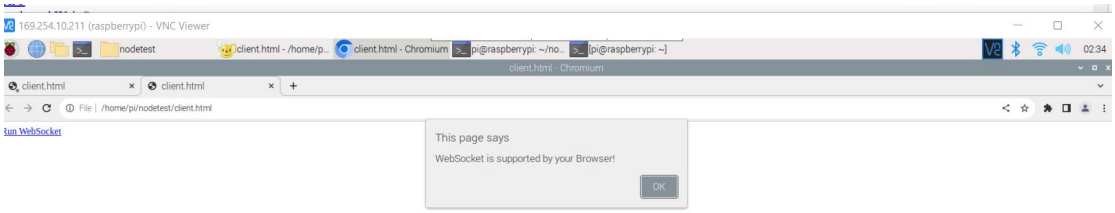
$ node websocket_blink.js

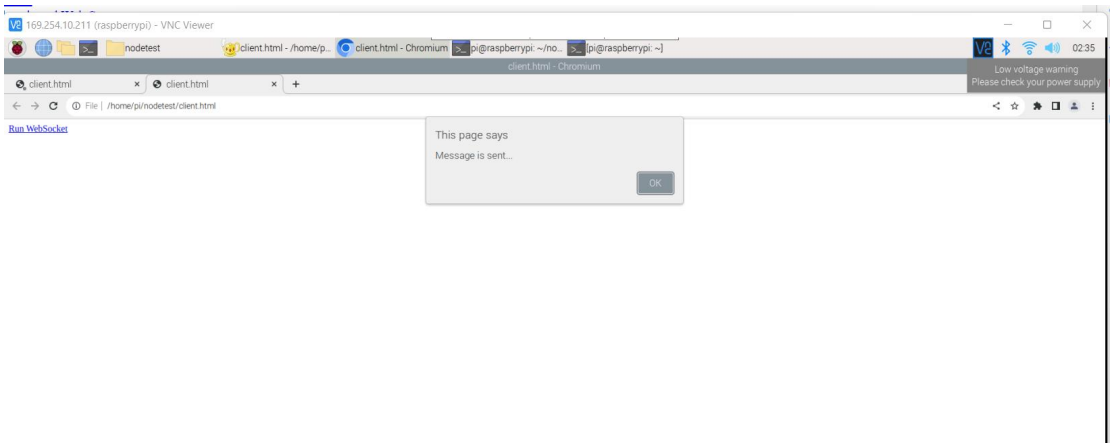


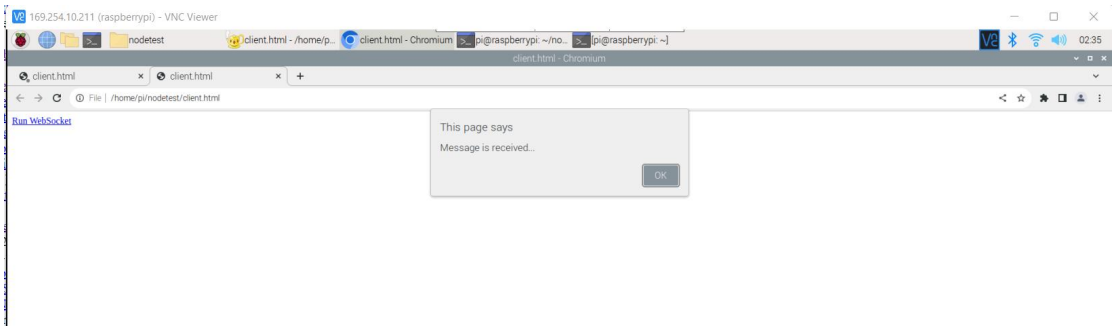
Open the client.html file:

