# MicroPython and WiFi part 3: Websockets

By jean-claude.feltes@education.lu

Thonny is used as IDE for all examples, see
http://staff.ltam.lu/feljc/electronics/uPython/ESP8266&uPython_01.pdf

For all examples it is assumed that there is a program main.py defining an accesspoint at boot time,
as described here:
http://staff.ltam.lu/feljc/electronics/uPython/uPy_WiFi_01.pdf

I use the MicroWebSrv library by Jean-Christophe Bos:
https://github.com/jczic/MicroWebSrv

## 3.1. Why websockets?

In this document: http://staff.ltam.lu/feljc/electronics/uPython/uPy_WiFi_02.pdf I have described how to switch something on and off over WiFi, using a webpage and a HTTP connection.

It works, but the latency is more than would be tolerated when steering a robot, for example.
This is due to the overhead of the HTTP protocol.

The websockets protocol allows the connection to stay open all the time, and there is very little overhead. Thus, the connection is less secure, but much faster.

This tutorial: https://www.tutorialspoint.com/websockets/index.htm helped me a lot.

## 3.2. A simple test program sending and receiving messages

With the MicroWebSrv library comes a test program (eventually named main.py) that already allows a test of the websocket communication.

I have reduced it to the minimum, allowing only text based communication.

As it is more interesting to control some hardware, and maybe even easier to understand, I have banned this example to the appendix.

## 3.3. Switching a LED via WiFi

Hardware: a LED connected to IO12

Software: two files

- websocket_LED.py in the root folder of the ESP
- led.html in the www folder of the ESP

ESP32 Python program websocket_LED.py:

```
from microWebSrv import MicroWebSrv
from machine import Pin
d12 = Pin(12, Pin.OUT)

# ----------------------------------------------------------------------------

def _acceptWebSocketCallback(webSocket, httpClient) :
    print("WS ACCEPT")
    webSocket.RecvTextCallback   = _recvTextCallback
    ## webSocket.RecvBinaryCallback = _recvBinaryCallback
    webSocket.ClosedCallback     = _closedCallback

def _recvTextCallback(webSocket, msg) :
    if msg == "LEDon":
        d12.on()
    elif msg == "LEDoff":
        d12.off()
    else:
        print('*')

    print("WS RECV TEXT : %s" % msg)
    webSocket.SendText("Reply for %s" % msg)



def _closedCallback(webSocket) :
    print("WS CLOSED")

# ----------------------------------------------------------------------------
import time

if __name__ == "__main__":

    print("Preparing server")
    srv = MicroWebSrv(webPath='www/')
    srv.MaxWebSocketRecvLen     = 256
    srv.WebSocketThreaded       = True
    srv.AcceptWebSocketCallback = _acceptWebSocketCallback
    print("Starting server")
    srv.Start(threaded = True)

    while True:
        print("*", end = '')
        time.sleep(2)
```

