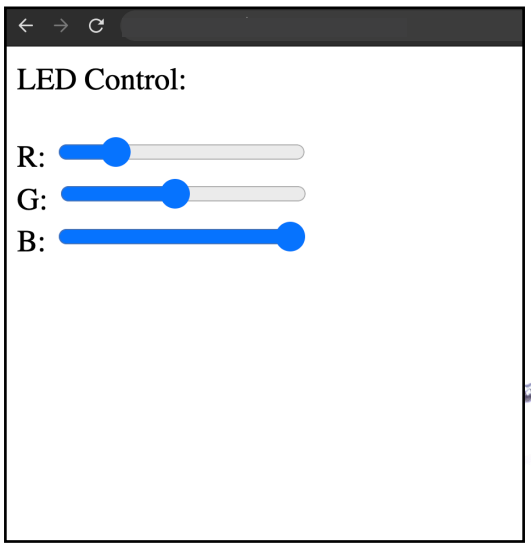
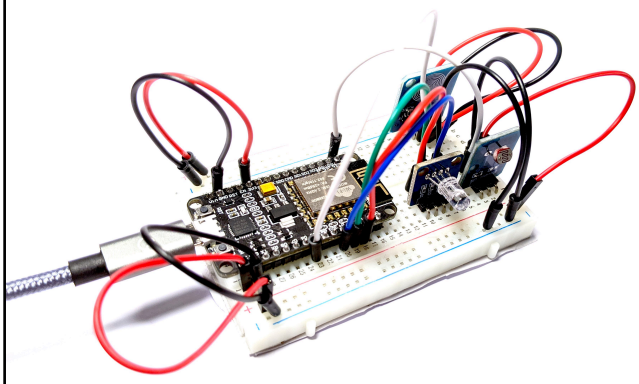
Web-sockets

An additional communication protocol

Project description:

In this project, you will examine one additional communication protocol: web sockets. Precisely, you will first explore the Arduino IDE web socket example sketch to control an RGB led component, then you will adapt the example sketch code to other components of your choice. There is no need for you to build a separate circuit for this exercise. You will use the smart chair circuit that you have built during the course and upload a different code sketch.



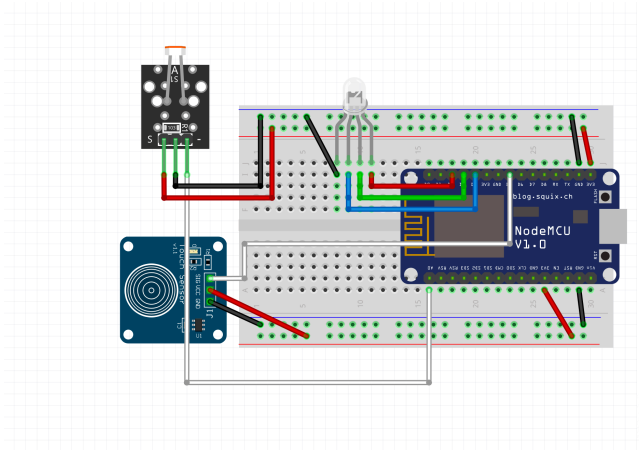


Project objectives:

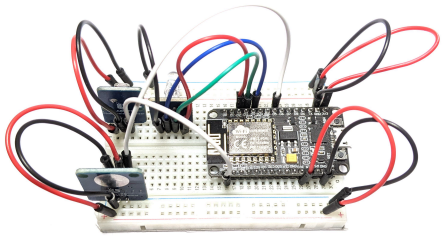
- Modify the smart chair circuit behaviour with alternative code
- Explore the Arduino IDE web socket example sketch
- Adapt the web socket example code to other components

Project components:

Component Reference	Component Quantity	Component Name	Component Description
A	1	Smart Chair circuit	The smart chair circuit which features the photoresistor, the RGB LED, and the touch sensor
B	1	Micro USB Cable	A USB cable to power and upload instructions to a microcontroller



