```
/*-----  3 Timersets with timerplans for the Denver SHP-310U 3 output power strip ------
For the Arduino IDE, select board to be Generic ESP8285 Module.
A brief hard- and software description is found at: https://www.larsenhenneberg.dk
-----
Conditions: The program uses Arduino OTA to flash new firmware verions via wifi.
Make sure to use the correct values for:
Access point (AP) SSID and password to use the simple captive portal.
-----
Work scheme at reset or program restart:
.01: The wifi network ssid and password is read from the EEPROM
.02: The wifi connection is tried for a few sec. with the ssid and password, see .01
.03: Upon RTC error, it waits for wifi connection, otherwise read time from RTC
.04: Upon an active setup button, an Access Point (AP) is started, see .06
.05: The program continues into the loop(), see .11
.06: Use f.ex. a mobile phone to use the AP network.
.07: Browse to the captive portal at http://192.168.4.1
.08: Enter the wifi ssid and password at the captive portal web page and press Save.
.09: The ssid and password from the captive portal are saved into the EEprom.
.10: The ESP32 is restarted, and starts from .01
.11: In the loop(): in case of an active AP, the captive portal web client is handled.
.12: A while(1) loop inside loop() is now the main loop.
.13: The yield() function is called in while(1) to avoid a watch dog program reset.
.14: Upon an active setup button, the wifi is diconnected and the ESP32 is restarted, see .01
.15: If wifi is available, the NTP time is read on the hour, e.g. xx:00 and the RTC is updated
.16: If no wifi is available, the time keeping is made with simple delay loops.
-----
The time plans are changed by using a http request to the ip address: http://ip_addr:http_port .
The start and end time points are typed in hh:mm format and saved to the ESP32-S2 EEprom.
#define timerSets sets the number of timer sets.
#define timerPlans sets the number of timeplans for ON/OFF scheduling, max 8 timeplans.
NOTE: the max total number timerplans is 16, calculated from timerSets * timerPlans
-----*/
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <ESP8266WebServer.h>
#include <EEPROM.h>

#define http_Port 8058        // http port to access the timer set/plan webpage
#define TZ_DENMARK "CET-1CEST,M3.5.0,M10.5.0/3" // time zone and daylight def.

// define number of timerSets and timerPlans for each timerSet
// NOTE: timerSet*timerPlans is MAX. 16
#define timerSets  3      // number of timeSets, each with an associated timerSetx_pin output
#define timerPlans 4      // number of timerPlans for each timerSet

#define wifi_LED_pin 4         // GPIO4 as output , active low for wifi LED light
#define setup_Pin 13           // GPIO13 input pin for start the captive portal

#define Relay1_pin 12          // GPIO12 as output , active low for relay1 and red LED1 on
#define Relay2_pin 14          // GPIO14 as output , active low for relay2 and red LED2 on
#define Relay3_pin 5           // GPIO5 as output , active low for relay3 and red LED3 on

// Define the output pin for each timerSet output
#define timerSet1_pin 12      // GPIO33 as timerset1 active high output and red LED1 on
#define timerSet2_pin 14      // GPIO37 as timerset2 active high output and red LED2 on
#define timerSet3_pin 5      // GPIO37 as timerset2 active high output and red LED2 on

#define EEPROM_SIZE 512     // Allocate 512 bytes for wifi ssid, passw., and timeplans
#define EEprom_adr 0        // EEprom start address normally 0
```

```
// Soft AP ( Access Point) network setup for the captive portal
const char* APssid = "Denver week timers";
const char* APpassword = "ap-password";              // might add AP password here !

//bool APactive;                // to notify the loop() of an active captive portal
ESP8266WebServer APserver(80);       // this webservice is for AP only
WiFiServer server(http_Port); // webservice for the timeplan setup

// EEprom parameters are read in at start, and saved on changes.
