Lab Report

Using ESP8266 as a web server

Goal of the experiment

In this experiment, an ESP8266 microcontroller equipped with a Wi-Fi chip is used as a web server to turn an internal LED on and off. A web page served by the web server shows a link for each action, each of which leads to it owns page that first updates the state of the LED then redirects to the homepage.

Components

This experiment involves the following components:

- *A Witty*, combination of:
    - *an ESP8266* microcontroller, holding a Wi-Fi microchip with built-in TCP/IP
    - *a CH340* USB adapter circuit
- A micro-USB to USB-A cable (to connect with the computer)

Relevant theory

Web servers are usually set up to receive requests from clients, to which they will formulate and send back a response. This configuration is commonly referred to as the **client-server architecture**.

Both the client and the server must follow common, well-established rules when receiving or sending data for it to be processed intelligibly. Those rules dictate the structure of requests and responses and make up a **protocol**, namely the *HyperText Transport Protocol* (**HTTP**), defined as an *application layer* protocol. Historically, the HTTP protocol has relied on the *transport layer* protocol: the *Transmission Control Protocol* (**TCP**), despite its latest available version – HTTP/3 – allowing other transport layer protocols to be used (Wikipedia, n.d.).

Content served over the HTTP protocol is referred to as HTTP resources, which are identified by a specific *Uniform Resource Locator* (**URL**) on the network (Wikipedia, n.d.). A three-digit status code is associated to each response, providing additional information about the content being served. Common resources are files using the *HyperText Markup Language* (**HTML**) format, that make up what we know as *web pages*. A specific web page can then refer to other web pages – as well as other resource types – by referring to their URL.

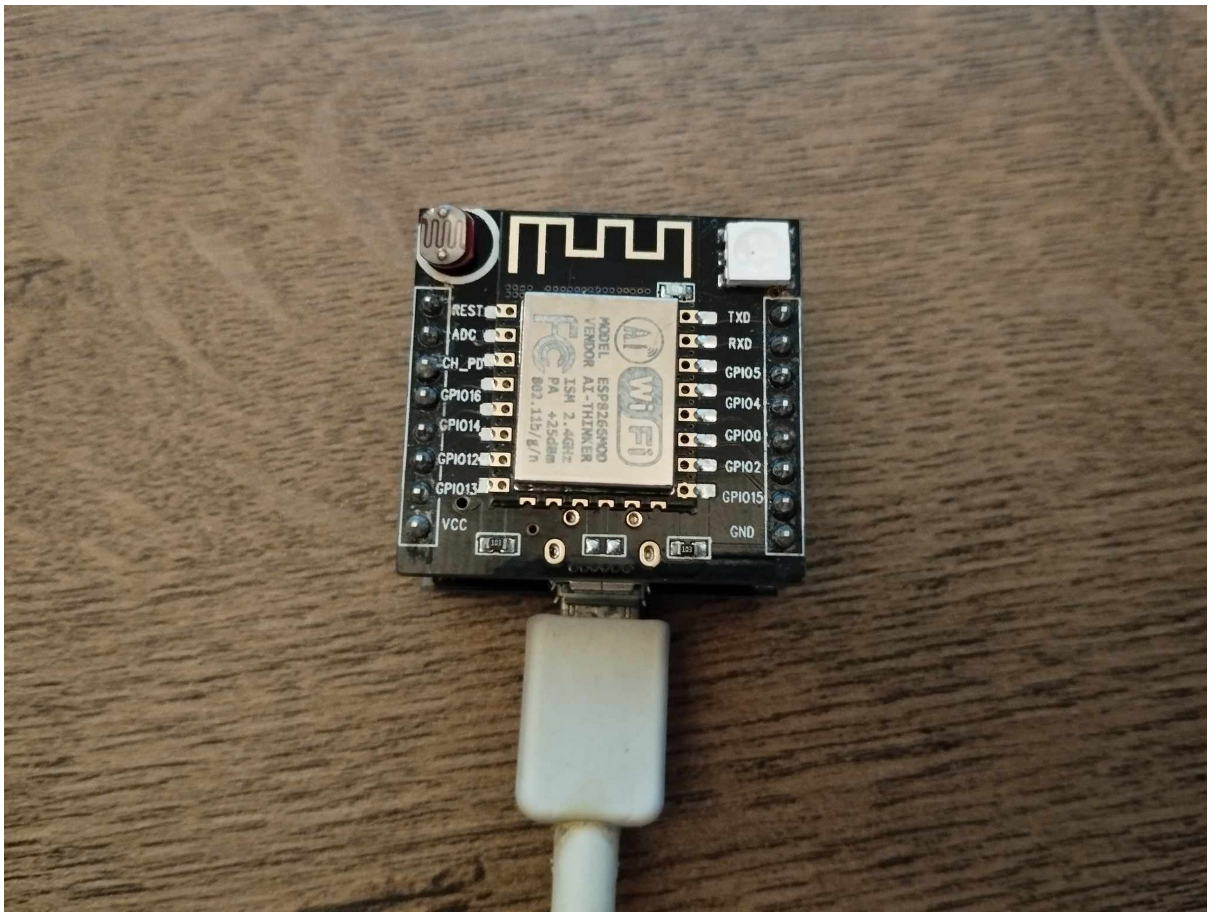
In this experiment, several web pages are made available on the web server, as well as an image file.

