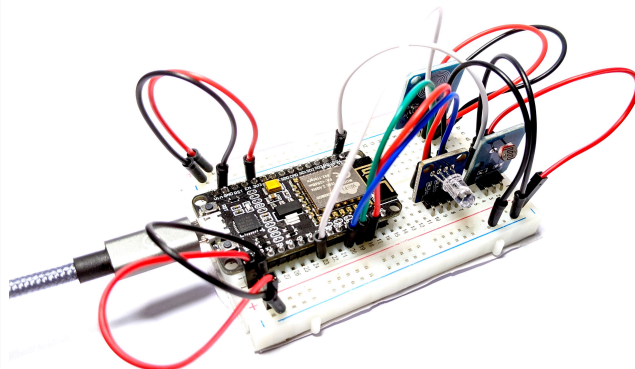
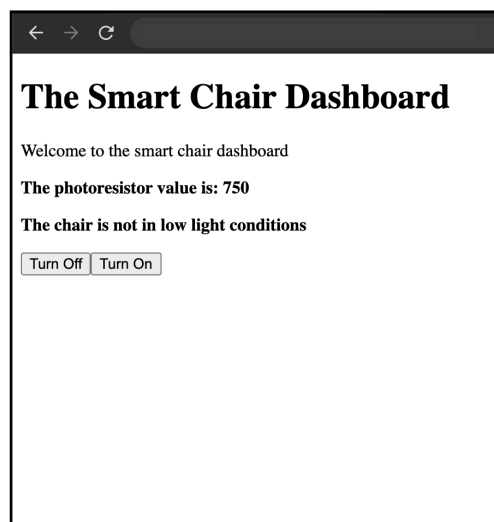


Smart Chair

Part two: Introduction to the web server

Project description:

In this project, you will learn how to turn your ESP board into a simple web server. The project uses the smart chair circuit that you built on previous exercises. You will use the smart chair code as a foundation and progressively add the web server functionalities. For instance, you will first transform your ESP board into a fully functional web server. From there, you will use the web server to serve a static html page to visualise information about your circuit. Wouldn't it be great if you had a web dashboard to read the status of your components, or better, change their behaviours?

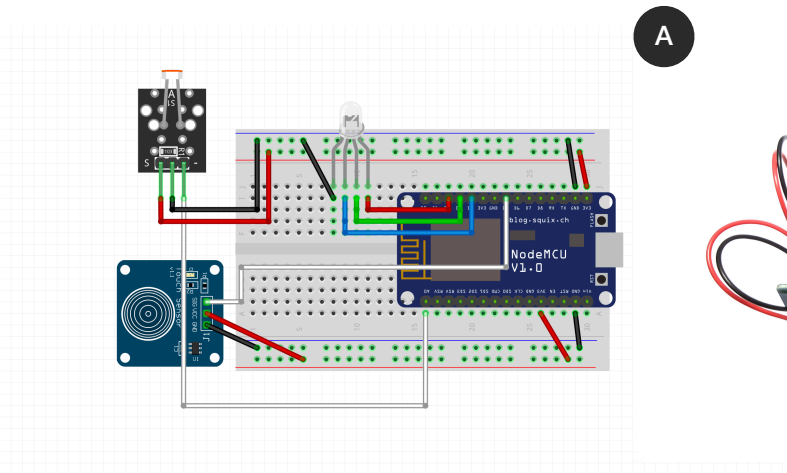


Project objectives:

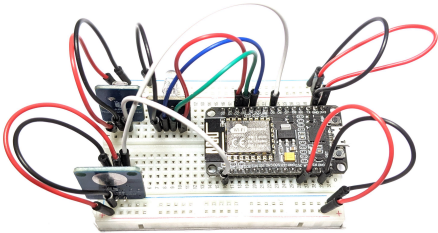
- Configure the ESP board to act as a web server
- Program the ESP board to serve a static web page
- Create a small dashboard for the static web page
- Manipulate components from the static web page

Project components:

Component Reference	Component Quantity	Component Name	Component Description
A	1	Smart Chair Part 1	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

Introduction and Theory

The aim of this project is to revisit the smart chair circuit that you built on previous exercises and use a web server dashboard to read and control some of its components. The ESP board, in fact, has a built-in WiFi module with integrated TCP/IP protocol stack that can give access to your WiFi network.