struct wifiSettings {   // the wifiSettings
  char ssid[30];
  char password[30];
} user_wifi = {};

struct turnTime{    // definition of one start time or end time for the time plans
  byte Hour;
  byte Minute;
};
// the timerSets with the timerPlans with start-end_Hour:Minute entries, 7 days a week
struct turnTime startTime[timerSets][7][timerPlans], endTime[timerSets][7][timerPlans];

// size of one day time plan entry e.g. size of start-time + sizeof end-time
#define dayturnTimeSize 2*sizeof(turnTime)

// size of one timerSet, 7 days a week
#define timerSetSize 7*dayturnTimeSize*timerPlans

struct turnMinutes{   // to hold number of minutes from 00:00
  unsigned int startMinutes;
  unsigned int endMinutes;
};
// for the number of minutes from 00:00 for all the Hour:Minute entries the in time plans
struct turnMinutes onoffMinutes[timerSets][7][timerPlans];

bool getTime;              // flag to request NTP time

// values in msec used by millis() for non-blocking loop() processing
uint32_t previousMillis = 0;
uint32_t OneMinute = 60 * 1000L;
uint32_t adjustMinute;    // to adjust sw time keeping to NTP
bool adjustSWtiming;      // bool to indicate use of adjustMinute

// globals for NTP and time keeping
const char *TZstr = TZ_DENMARK;   // NTP DK time string
struct tm timeinfo;
time_t now;
String Hour_str = "   ";
String Minute_str = "   ";
String Second_str = "   ";
String Day_str = "   ";
String timestr;
// RTC variables
byte Second;      // Second 0..59
byte Hour;        // Hours 0..23
byte Minute;      // Minutes 0..59
byte Day;         // Day 0..6 , 0 is Monday

void setup() {
  pinMode(setup_Pin, INPUT_PULLUP);
  pinMode(wifi_LED_pin, OUTPUT);   // Blue LED pin as output
  pinMode(timerSet1_pin, OUTPUT);  // timerset1 and red LED1 pin as output
  pinMode(timerSet2_pin, OUTPUT);  // timerset2 and red LED2 pin as output
  pinMode(timerSet3_pin, OUTPUT);  // timerset3 and red LED3 pin as output
  digitalWrite(wifi_LED_pin, 1);   // set blue LED inactive
```

```
  digitalWrite(timerSet1_pin, 0);  // set timerset1 output inactive
  digitalWrite(timerSet2_pin, 0);  // set timerset2 output inactive
  digitalWrite(timerSet3_pin, 0);  // set timerset3 output inactive
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.get(0, user_wifi);
  // read timeSets with timePlans from EEprom
  for (int ts=0; ts < timerSets; ts++) {
    for (int tp=0; tp < timerPlans; tp++) {
      for (int wday=0; wday < 7; wday++) {
        EEPROM.get(EEprom_adr+ ts*timerSetSize +(wday+7*tp)*dayturnTimeSize+sizeof(struct
wifiSettings), startTime[ts][wday][tp]);
        EEPROM.get(EEprom_adr+ ts*timerSetSize +((2*wday+1)+2*7*tp)*sizeof(turnTime)+sizeof(
struct wifiSettings), endTime[ts][wday][tp]);
      }
    }
  }
  EEPROM.end();
   // check if Hour:Minute values from the EEprom are out of limit, then Hour:Minute = 00:00
  for (int ts=0; ts < timerSets; ts++) {
    for (int tp=0; tp < timerPlans; tp++) {
      for (int wday=0; wday < 7; wday++) {
        if ((startTime[ts][wday][tp].Hour < 0) || (startTime[ts][wday][tp].Hour > 23)) {
startTime[ts][wday][tp].Hour = 0; }
        if ((startTime[ts][wday][tp].Minute < 0) || (startTime[ts][wday][tp].Minute > 59)) {
startTime[ts][wday][tp].Minute = 0; }
        if ((endTime[ts][wday][tp].Hour < 0) || (endTime[ts][wday][tp].Hour > 23)) { endTime[ts
][wday][tp].Hour = 0; }
          if ((endTime[ts][wday][tp].Minute < 0) || (endTime[ts][wday][tp].Minute > 59)) {
endTime[ts][wday][tp].Minute = 0; }
      }
    }
  }
  // for all Hour:Minute entries in the timePlans for each timeSet:
  // calculate start-ON and end-OFF points in minutes: time_in_minutes = 60*hours+minutes
  for (int ts=0; ts < timerSets; ts++) {
    for (int tp=0; tp < timerPlans; tp++) {
      for (int wday=0; wday < 7; wday++) {
        onoffMinutes[ts][wday][tp].startMinutes = int(startTime[ts][wday][tp].Hour)*60+int(
startTime[ts][wday][tp].Minute);
        onoffMinutes[ts][wday][tp].endMinutes = int(endTime[ts][wday][tp].Hour)*60+int(endTime[
ts][wday][tp].Minute);
      }
    }
  }
 // try wifi connection with ssid,password from EEprom
  WiFi.mode(WIFI_STA);
  WiFi.begin(user_wifi.ssid, user_wifi.password);