Bibliography

Wikipedia. (n.d.). *HTTP*.
Retrieved August 28, 2023, from https://en.wikipedia.org/wiki/HTTP

Setup description

An ESP8266 microcontroller attached to a CH340 USB adapter circuit (together making up the Witty component) is connected to a computer through a micro-USB to USB-A cable. An internal LED on the Witty can be turned on or off by accessing specific web pages on the server.



Figur 1 - Witty component with micro-USB to USB-A cable

Source code

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

// WiFi credentials
#ifndef STASSID
#define STASSID "<SSID>"
#define STAPSK "<PASSWORD>"
#endif

const char* ssid = STASSID;
const char* password = STAPSK;
```

```
ESP8266WebServer server(80);

const int led = 13;

String s;

/**
  Handle for the homepage.
*/
void handleRoot() {
  digitalWrite(led, 1);
  server.send(200, "text/html", s);
  digitalWrite(led, 0);
}

void handleNotFound() {
  digitalWrite(led, 1);
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) { message += " " +
server.argName(i) + ": " + server.arg(i) + "\n"; }
  server.send(404, "text/plain", message);
  digitalWrite(led, 0);
}

void handleTurnLEDOff() {
  Serial.println("LED off");
  digitalWrite(2, HIGH);
  server.sendHeader("Location", "/");
  server.send(303);
}

void handleTurnLEDOn() {
  Serial.println("LED on");
  digitalWrite(2, LOW);
  server.sendHeader("Location", "/");
  server.send(303);
}

void setup(void) {
  pinMode(2, OUTPUT);
  pinMode(led, OUTPUT);
  digitalWrite(led, 0);
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (MDNS.begin("esp8266")) { Serial.println("MDNS responder
started"); }

  // HTML content for the homepage
  s += "<html><head><title>A web page</title></head>";
  s += "<body>";
  s += "<h1>Lab report</h1>";
  s += "<h2>Using ESP8266 as a web server</h2>";
  s += "<p>Control the Witty's internal LED:</p>";
  s += "<ul><li><a href='gpio0'>Turn LED off</a></li><li><a
href='gpio1'>Turn LED on</a></li></ul>";
  s += "</body>";
  s += "</html>";

  server.on("/", handleRoot);

  // Handling turning the LED on/off
  server.on("/gpio0", handleTurnLEDOff);
  server.on("/gpio1", handleTurnLEDOn);

  server.on("/inline", []() {
    server.send(200, "text/plain", "this works as well");
  });