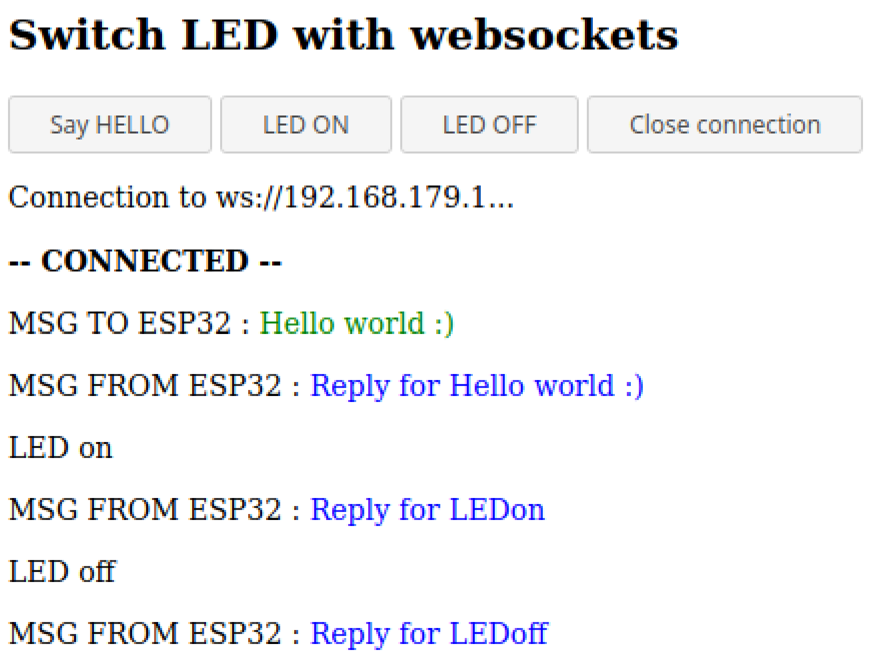
led.html in folder www:

```
<!DOCTYPE html>

<html>

    <head>
        <meta charset="utf-8" />
        <title>MicroWebSocket Switch LED</title>
        <meta http-equiv="Pragma" content="no-cache">
```

```
<script>

var output;

function init() {
    output = document.getElementById("output");

    var wsUri           = "ws://" + window.location.hostname;
    writeToScreen("Connection to " + wsUri + "...")
    websocket           = new WebSocket(wsUri);
    websocket.onopen    = function(evt) { onOpen    (evt) };
    websocket.onclose   = function(evt) { onClose   (evt) };
    websocket.onmessage = function(evt) { onMessage (evt) };
    websocket.onerror   = function(evt) { onError   (evt) };
    }

function onOpen(evt) {
    writeToScreen("<strong>-- CONNECTED --</strong>");
    SendMsg("Hello world :)");
    }

function onClose(evt) {
    writeToScreen("<strong>-- DISCONNECTED --</strong>");
    }

function onMessage(evt) {
    writeToScreen('MSG FROM ESP32 : <span style="color: blue;">' +
      evt.data + '</span>');
    }

function onError(evt) {
    writeToScreen('ERROR : <span style="color: red;">' +
      evt.data + '</span>');
    }

function SendMsg(msg) {
    writeToScreen('MSG TO ESP32 : <span style="color: green;">' +
      msg + '</span>');
    websocket.send(msg);
    }

function writeToScreen(s) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = s;
    output.appendChild(pre);
    }

window.addEventListener("load", init, false);

function sayHello() {
      writeToScreen("HELLO");
      websocket.send("HELLO from button");
      }

    function LEDon() {
      writeToScreen("LED on");
      websocket.send("LEDon");
      }
```

```
            function LEDoff() {
              writeToScreen("LED off");
              websocket.send("LEDoff");
              }

            function closeconnection() {
              writeToScreen("Closing ...");
              websocket.close();
              }

         </script>
    </head>

    <body>
         <h2>Switch LED with websockets</h2>
         <button type="button" onClick = "sayHello()">Say HELLO</button>
         <button type="button" onClick = "LEDon()">LED ON</button>
         <button type="button" onClick = "LEDoff()">LED OFF</button>
         <button type="button" onClick ="closeconnection()">Close connection</button>
         <div id="output"></div>

    </body>
</html>
```

Testing with the browser with the URL 192.168.179.1/led.html:
(supposed the IP of the ESP is  192.168.179.1)



**Switch LED with websockets**

Connection to ws://192.168.179.1...

**-- CONNECTED --**

MSG TO ESP32 : Hello world :)

MSG FROM ESP32 : Reply for Hello world :)