This means that you can connect the ESP board to your home WiFi, use it as a web server, and access it from a browser.

You are not required to create a new circuit for this exercise as you will use the smart chair circuit that you previously wired up. If you have followed the first part of the smart chair project, you should have a circuit which uses a RGB LED component, a photoresistor component, and a touch sensor component.

The idea here is to use the ESP board as a web server and create a simple dashboard that you can access through a web browser to read and manipulate the values of the circuit components. For example, you will create a static web page to display the value of the photoresistor and use a html button to control the RGB LED component.

Step Two

Connect the ESP board to the Wifi

Let's add some code to the previous smart chair sketch to connect your ESP board to your home WiFi. Go ahead and create a new empty sketch from the Arduino IDE.

You should see an empty sketch like the following:



Feel free to save the sketch and rename it to something sensible: **smart_chair_part2** for instance.

Now copy and paste the code that you wrote for the first part of the smart chair exercise. You should have the code with the **rgbLed()**, the **chairSignal()** , the **isNight()**, and the **lightShow()** utility functions.

Quickly compile and upload the code to your board to make sure that everything still works as expected.

Let's begin by adding some variables at the top of the sketch:

```
#include <ESP8266WiFi.h>

// The WiFi details
const char* ssid = "9145 Hyperoptic Fibre Broadband";
const char* password = "S0g0r47h11K";
```

Type the above code at the top of the sketch, before the `setup()` function.

Here we first `#include` the **ESP8266WiFi** library. The library allows us to use simple functions to connect the ESP board to the local WiFi. **ssid** and **password** are variables to store the WiFi access name and password. Go ahead and change these settings to match your home WiFi.

The very next step is to update the `setup()` function. Here we want to first connect to the WiFi and then print a message with the ESP board IP address on the serial monitor.

The updated **setup()** function:

```
// Put your setup code here, to run once:
void setup() {

    //Connect to the WiFi network
    WiFi.begin(ssid, password);

    // LEDs as OUTPUT
    pinMode(red_led_pin, OUTPUT);
    pinMode(green_led_pin, OUTPUT);
    pinMode(blue_led_pin, OUTPUT);

    // Touch sensor and photoresistors as INPUT
    pinMode(touch_sensor, INPUT);
    pinMode(photo_sensor, INPUT);

    // Start the serial to debug the values
    Serial.begin(9600);

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Waiting to connect...");
    }

    //Print the board IP address
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
}
```

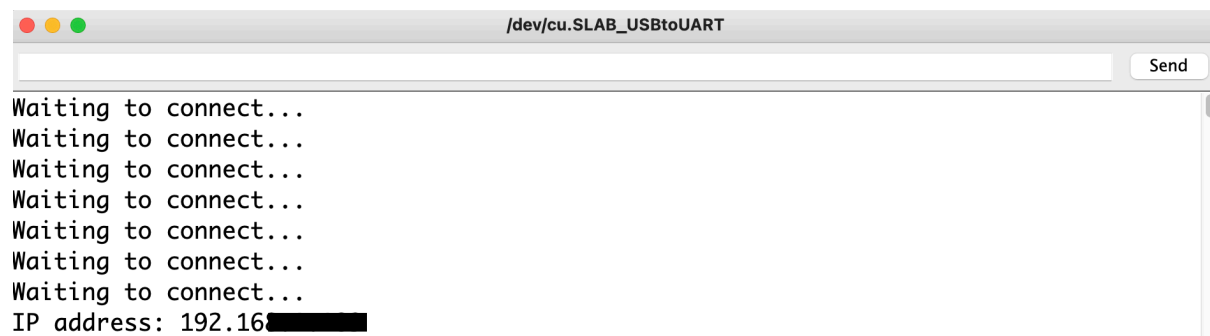
Here there are two main pieces that we added to the setup() function. The **WiFi.begin()** function initialises the WiFi connection. The **while loop** repeatedly checks the connection status (WiFi.status) until the ESP board is connected. Once there is a connection, the serial monitor prints the IP address of the connected ESP board (WiFi.localIP).