A



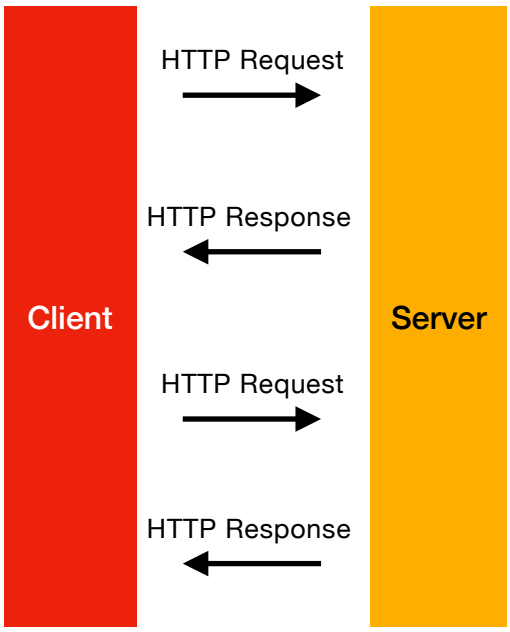
B

Step One
Project Introduction

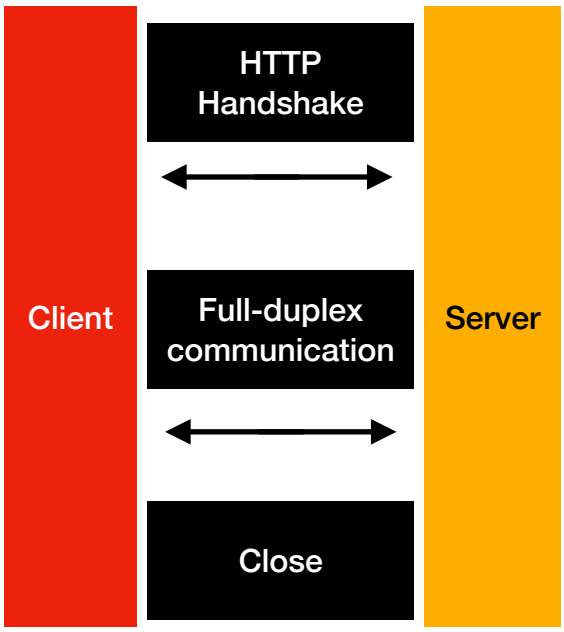
The idea behind this exercise is for you to explore how web sockets can serve as an additional communication protocol to exchange data.

On your previous exercises, you have only used simple HTTP requests to either GET data from the ESP board or to send (POST) data to it via a web server. This always resulted in a browser refresh every time you either wanted to send or receive data. Would it be great if you could exchange data between your ESP and a web server without the need of refreshing the client web page each time?

WebSocket is a protocol that keeps the transmission control protocol connection open, so you can constantly send data back and forth between the ESP and the client without refreshing the webpage. It all happens with an initial HTTP handshake where the client and the server agree to open a connection. At that point a full-duplex persistent connection is established where the client and the server can constantly communicate through messages. The communication will eventually end when there is no longer the need to exchange information between the client and the server.