LED on

MSG FROM ESP32 : Reply for LEDon

LED off

MSG FROM ESP32 : Reply for LEDoff

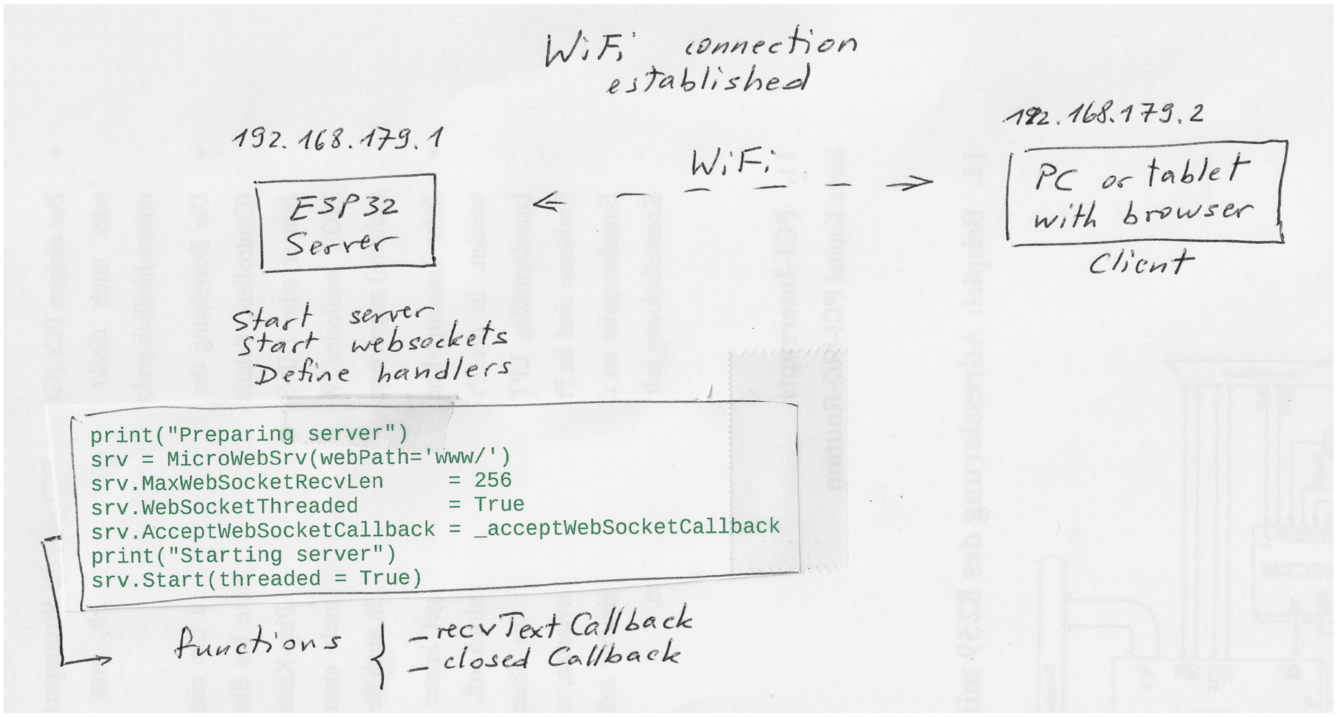
## 3.4. How does it work?

(Switching the LED)

As the software is executed partially on the server and partially on the client, the interaction is somewhat complex, and it took me a while to understand the details. What helped a lot here was a listing, paper, scissors, a pen and glue (really old fashioned!).

In all the following sketches the server is on the left side (listings in green) and the client is on the right side (listings in blue).

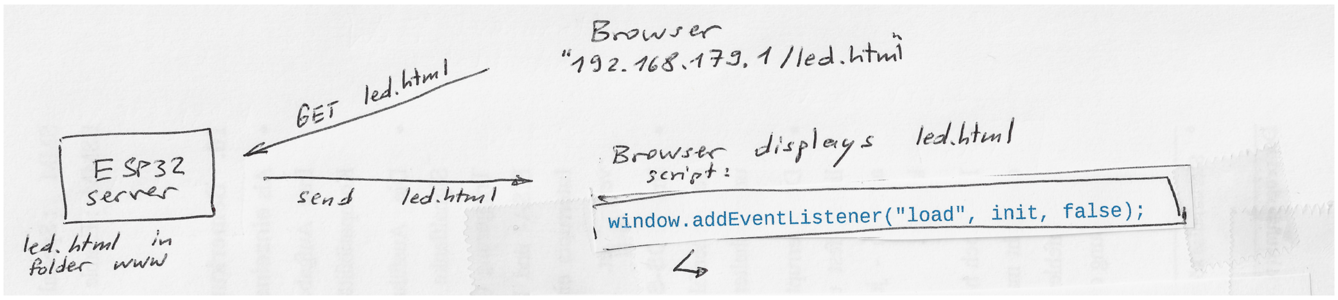
We start with a working WiFi connection and see what happens on the ESP server:



```
print("Preparing server")
srv = MicroWebSrv(webPath='www/')
srv.MaxWebSocketRecvLen     = 256
srv.WebSocketThreaded       = True
srv.AcceptWebSocketCallback = _acceptWebSocketCallback
print("Starting server")
srv.Start(threaded = True)
```