The WiFi configuration is now ready.

Go ahead and upload the sketch to your ESP board.

Open the serial monitor and verify that the board connects to your local WiFi.

A successful connection will show the board IP address inside the serial monitor as shown below:



```
/dev/cu.SLAB_USBtoUART
Waiting to connect...
Waiting to connect...
Waiting to connect...
Waiting to connect...
Waiting to connect...
Waiting to connect...
Waiting to connect...
IP address: 192.16[REDACTED]
```

Great, you have now connected your ESP board to you local WiFi network.

Next, you will create a simple web server and use the ESP board IP address to access it via your browser.

Step Three

Create a simple web server

Now that you ESP board is connected to the Wifi, you can turn it into a web server. The idea is to have your ESP board serve a simple static web page that you can access via your browser.

Let's begin by importing a new library to help us create the web server:

```
#include <ESP8266WebServer.h>

// Set the PORT for the web server
ESP8266WebServer server(80);
```

Add the above code just below the **#include <ESP8266WiFi.h>** statement, before the `setup()` function.

Here we first #include the **ESP8266WebServer** library. The library allows us to use simple functions to create a web server. Then, we expose the web server on PORT 80.

The very next step is to update the `setup()` function. Here we want to first initialise the web server and successively create an index route which serves a welcoming plain text.

The updated **`setup()`** function:

```
server.on("/", get_index); // Get the index page on root route
server.begin(); //Start the server
Serial.println("Server listening");
```

Add the above code at the end of the `setup()` function.

Here, we first setup an event listener (`server.on()`) which is triggered when we access the root route “/” of the ESP web server. Then, we start the server and print the “Server listening” message on the serial monitor.

At this point you might be wondering what is **`get_index`**.

`Server.on()` takes two arguments: the first argument is the route (“/”) and the second argument is a reference to a function that should be executed when such route is visited. Here we want to execute the function `get_index()` when we visit the web server root “/” route.

The **`get_index()`** function:

```
void get_index() {

    //Print a welcoming message on the index page
    server.send(200, "text/plain", "Hello! This is an index page.");

}
```

Add the above utility function at the end of the sketch, after the `lightShow()` function.

The **get_index()** function sends a response back, in this case a welcoming message, with the status code (200), the type of the response (text/plain), and the response (Hello! This is an index page).

Great, we have now setup a web server which responds with some plain text when we visit the root route of our ESP board web server. There is one last small step before we can use the web browser to access the response.

The updated **loop()** function:

```
// put your main code here, to run repeatedly:
void loop() {

    // This will keep the server and serial monitor available
    Serial.println("Server is running");

    //Handling of incoming client requests
    server.handleClient();

    // Check the photoresistor threshold
    if(isNight()){
        // It is night, start the light show
        lightShow();
    }else{
        // It is not night, check for chair availability
        chairSignal();
    }

}
```

Here there are two additional lines that we added to the loop() function. The **Serial.println** line will keep the server accessible. **server.handleClient()** is a function which is part of the ESP8266WebServer library. It constantly checks for incoming client requests.

Great, now you have a fully functional web server.

In the next section you will see how you can access the “Hello! This is an index page” response using a web browser.