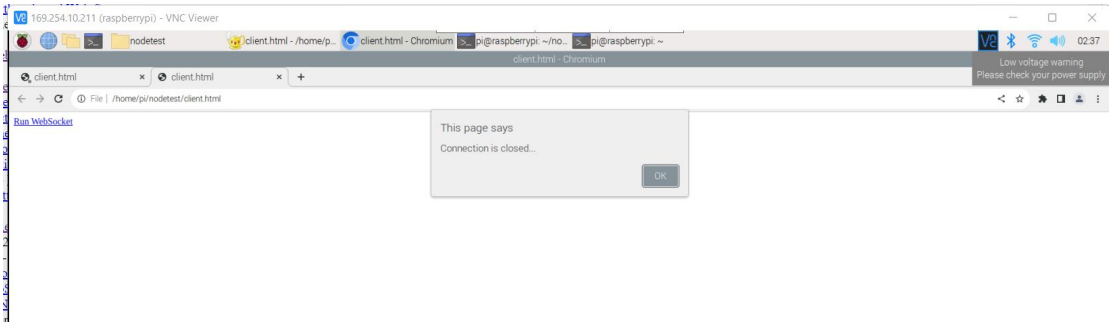



169.254.10.211 (raspberrypi) - VNC Viewer
nodetest
client.html - /home/p...
client.html - Chromium
pi@raspberrypi: ~/no...
pi@raspberrypi: ~]
02:34
client.html - Chromium
client.html
client.html
File | /home/pi/nodetest/client.html
Run WebSocket

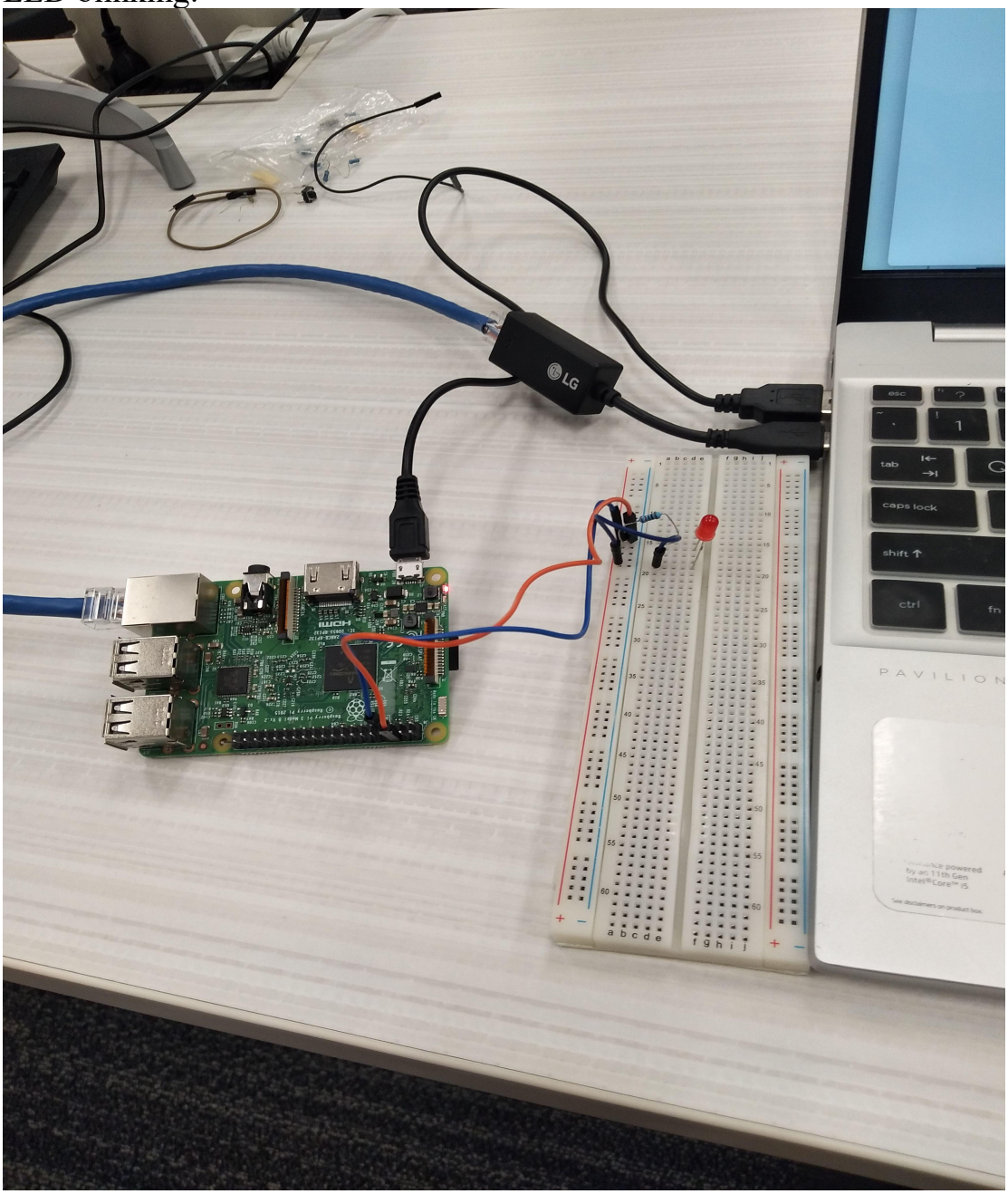

169.254.10.211 (raspberrypi) - VNC Viewer
nodetest
client.html - /home/p...
client.html - Chromium
pi@raspberrypi: ~/no...
pi@raspberrypi: ~]
02:34
client.html - Chromium
client.html
client.html
File | /home/pi/nodetest/client.html
Run WebSocket
This page says
WebSocket is supported by your Browser!
OK