  while ((WiFi.status() != WL_CONNECTED) && (!setupButton())) {
    digitalWrite(wifi_LED_pin, 0);
    delay(500);
    digitalWrite(wifi_LED_pin, 1);
    delay(500);
  }
  if ((WiFi.status() != WL_CONNECTED) || (setupButton())){
    WiFi.mode(WIFI_AP);
    WiFi.softAP(APssid, APpassword);
    APserver.on("/",  handlePortal);      // start captive portal
    APserver.begin();
  }
  initOTA();    // start the "Over The Air" firmware update option

  // Set time via NTP, as required for x.509 validation
  //configTime(3 * 3600, 0, "pool.ntp.org", "time.nist.gov");
    configTime (TZstr, "pool.ntp.org", "time.nist.gov");
```

```
  initOTA();      // start the "Over The Air" firmware update option
  server.begin();
  getTime = true; // force a NTP request in loop()
  previousMillis = millis();
}

void loop() {
  while (WiFi.status() != WL_CONNECTED) { // waiting for AP client
    digitalWrite(wifi_LED_pin, 0);
    delay(100);                       // short blink to indicate
    digitalWrite(wifi_LED_pin, 1);    // an active captive portal
    delay(500);
    APserver.handleClient();              // to finish portal
  }
  APserver.close();   // Close the APserver
  while (1) {             // going into the main loop !
    yield();              // refresh wdt (watch dog timer) to avoid reset
    if (setupButton()) {  // setup button pressed ?
      WiFi.disconnect();  // disconnect wifi
      ESP.restart();      // restart to continue with AP
    }
   // if wifi available then request server regularly
  if ((WiFi.status() == WL_CONNECTED) && ( getTime == true)) {
    getTime = false;
    digitalWrite(wifi_LED_pin, 0); // set blue LED active
    now = time(nullptr);
    while (now < 8 * 3600 * 2) {
      delay(500);
      now = time(nullptr);
    }
    gmtime_r(&now, &timeinfo);
    timestr = ctime(&now);  // ex.: Wed Dec 27 17:43:24 2023
    Day_str[0] = timestr[0];
    Day_str[1] = timestr[1];
    Day_str[2] = timestr[2];       // get day of week
    Day = 0;
    if (Day_str == "Mon") { Day = 0; }
    if (Day_str == "Tue") { Day = 1; }
    if (Day_str == "Wed") { Day = 2; }
    if (Day_str == "Thu") { Day = 3; }
    if (Day_str == "Fri") { Day = 4; }
    if (Day_str == "Sat") { Day = 5; }
    if (Day_str == "Sun") { Day = 6; }
    Hour_str[0] = timestr[11];
    Hour_str[1] = timestr[12];
    Hour = Hour_str.toInt();         // save hour as byte
    Minute_str[0] = timestr[14];
    Minute_str[1] = timestr[15];
    Minute = Minute_str.toInt();     // save minute as byte
    Second_str[0] = timestr[17];
    Second_str[1] = timestr[18];
    Second = Second_str.toInt();    // save second as byte
    adjustSWtiming = true;          // use adjustMinute once in SW timer
    adjustMinute = (60 - Second) * 1000L; // set sw time adjust value
  }
  unsigned int timeNow_minutes = Hour*60 + Minute; // current no. of minutes since 00:00
  // check the ON periods for the outputs
  bool outputStates[timerSets];
  for (int ts=0; ts < timerSets; ts++) {
    outputStates[ts] = false;
    for (int tp = 0; tp < timerPlans; tp++) {
      if ((timeNow_minutes < onoffMinutes[ts][Day][tp].endMinutes) && (timeNow_minutes >=
onoffMinutes[ts][Day][tp].startMinutes)) {
```

```
        outputStates[ts] = true;
      }
    }
  }
  // set port value for output 1
  if (outputStates[0] == true) { digitalWrite(timerSet1_pin, 1); } // set timerset1 output and
red LED1 active
  else { digitalWrite(timerSet1_pin, 0); }                   // disable timerset1 output and
red LED1
  // set port value for output 2
  if (outputStates[1] == true) { digitalWrite(timerSet2_pin, 1); } // set timerset2 output and
red LED2 active
  else { digitalWrite(timerSet2_pin, 0); }                   // disable timerset2 output and
red LED2
  // set port value for output 3
  if (outputStates[2] == true) { digitalWrite(timerSet3_pin, 1); } // set timerset3 output and
red LED3 active
  else { digitalWrite(timerSet3_pin, 0); }                   // disable timerset3 output and
red LED3
  // if wifi: look for an OTA update and client request
  if ((WiFi.status() == WL_CONNECTED)) {
   digitalWrite(wifi_LED_pin, 0); // set blue LED active
   ArduinoOTA.handle();
   WiFiClient client = server.available();
   if (client) {