HTTP basic protocol



HTTP web socket protocol

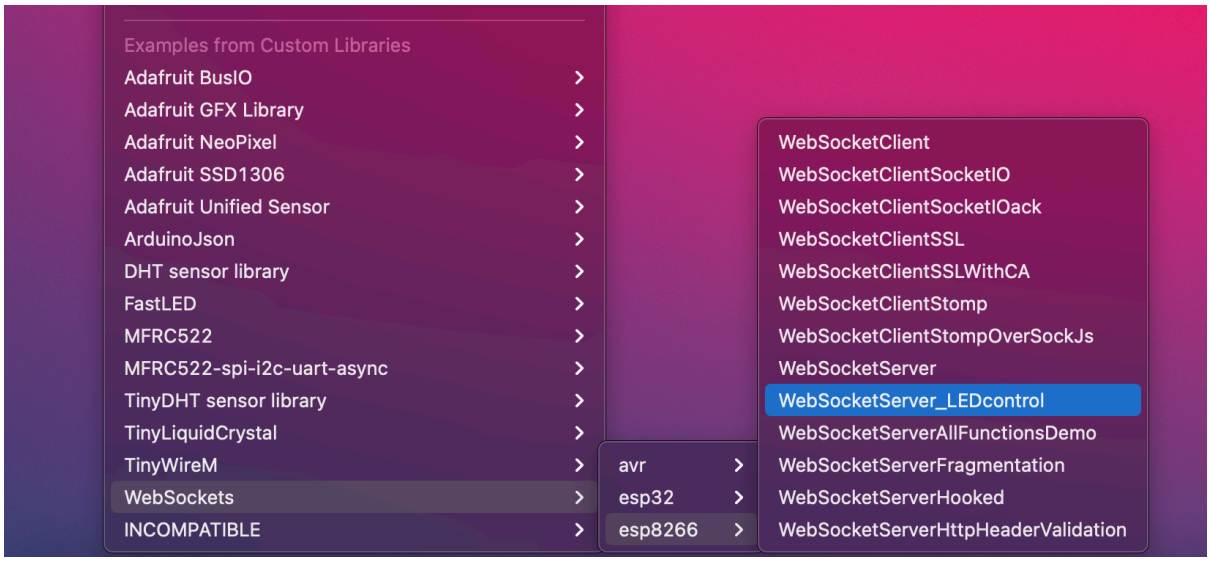
Step Two

Writing the Code

In this section of the exercise, you will start with Arduino IDE web sockets example sketch so that you can quickly see how such protocol can be implemented. The example sketch allows the ESP board to host a simple webpage with three sliders to set the red, green and blue levels of an RGB LED component. The web socket protocol will allow you to manipulate the sliders and update the colours of the RGB component without refreshing the webpage.

Now that you are familiar with the task, it is time to explore the code for the web socket protocol. Go ahead and open the following Arduino sketch example:

File > Examples > WebSockets > esp8266 > WebSocketServer_LEDcontrol



Feel free to save the sketch and rename it to something sensible: **websocket_smat_chair** for instance.

There are three main parts that you need to change before uploading this sketch to your smart chair ESP circuit:

- Change the RGB LED pin references
- Add your WiFi credentials
- Add a serial print to visualise the IP address of the web server

Change the RGB LED pin references:

```
#define LED_RED     D2
#define LED_GREEN   D3
#define LED_BLUE    D4
```

The above change now references the RGB LED pins on your smart chair circuit.

Add your WiFi credentials:

```
WiFiMulti.addAP("[redacted]");
```

Add both your WiFi network name (SSID) and your WiFi password (passpass) to the **addAp** function.

Add a serial print to visualise the IP address of the web server:

```
if(MDNS.begin("esp8266")) {
    USE_SERIAL.println("MDNS responder started");
    //Print the board IP address
    USE_SERIAL.print("IP address: ");
    USE_SERIAL.println(WiFi.localIP());
}
```

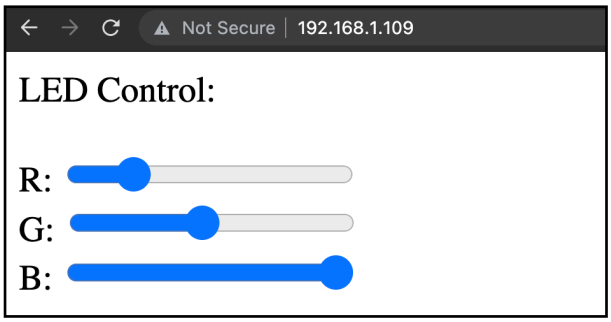
Add the above two lines of code inside the conditional statement. This will allow you to see the web server IP address printed on the Serial Monitor.

Hurray, you have successfully modified the example sketch.
Go ahead, compile your code, and upload it to your ESP smart chair board.

Once the code has been uploaded, open the Serial Monitor and look for the “IP address”. You can then use such address to access the client webpage on your favourite browser.

Note: the Serial Monitor baud rate should be set to 115200.

You should now be able to see the client webpage:



Use the sliders to change the colour of your RGB LED component. You will see that the RGB component updates without the need for the webpage to refresh.

Step Three
Looking at the code

The code should be relatively familiar to the one you wrote for the smart chair and the smart fridge web server exercises. In fact, web sockets are just an extension of the HTTP protocol with the exception that once the connection has been made between the client and the server, there is no need to establish a new TCP connection for every message you send.