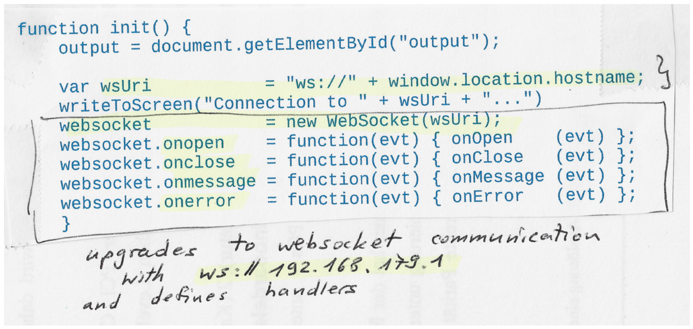
The server and the websockets are started and handlers for incoming messages (and for close) are defined. Now the server is ready and waiting for incoming connections.

Now on a remote PC or tablet (the client), a browser tries to open the webpage “192.168.179.1/led.html” in the www folder of the ESP:



```
window.addEventListener("load", init, false);
```

The browser displays the webpage and, as there is an event listener for the load event, calls the init function:

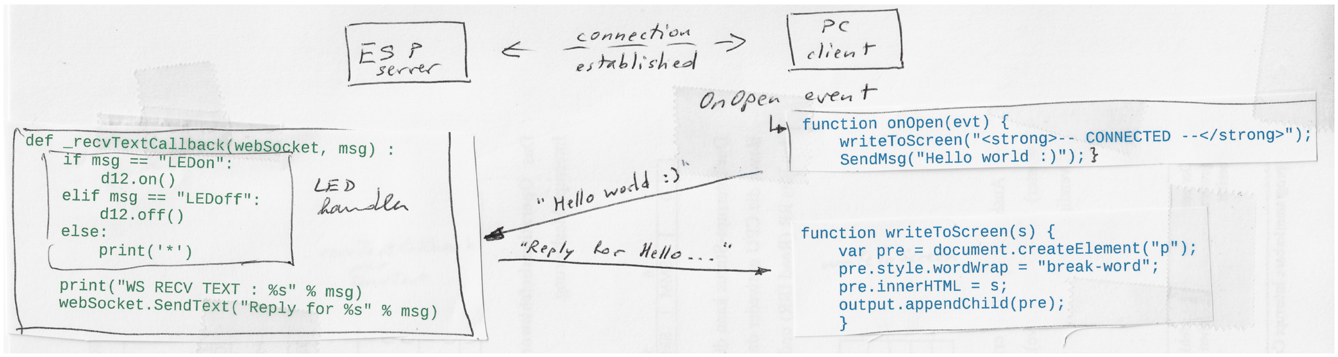


```
function init() {
    output = document.getElementById("output");

    var wsUri              = "ws://" + window.location.hostname;
    writeToScreen("Connection to " + wsUri + "...")
    websocket              = new WebSocket(wsUri);
    websocket.onopen       = function(evt) { onOpen    (evt) };
    websocket.onclose      = function(evt) { onClose   (evt) };
    websocket.onmessage    = function(evt) { onMessage (evt) };
    websocket.onerror      = function(evt) { onError   (evt) };
    }
```

upgrades to websocket communication
with ws:// 192.168.179.1
and defines handlers

This function upgrades the HTTP connection to a websocket connection and defines handlers for open, close, errors, and the most important, for an incoming message.