    String currentLine = "";                // make a String to hold incoming data from the
client
    String header = "";                     // Variable to store the HTTP request
    byte oldHour, oldMinute;
    char hourMSB, hourLSB, minMSB, minLSB, wday;
    while (client.connected()) { // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        header += c;                        // add to the header
        if (c == '\n') {                    // if end of the client HTTP request
          if (currentLine.length() == 0) {  // then send a response:
            client.println("HTTP/1.0 200 OK");    // send HTTP header
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            int len = header.length();
            int idx, timeSet;
            idx = header.indexOf("s"); // index of timeplan set and start timeplan
            if (idx > 0) { timeSet = header[idx+1] - '0'; }
            int timeplan = header[idx+2] - '0'; // timeplan 1 (from sx0)
            int wday = header[idx+4] - '0';   // day of week (from sx0d0)
            if ((idx > 0) && (timeplan == 0) && (wday == 0)) {
              while ((idx < len) && (timeplan < timerPlans)) {
                for (wday = 0; wday < 7; wday++) {
                  oldHour = startTime[timeSet][wday][timeplan].Hour;
                  oldMinute = startTime[timeSet][wday][timeplan].Minute;
                  hourMSB = header[idx+6] - '0';
                  hourLSB = header[idx+7] - '0';
                  minMSB = header[idx+11] - '0';
                  minLSB = header[idx+12] - '0';
                  if (((hourMSB >= 0) && (hourMSB <= 2)) && ((hourLSB >= 0) && (hourLSB <= 9)) &&
                    ((minMSB >= 0) && (minMSB <= 5)) && ((minLSB >= 0) && (minLSB <= 9))) {
                    startTime[timeSet][wday][timeplan].Hour = hourMSB*10 + hourLSB;
                    startTime[timeSet][wday][timeplan].Minute = minMSB*10 + minLSB;
                    if ((oldHour != startTime[timeSet][wday][timeplan].Hour) || (oldMinute !=
startTime[timeSet][wday][timeplan].Minute)) {
                      EEPROM.begin(EEPROM_SIZE);
                      EEPROM.put(EEprom_adr+timeSet*timerSetSize+(wday+ 7*timeplan)*
dayturnTimeSize+sizeof(struct wifiSettings), startTime[timeSet][wday][timeplan]);
```

```
                      EEPROM.end();
                      onoffMinutes[timeSet][wday][timeplan].startMinutes = startTime[timeSet][
wday][timeplan].Hour*60+startTime[timeSet][wday][timeplan].Minute;
                    }
                  }
                  oldHour = endTime[timeSet][wday][timeplan].Hour;
                  oldMinute = endTime[timeSet][wday][timeplan].Minute;
                  hourMSB = header[idx+17] - '0';
                  hourLSB = header[idx+18] - '0';
                  minMSB = header[idx+22] - '0';
                  minLSB = header[idx+23] - '0';
                  if (((hourMSB >= 0) && (hourMSB <= 2)) && ((hourLSB >= 0) && (hourLSB <= 9)) &&
                    ((minMSB >= 0) && (minMSB <= 5)) && ((minLSB >= 0) && (minLSB <= 9))) {
                    endTime[timeSet][wday][timeplan].Hour = hourMSB*10 + hourLSB;
                    endTime[timeSet][wday][timeplan].Minute = minMSB*10 + minLSB;
                    if ((oldHour != endTime[timeSet][wday][timeplan].Hour) || (oldMinute !=
endTime[timeSet][wday][timeplan].Minute)) {
                      EEPROM.begin(EEPROM_SIZE);
                      EEPROM.put(EEprom_adr+timeSet*timerSetSize+(((2*wday+1)+2*7*timeplan))*
sizeof(turnTime)+sizeof(struct wifiSettings), endTime[timeSet][wday][timeplan]);
                      EEPROM.end();
                      onoffMinutes[timeSet][wday][timeplan].endMinutes = endTime[timeSet][wday][
timeplan].Hour*60+endTime[timeSet][wday][timeplan].Minute;
                    }
                  }
                  idx = idx + 25;
                } // for wday < 7
                timeplan++;
              } // while (len < .. ) loop
            } // if found timeset
            String s, timeValue, strDay, outputState;
            byte timePart;
            if (Day == 0) { strDay = "Monday"; }
            if (Day == 1) { strDay = "Tuesday"; }
            if (Day == 2) { strDay = "Wednesday"; }