  server.on("/gif", []() {
    static const uint8_t gif[] PROGMEM = {
      0x47, 0x49, 0x46, 0x38, 0x37, 0x61, 0x10, 0x00, 0x10, 0x00, 0x80,
0x01,
      0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0x2c, 0x00, 0x00, 0x00,
0x00,
      0x10, 0x00, 0x10, 0x00, 0x00, 0x02, 0x19, 0x8c, 0x8f, 0xa9, 0xcb,
0x9d,
      0x00, 0x5f, 0x74, 0xb4, 0x56, 0xb0, 0xb0, 0xd2, 0xf2, 0x35, 0x1e,
0x4c,
      0x0c, 0x24, 0x5a, 0xe6, 0x89, 0xa6, 0x4d, 0x01, 0x00, 0x3b
    };
    char gif_colored[sizeof(gif)];
    memcpy_P(gif_colored, gif, sizeof(gif));
    // Set the background to a random set of colors
    gif_colored[16] = millis() % 256;
    gif_colored[17] = millis() % 256;
    gif_colored[18] = millis() % 256;
    server.send(200, "image/gif", gif_colored, sizeof(gif_colored));
  });

  server.onNotFound(handleNotFound);

  /////////////////////////////////////////////////////////
  // Hook examples

  server.addHook([](const String& method, const String& url,
WiFiClient* client, ESP8266WebServer::ContentTypeFunction contentType)
{
    (void)method;       // GET, PUT, ...
    (void)url;          // example: /root/myfile.html
    (void)client;       // the webserver tcp client connection
    (void)contentType;  // contentType(".html") => "text/html"
    Serial.printf("A useless web hook has passed\n");
    Serial.printf("(this hook is in 0x%08x area (401x=IRAM
402x=FLASH))\n", esp_get_program_counter());
    return ESP8266WebServer::CLIENT_REQUEST_CAN_CONTINUE;
  });

  server.addHook([](const String&, const String& url, WiFiClient*,
ESP8266WebServer::ContentTypeFunction) {
    if (url.startsWith("/fail")) {
      Serial.printf("An always failing web hook has been triggered\n");
      return ESP8266WebServer::CLIENT_MUST_STOP;
    }
    return ESP8266WebServer::CLIENT_REQUEST_CAN_CONTINUE;
  });

  server.addHook([](const String&, const String& url, WiFiClient*
client, ESP8266WebServer::ContentTypeFunction) {
    if (url.startsWith("/dump")) {
      Serial.printf("The dumper web hook is on the run\n");

      // Here the request is not interpreted, so we cannot for sure
      // swallow the exact amount matching the full request+content,
      // hence the tcp connection cannot be handled anymore by the
      // webserver.
#ifdef STREAMSEND_API
      // we are lucky
      client->sendAll(Serial, 500);
#else
      auto last = millis();
      while ((millis() - last) < 500) {
        char buf[32];
        size_t len = client->read((uint8_t*)buf, sizeof(buf));
        if (len > 0) {
          Serial.printf("(<%d> chars)", (int)len);
          Serial.write(buf, len);
          last = millis();
        }
      }
#endif
      // Two choices: return MUST STOP and webserver will close it
      //                        (we already have the example with
'/fail' hook)
      // or                    IS GIVEN and webserver will forget it
      // trying with IS GIVEN and storing it on a dumb WiFiClient.
      // check the client connection: it should not immediately be
closed
      // (make another '/dump' one to close the first)
      Serial.printf("\nTelling server to forget this connection\n");
      static WiFiClient forgetme = *client;  // stop previous one if
present and transfer client refcounter
      return ESP8266WebServer::CLIENT_IS_GIVEN;
    }
    return ESP8266WebServer::CLIENT_REQUEST_CAN_CONTINUE;
  });

  // Hook examples
  ///////////////////////////////////////////////////////////

  server.begin();
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();
  MDNS.update();
  delay(1000);
}
```

Result

When browsing to the homepage (at the root directory), a simple web page is displayed with – as expected – two links: one for turning on the internal LED, another for turning it off.

A demonstration of the achieved result can be seen in the video below.

Figur 2 - Live demonstration of the result

Reflection

Using Witty as a web server depicts – on a lower scale – how servers work and can be used and configured to host websites that have the potential to interact with their physical environment. In this specific experiment, it was demonstrated through turning a LED on and off as a response to an HTTP GET request.

On top of demonstrating something that should be common knowledge for most IT professionals, this experiment opens the perspective of more advanced connected objects that can be manipulated through the Internet.