Once the connection is ready, the OnOpen event is fired on the client side and a message is sent to the server, who replies:



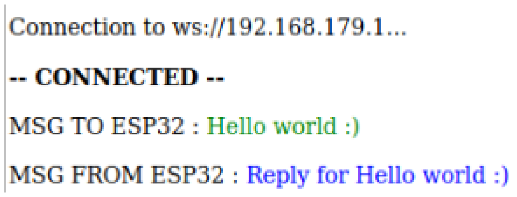
```
def _recvTextCallback(webSocket, msg) :
    if msg == "LEDon":
        d12.on()
    elif msg == "LEDoff":
        d12.off()
    else:
        print('*')

    print("WS RECV TEXT : %s" % msg)
    webSocket.SendText("Reply for %s" % msg)
```

```
function onOpen(evt) {
    writeToScreen("<strong>-- CONNECTED --</strong>");
    SendMsg("Hello world :)"); }
```

```
function writeToScreen(s) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = s;
    output.appendChild(pre);
    }
```

The function writeToScreen is a helper function that it is called by the onMessage event. It displays the message in the browser window.

All that is described above  happens automatically and gives this result in the browser:



Connection to ws://192.168.179.1...

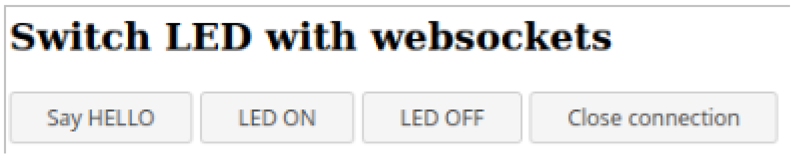
**-- CONNECTED --**

MSG TO ESP32 : Hello world :)

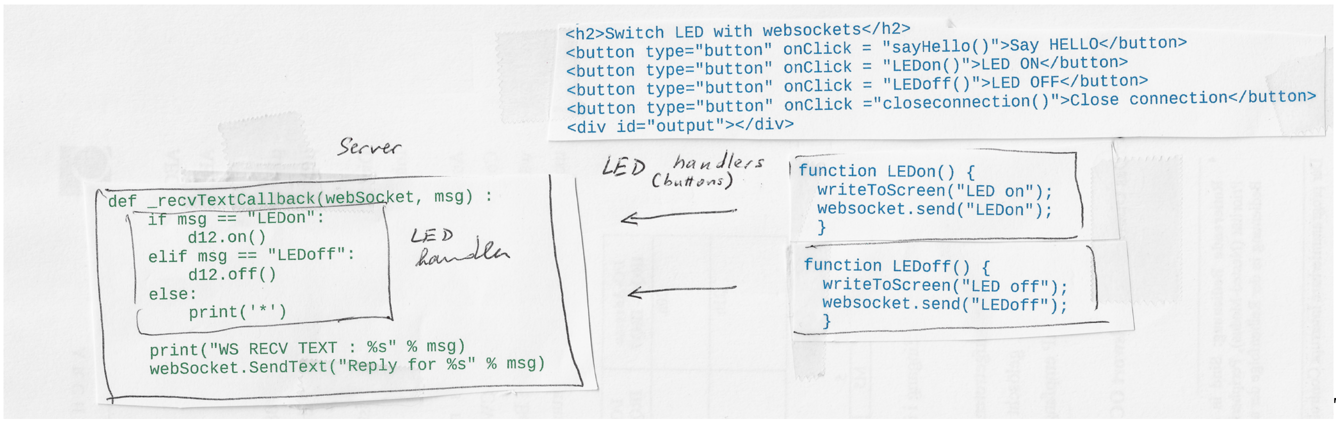
MSG FROM ESP32 : Reply for Hello world :)

User action:

In the HTML file we have defined some buttons and the corresponding handler functions:



**Switch LED with websockets**



```
<h2>Switch LED with websockets</h2>
<button type="button" onClick = "sayHello()">Say HELLO</button>
<button type="button" onClick = "LEDon()">LED ON</button>
<button type="button" onClick = "LEDoff()">LED OFF</button>
<button type="button" onClick ="closeconnection()">Close connection</button>
<div id="output"></div>
```

```
def _recvTextCallback(webSocket, msg) :
    if msg == "LEDon":
        d12.on()
    elif msg == "LEDoff":
        d12.off()
    else:
        print('*')

    print("WS RECV TEXT : %s" % msg)
    webSocket.SendText("Reply for %s" % msg)
```

```
function LEDon() {
  writeToScreen("LED on");
  websocket.send("LEDon");
 }
```

```
function LEDoff() {
  writeToScreen("LED off");
  websocket.send("LEDoff");
 }
```

Th

The most important for us are the LED buttons. When a LED button is pushed, the message “LEDon” or “LEDoff” is sent to the server:

The server replies with “Reply for LED on” or “Reply for LED off” to the client.