            if (Day == 3) { strDay = "Thursday"; }
            if (Day == 4) { strDay = "Friday"; }
            if (Day == 5) { strDay = "Saturday"; }
            if (Day == 6) { strDay = "Sunday"; }
            strDay += " " + time2String(); // Weekday hour:minute
            for (int ts=0; ts < timerSets; ts++) {
              outputStates[ts] = false;
              for (int tp = 0; tp < timerPlans; tp++) {
                if ((timeNow_minutes < onoffMinutes[ts][Day][tp].endMinutes) && (timeNow_minutes
 >= onoffMinutes[ts][Day][tp].startMinutes)) {
                  outputStates[ts] = true;
                }
              }
            }
            outputState = "The states for the outputs [";
            for (int ts=0; ts < timerSets-1; ts++) { outputState += "#" + String(ts+1) + ","; }
            outputState += "#" + String(timerSets) + "]: ";
            for (int ts=0; ts < timerSets-1; ts++) {
              if (outputStates[ts] == true) { outputState += "ON, "; }
              else { outputState += "OFF, "; }
            }
            if (outputStates[timerSets-1] == true) { outputState += "ON."; }
            else { outputState += "OFF."; }
            client.println("<!DOCTYPE html><html lang=\"en\"><head><title>Power strip week
plans</title><style>td { font-size: 14px; text-align: center; }</style></head><body>"); //
center table cell text
            client.println("<h4> " + strDay + " - " + outputState + "</h4>");
            for (int ts = 0; ts < timerSets; ts++) {
              s = "<form><table><tr><td><b>Output #" + String(ts+1) +
```

```
"</b></td><td><b>Monday</b></td><td></td><td><b>Tuesday</b></td><td></td><td><b>Wednesday</b></td
><td></td><td><b>Thursday</b></td><td></td><td><b>Friday</b></td><td></td><td><b>Saturday</b></td
><td></td><td><b>Sunday</b></td></tr>";
              for (idx=0; idx < timerPlans; idx++) {
                s += "<tr><td><b>Timeplan " + String(idx+1) + "</b></td>";
                for (wday=0; wday < 7; wday++) {
                  timeValue ="";
                  timePart = startTime[ts][wday][idx].Hour;
                  if (timePart < 10) { timeValue = "0"; }
                  timeValue +=  String(timePart) + ":";
                  timePart = startTime[ts][wday][idx].Minute;
                  if (timePart < 10) { timeValue += "0"; }
                  timeValue += String(timePart);
                  s += "<td><input type=\"time\" name=\"s" + String(ts) + String(idx) + "d" +
String(wday) + "\" value=" + "\"" + timeValue + "\">";
                  timeValue = "";
                  timePart = endTime[ts][wday][idx].Hour;
                  if (timePart < 10) { timeValue = "0"; }
                  timeValue += String(timePart) +":";
                  timePart = endTime[ts][wday][idx].Minute;
                  if (timePart < 10) { timeValue += "0"; }
                  timeValue += String(timePart);
                  s += "<input type=\"time\" name=\"en\" value=\"" + timeValue + "\"></td>";
                  if (wday != 6) { s += "<td>-</td>"; }
                }  // for wday 0..6
                s += "</tr>";
              }  // for 0..timeplans
              s += "</table><input type=\"submit\" value=\"Save output #" + String(ts+1) + "
time plans" + "\" style=\" border-radius:5px; padding: 5px; margin:5px 0;
background-color:#eafafb;\"></form><p></p>";
                client.println(s);
            }
            client.println("</body></html>");
            client.println();
            break;
          } // len == 0
          else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } // if c == '\n'
        else {
          if (c != '\r') {  // if you got anything else but a carriage return character,
            currentLine += c;      // add it to the end of the currentLine
          }
        }
      } // if client available
      yield();
    }  //  while client connected
    header = "";
    client.stop();
   } // if client
  }  // if wifi connected
  else {  // if no wifi: set blue LED inactive
   digitalWrite(wifi_LED_pin, 1);
  }
  // always time keeping
  if (adjustSWtiming == true) { // check NTP sync.:
    OneMinute = adjustMinute;   // using an adjusted, shorter OneMinute !