Step Four

Accessing the web server

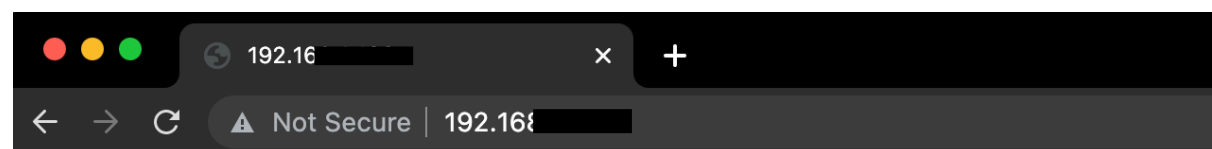
Now that you wrote the code for the web server, it is time for you to upload it to your ESP board. Once it is running, you will be able to access the route “/” web page from your browser and visualise the response.

First thing first, compile your code and upload it to your ESP board.

Once the code is uploaded, open the serial monitor and make sure that the ESP board connected successfully. Now, copy the IP address that you find in the serial monitor. You can use the IP address on a browser to access the ESP board web server:



Now open your favourite browser, and paste the IP address on the URL search bar to access the root route and visualise the message response:



Great, you have successfully turned your ESP board into a web server. In the next session, you will make the home page a little more interesting and serve back an actual HTML page.

Step Five

HTML and the web server

Your ESP board is now a fully functional web server, which means that you can request it to serve HTML files. Let's make the smart chair dashboard a little more interesting.

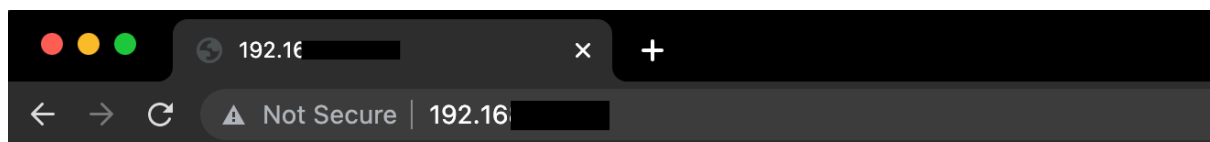
Instead of sending back a plain text response, we can create a string containing some HTML markup and use the `server.send()` function to serve it back to the client.

The updated `get_index()` function:

```
void get_index() {  
  
    String html = "<!DOCTYPE html> <html> ";  
    html += "<head></head>";  
    html += "<body> <h1>The Smart Chair Dashboard</h1>";  
    html += "<p> Welcome to the smart chair dashboard</p>";  
    html += "</body> </html>";  
  
    //Print a welcoming message on the index page  
    server.send(200, "text/html", html);  
  
}
```

Here, you can see that we first initialise a string (`html`) variable containing some html markup and then we use the `server.send()` function to serve it back to the client.

Upload the sketch to your board with the updated `get_index()` function and access the root route on your browser, you should see the following:



The Smart Chair Dashboard

Welcome to the smart chair dashboard

Definitely better than the plain text response. What is going to stop us now from sending back our components readings as part of the HTML document?

Step Six

Displaying values on the dashboard

Now that you know how to serve back an HTML document from the ESP web server, you can incorporate the readings of some of the circuit components. Perhaps we can visualise the photoresistor value and print whether the chair is in a low light condition or not.

The updated `get_index()` function:

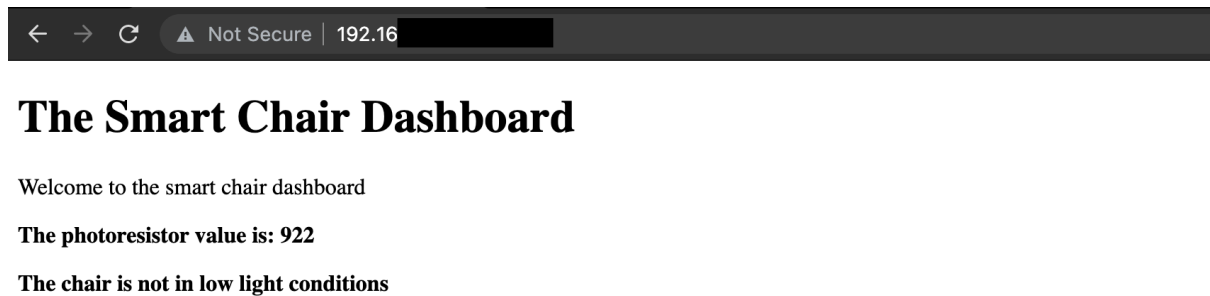
```
void get_index() {
    String html = "<!DOCTYPE html> <html> ";
    html += " <head><meta http-equiv=\"refresh\" content=\"2\"><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"></head>";
    html += " <body> <h1>The Smart Chair Dashboard</h1>";
    html += " <p> Welcome to the smart chair dashboard</p>";
    html += " <div> <p> <strong> The photoresistor value is: ";
    html += analogRead(photo_sensor);
    html += "</strong> </p>";
    html += " <p> <strong> The chair ";
    html += isNight()? "is in low light conditions": "is not in low light conditions";
    html += "</strong> </p> </div>";
    html += " </body> </html>";

    //Print a welcoming message on the index page
    server.send(200, "text/html", html);
}
```