When the user presses the “Close connection” button, the websocket connection is closed on the client side. The server reacts to this event with the _closedCallback function.

## 3.5. In case of problems...

There are a certain number of things that can go wrong for example:

- You have corrected a bug, but the result is the same as before.
  Maybe you have used the HTML file that is cached.
  Refresh the browser view.

  You are not using the HTML file saved to the ESP device, but the one saved to your computer.
  Be sure to save “to Micropython device”.
  In Firefox you can see the source code under “Developer options” and so verify if you have the right version.
- There is a bug in the HTML file.
  In this case eventually a previous version is loaded and you are wondering what happens.

  In Firefox a look at the Web console shows errors.
  There are also online validators for the code that you can use, for example
  https://validator.w3.org/
  https://www.freeformatter.com/html-validator.html

- The MicroWebSocket.py module must be present in the root folder!

In summary, if anything goes wrong, go to “Developer” - “Page source” in the browser.
Here you see what’s actually in the HTML  the browser has loaded.

I have experienced a strange behavior. Sometimes when the page was refreshed it did not work, but after looking at the source code and then refreshing the web page it worked.

## 3.6. Appendix

A simple example of communication over websockets:

The ESP server is contacted over HTTP by the remote browser, sends the file wstest.html which starts a conversation.

```
from microWebSrv import MicroWebSrv

# ----------------------------------------------------------------------------

def _acceptWebSocketCallback(webSocket, httpClient) :
    print("WS ACCEPT")
    webSocket.RecvTextCallback   = _recvTextCallback
    webSocket.ClosedCallback     = _closedCallback

def _recvTextCallback(webSocket, msg) :
    print("WS RECV TEXT : %s" % msg)
    webSocket.SendText("Reply for %s" % msg)

def _closedCallback(webSocket) :
    print("WS CLOSED")

# ----------------------------------------------------------------------------
import time

if __name__ == "__main__":

    print("Preparing server")
    srv = MicroWebSrv(webPath='www/')
    srv.MaxWebSocketRecvLen     = 256
    #srv.WebSocketThreaded       = True
    srv.AcceptWebSocketCallback = _acceptWebSocketCallback
    print("Starting server")
    srv.Start()
```

The program defines callback handlers for received text messages and a close handler.