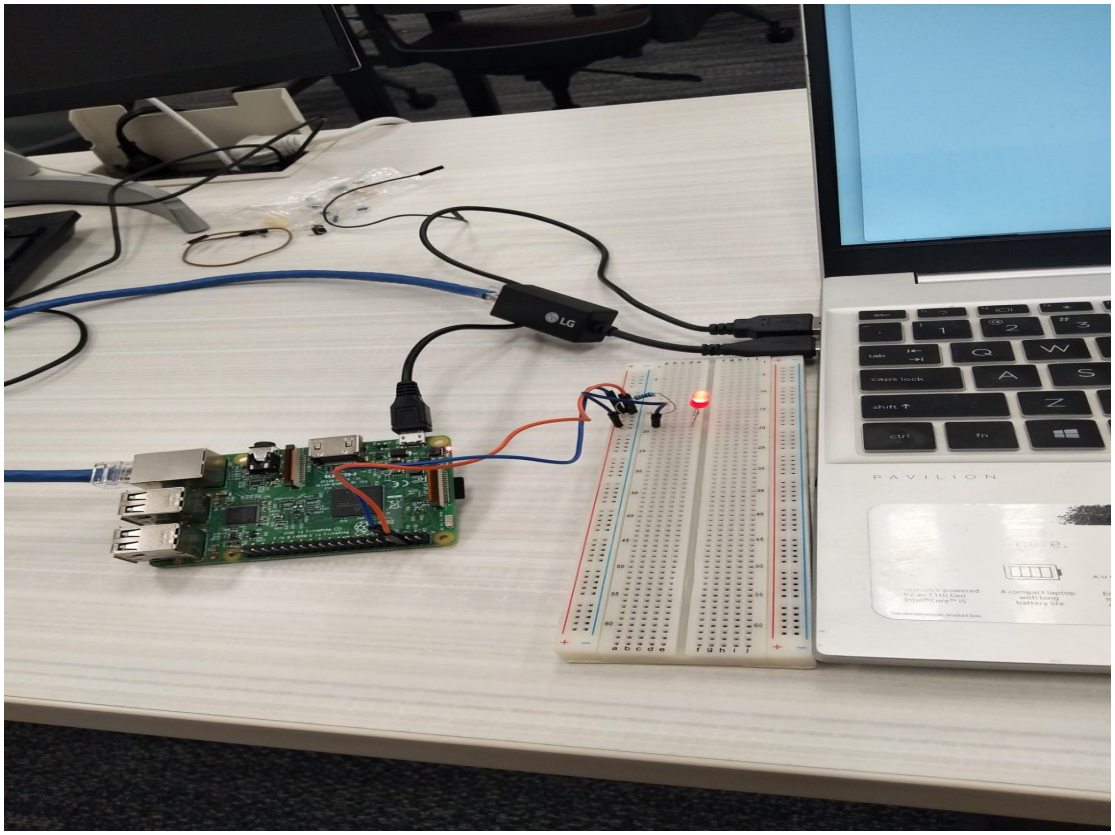

169.254.10.211 (raspberrypi) - VNC Viewer
nodetest
client.html - /home/p...
client.html - Chromium
pi@raspberrypi: ~/no...
pi@raspberrypi: ~]
02:35
Low voltage warning
Please check your power supply
client.html - Chromium
client.html
client.html
File | /home/pi/nodetest/client.html
Run WebSocket
This page says
Message is sent...
OK


169.254.10.211 (raspberrypi) - VNC Viewer
nodetest
client.html - /home/p...
client.html - Chromium
pi@raspberrypi: ~/no...
pi@raspberrypi: ~]
02:35
client.html - Chromium
client.html
client.html
File | /home/pi/nodetest/client.html
Run WebSocket
This page says
Message is received...
OK



LED blinking:




LG
PAVILION