Libraries and variables initialisation:

```
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <WebSocketsServer.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <Hash.h>

#define LED_RED     D2
#define LED_GREEN   D3
#define LED_BLUE    D4

#define USE_SERIAL Serial


ESP8266WiFiMulti WiFiMulti;

ESP8266WebServer server(80);
WebSocketsServer webSocket = WebSocketsServer(81);
```

Necessary libraries and packages

Pin references for the RGB LED component

Rename Serial to USE_SERIAL

Run server on PORT 80 and WebSocket on PORT 81

The webSocketEvent() function:

This function is responsible for listening to incoming messages via the web socket protocol. It checks for three main scenarios:

- The web socket disconnection (case WStype_DISCONNECTED)
- The web socket connection (case WStype_CONNECTED)
- The web socket incoming messages (case WStype_TEXT)

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {

    switch(type) {
        case WStype_DISCONNECTED:
            USE_SERIAL.printf("[%u] Disconnected!\n", num);
            break;
        case WStype_CONNECTED: {
            IPAddress ip = webSocket.remoteIP(num);
            USE_SERIAL.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num, ip[0], ip[1], ip[2], ip[3], payload);

            // send message to client
            webSocket.sendTXT(num, "Connected");
        }
            break;
        case WStype_TEXT:
            USE_SERIAL.printf("[%u] get Text: %s\n", num, payload);

            if(payload[0] == '#') {
                // we get RGB data

                // decode rgb data
                uint32_t rgb = (uint32_t) strtol((const char *) &payload[1], NULL, 16);

                analogWrite(LED_RED,    ((rgb >> 16) & 0xFF));
                analogWrite(LED_GREEN,  ((rgb >> 8) & 0xFF));
                analogWrite(LED_BLUE,   ((rgb >> 0) & 0xFF));
            }

            break;
    }

}
```

The “WStype_TEXT” case checks for incoming messages starting with the character “#” (meaning it is an RGB message). If so, the message is decoded and the RGB LED components are updated accordingly via the **analogWrite()** function.

The **setup()** function is mainly responsible to setup the web server, connect the ESP to the home wifi, and initialise the web socket protocol event listener “webSocket.onEvent(webSocketEvent);”. The latter will trigger the **webSocketEvent()** function each time a message is sent through the web socket protocol.

The **server.on()** function is similar to other routing functions that you encountered for the smart chair and smart fridge web server exercises. The main difference is that the HTML page contains some JavaScript code to first connect to the web socket and to then send the #RGB message any time the user changes one of the sliders from the client HTML page served:

var connection = new WebSocket('ws://'+location.hostname+':81/',['arduino']);

connection.send(rgb);

```
void setup() {
    //USE_SERIAL.begin(921600);
    USE_SERIAL.begin(115200);

    //USE_SERIAL.setDebugOutput(true);

    USE_SERIAL.println();
    USE_SERIAL.println();
    USE_SERIAL.println();

    for(uint8_t t = 4; t > 0; t--) {
        USE_SERIAL.printf("[SETUP] BOOT WAIT %d...\n", t);
        USE_SERIAL.flush();
        delay(1000);
    }

    pinMode(LED_RED, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
    pinMode(LED_BLUE, OUTPUT);

    digitalWrite(LED_RED, 1);
    digitalWrite(LED_GREEN, 1);
    digitalWrite(LED_BLUE, 1);

    WiFiMulti.addAP("9BAC Hyperoptic Fibre Broadband", "SGsgeU4ZhWDK");

    while(WiFiMulti.run() != WL_CONNECTED) {
        delay(100);
    }

    // start webSocket server
    webSocket.begin();
    webSocket.onEvent(webSocketEvent);

    // start webSocket server
    webSocket.begin();
    webSocket.onEvent(webSocketEvent);

    if(MDNS.begin("esp8266")) {
        USE_SERIAL.println("MDNS responder started");
        //Print the board IP address
        USE_SERIAL.print("IP address: ");
        USE_SERIAL.println(WiFi.localIP());
    }

    // handle index
    server.on("/", []() {
        // send index.html
        server.send(200, "text/html", "<html><head><script>var connection = new WebSocket('ws://'+l
    });

    server.begin();

    // Add service to MDNS
    MDNS.addService("http", "tcp", 80);
    MDNS.addService("ws", "tcp", 81);

    digitalWrite(LED_RED, 0);
    digitalWrite(LED_GREEN, 0);
    digitalWrite(LED_BLUE, 0);

}
```

Finally the **loop()** function constantly check for web socket events and runs the web server:

```
void loop() {
    webSocket.loop();
    server.handleClient();
}
```

Step Four
Practice with WebSockets

Now that you are familiar with web sockets and also explored one example code, why don't you try to adapt the example sketch code to either a different component of the smart chair circuit or maybe a different circuit?

You can perhaps update one of your early web server sketches and integrate the web socket protocol. What about controlling the buzzer of your smart fridge without refreshing the web page?