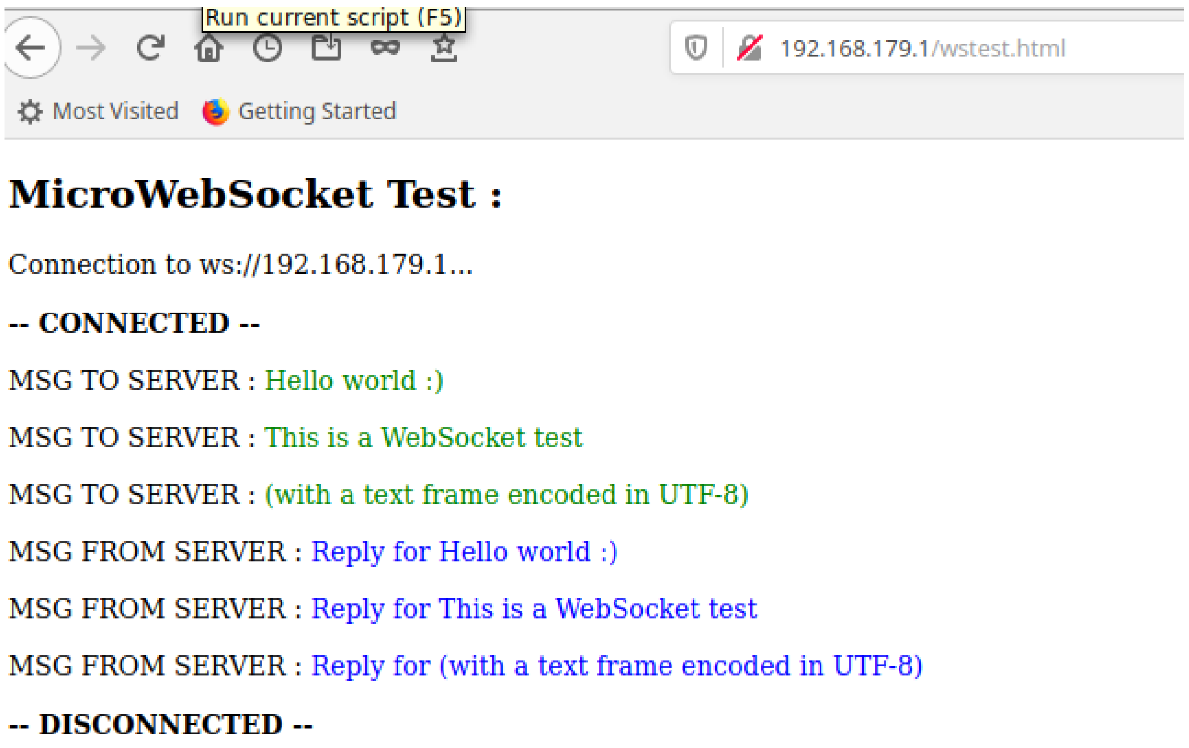
In the www folder of the ESP, we have a HTML file that contains a Javascript part to handle the Websocket communication:

wstest.html:

```
<!DOCTYPE html>

<html>

    <head>
        <meta charset="utf-8" />
        <title>MicroWebSocket Test 2</title>
```

```
</head>

<script language="javascript">

    var output;

    function init()
    {
        output = document.getElementById("output");

        var wsUri            = "ws://" + window.location.hostname;
        writeToScreen("Connection to " + wsUri + "...")
        websocket            = new WebSocket(wsUri);
        websocket.onopen    = function(evt) { onOpen    (evt) };
        websocket.onclose   = function(evt) { onClose   (evt) };
        websocket.onmessage = function(evt) { onMessage (evt) };
        websocket.onerror   = function(evt) { onError   (evt) };
    }

    function onOpen(evt)
    {
        writeToScreen("<strong>-- CONNECTED --</strong>");
        SendMsg("Hello world :)");
        setTimeout( function() { websocket.close() }, 5000 )
    }

    function onClose(evt)
    {
        writeToScreen("<strong>-- DISCONNECTED --</strong>");
    }

    function onMessage(evt)
    {
        writeToScreen('MSG FROM ESP32 : <span style="color: blue;">' +
          evt.data + '</span>');
    }

    function onError(evt)
    {
        writeToScreen('ERROR : <span style="color: red;">' + evt.data +
      '</span>');
    }

    function SendMsg(msg)
    {
        writeToScreen('MSG TO ESP32 : <span style="color: green;">' +
          msg + '</span>');
        websocket.send(msg);
    }

    function writeToScreen(s)
    {
        var pre = document.createElement("p");
        pre.style.wordWrap = "break-word";
        pre.innerHTML = s;
        output.appendChild(pre);
    }

    window.addEventListener("load", init, false);
```

```
</script>

<body>
  <h2>MicroWebSocket Test :</h2>
  <div id="output"></div>
</body>
```



## MicroWebSocket Test :

```
Connection to ws://192.168.179.1...

-- CONNECTED --

MSG TO SERVER : Hello world :)

MSG TO SERVER : This is a WebSocket test

MSG TO SERVER : (with a text frame encoded in UTF-8)

MSG FROM SERVER : Reply for Hello world :)

MSG FROM SERVER : Reply for This is a WebSocket test

MSG FROM SERVER : Reply for (with a text frame encoded in UTF-8)

-- DISCONNECTED --
```