As you might have expected, the html string can be concatenated with variables. The first major change here is the `<head>` tag of the HTML document. The “http-equiv= refresh” meta tag allows the page to refresh on its own. Here, we set the page to refresh every two seconds so that we can read updated incoming values without manually refreshing the page. The second major change is that we concatenate the values that we want to send as a response with the HTML markup. The `isNight()` function is used as a ternary operator to determine the outcome of the message: either “low light conditions” or “not low light conditions”.

Upload the updated code to your ESP board and visit the index page using the browser.

You should see the following dashboard:



Welcome to the smart chair dashboard

The photoresistor value is: 922

The chair is not in low light conditions

The page will refresh every two seconds and update the values. Place the circuit in a low light condition, or vice versa, and see if the message updates correctly.

Great, you can now read your circuit components values from the web server dashboard.

Reading values on a web page is a nice feature for your smart chair system but what about changing the status of a component from the web server dashboard?

Let's say that we want to control the blue component of the RGB LED from the web browser dashboard. We can create two buttons (Turn Off, Turn On) as part of the HTML markup and use them to turn ON and OFF the blue component of the RGB LED from the web server static page.

First, let's add a new route to our server configuration inside the `setup()` function:

```
server.on("/", get_index); // Get the index page on root route
server.on("/setLEDStatus", setLEDStatus); // Get the setLed page
```

Add the “/setLEDStatus” route inside the setup() function, just below the “/” main route. We will create the setLEDStatus function in a moment.

Next, let’s update the get_index() function with two HTML buttons.

The updated **get_index()** function:

```
void get_index() {
    String html = "<!DOCTYPE html> <html> ";
    html += "<head><meta http-equiv='refresh' content='2'><meta name='viewport' content='width=device-width, initial-scale=1.0'></head>";
    html += "<body> <h1>The Smart Chair Dashboard</h1>";
    html += "<p> Welcome to the smart chair dashboard</p>";
    html += "<div> <p> <strong> The photoresistor value is: ";
    html += analogRead(photo_sensor);
    html += "</strong> </p>";
    html += "<p> <strong> The chair ";
    html += isNight()? "is in low light conditions": "is not in low light conditions";
    html += "</strong> </p> </div>";

    html += "<a href='/setLEDStatus?s=0' target='_blank'><button>Turn Off </button></a>";
    html += "<a href='/setLEDStatus?s=1' target='_blank'><button>Turn On </button></a>";

    html += "</body> </html>";

    //Print a welcoming message on the index page
    server.send(200, "text/html", html);
}
```

Here, we added two buttons. The “Turn Off” button will send a request to the ‘setLEDStatus’ route with ‘s=0’ as the query string. The “Turn On” button will send a request to the ‘setLEDStatus’ route with ‘s=1’ as the query string.

The final step is to add the setLEDStatus() function to handle the requests from the “Turn Off” and “Turn On” HTML buttons.

The **setLEDStatus()** function:

```
void setLEDStatus(){
    int query_string = 0;

    // Check the query string
    if (server.arg("s") != ""){ //Parameter found
        // Parse the value from the query
        query_string = server.arg("s").toInt();
        // Check the value and update the blue led pin of the RGB component
        if(query_string==1){
            analogWrite(blue_led_pin, 255);
        }else{
            analogWrite(blue_led_pin, 0);
        }
    }
}
```

Add the above function at the end of the sketch, below the `get_index()` function. The **`setLEDStatus()`** function is called when the client visits the `"/setLEDStatus"` route on the web browser. The function checks if there is a query string parameter with the name `"s"` (`server.args`).

Remember, when we click on the two HTML buttons, we send a request with a query string `"s"` value of 1 (Turn On) or 0 (Turn Off).

The function then checks the value of the retrieved query string and changes the status of the blue led accordingly.

Almost there, the RGB LED component is controlled by the value read from the photoresistor. This means that even if we send a signal to turn on the blue led, the blue led will be turned off by the current code logic.

Go ahead and comment out the code logic inside the `loop()` function responsible for the RGB LED controller of your smart chair. It should look like the following:

```
// put your main code here, to run repeatedly:
void loop() {

    // This will keep the server and serial monitor available
    Serial.println("Server is running");

    //Handling of incoming client requests
    server.handleClient();

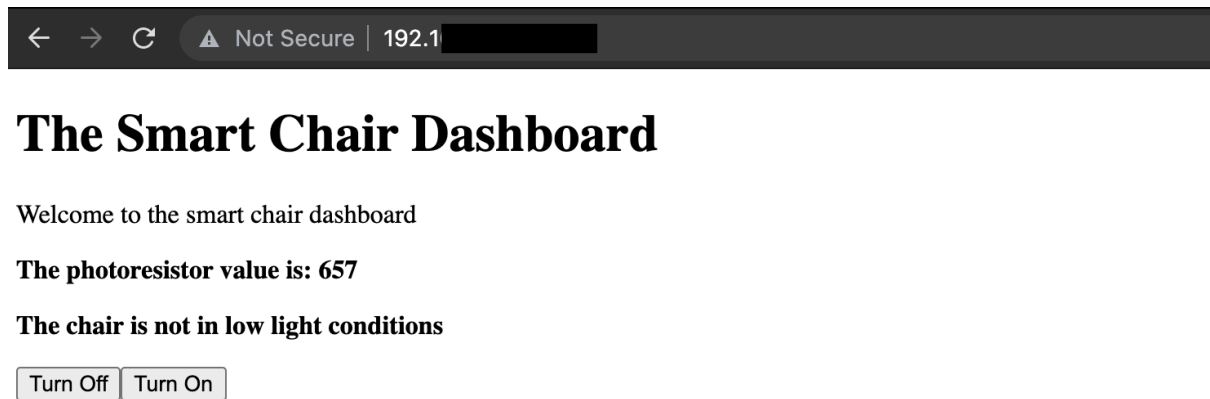
    // Check the photoresistor threshold
    //if(isNight()){
    // It is night, start the light show
    //lightShow();
    //}else{
    // It is not night, check for chair availability
    //chairSignal();
    //}

}
```

You should only have the above two lines of code inside the `loop()` function.

It is now the time for you to upload the latest version of the smart chair code to your ESP board.

Once the code has been uploaded, use the browser to access the dashboard. You should see the following page:



Click on the buttons and watch the blue component of the RGB LED turn ON and OFF. You might want to change the web page refresh rate to a longer period as it is currently 2 seconds.

Congratulations, you have reached the end of this exercise. You are now able to connect the ESP board to your home WiFi, turn it into a web server, and read and write components status from the web dashboard.

Step Five

Additional Tasks

Now that you have successfully completed the web server smart chair exercise, play around with the code and improve the dashboard.

Try the following:

- Improve the overall dashboard layout. The HTML buttons do not have a label and it is difficult to understand what they do.
- Can you add another component to your dashboard? What about reading the value of the touch sensor?
- In the next exercise, you will create the dashboard for the smart fridge circuit on your own. Make sure you take the time to understand the code.