    adjustSWtiming = false;     // only once per NTP sync. !
  }
  else { OneMinute = 60 * 1000L; } // just a full minute check
  if (millis() - previousMillis >= OneMinute) {
    previousMillis = millis();
    Minute++;
```

```
    if (Minute >= 60) {
      getTime = true;
      Minute = 0;
      Hour++;
      if (Hour >= 24) { // passed midnight ?
        Hour = 0;
        Day++;
        if (Day >= 7) { Day = 0; }
      }
    }
  }
  }  //  while(1)
}

String time2String() {
  String time_2digits = "";
  if (Hour <= 9) { time_2digits += "0"; }
  time_2digits += String(Hour) + ":";
  if (Minute <= 9) { time_2digits += "0"; }
  time_2digits += String(Minute);
  return time_2digits;
}

bool setupButton() {
  bool result = false;
  if (digitalRead(setup_Pin) == 0) {
    delay(20);    // simple anti bounce delay
    if (digitalRead(setup_Pin) == 0) { result = true; }
  }
  return result;
}

void handlePortal() {
  if (APserver.method() == HTTP_POST) {
    struct wifiSettings temp_user_wifi;
    strncpy(temp_user_wifi.ssid,     APserver.arg("ssid").c_str(),     sizeof(user_wifi.ssid));
    strncpy(temp_user_wifi.password, APserver.arg("password").c_str(), sizeof(user_wifi.password
));
    temp_user_wifi.ssid[APserver.arg("ssid").length()] = temp_user_wifi.password[APserver.arg(
"password").length()] = '\0';
    if ((temp_user_wifi.ssid != user_wifi.ssid) || (temp_user_wifi.password != user_wifi.
password)) {
      user_wifi = temp_user_wifi;
      EEPROM.begin(sizeof(struct wifiSettings));
      EEPROM.put(0, user_wifi);
      EEPROM.end();
    }
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
Setup</title><style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.for
m-control{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max
-width:400px;padding:15px;margin:auto;}h1,p{text-align: center}</style></head><body><main
class='form-signin'><h1>Wifi Setup</h1><br><br><p>The wifi settings have been saved.<br><br>The
device will restarted.</p></main></body></html>" );
    delay(200);     // delay to allow the response to be send to the client..
    ESP.restart();  // ..before resetting for setup() to try the ssid, password
  } else {
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
```

```
Setup</title> <style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.for
m-control{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{cursor: pointer;border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max
-width:400px;padding:15px;margin:auto;}h1{text-align: center}</style></head><body><main
class='form-signin'><form action='/' method='post'><h1 class=''>Wifi Setup</h1><br><div
class='form-floating'><label>SSID</label><input type='text' class='form-control'
name='ssid'></div><div class='form-floating'><br><label>Password</label><input type='password'
class='form-control' name='password'></div><br><br><button
type='submit'>Save</button></form></main> </body></html>" );
  }
}
void initOTA() {
    // Port defaults to 8266
    // ArduinoOTA.setPort(8266);

    // Hostname defaults to esp8266-[ChipID]
    // ArduinoOTA.setHostname("smart_plug");

    // No authentication by default
    // ArduinoOTA.setPassword("admin");

    // Password can be set with it's md5 value as well
    // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
    // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

    ArduinoOTA.onStart([]() {
            String type;
            if (ArduinoOTA.getCommand() == U_FLASH) {
                    type = "sketch";
            } else { // U_FS
                    type = "filesystem";
            }
    });

    ArduinoOTA.onEnd([]() {     });

    ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
//          Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    });

    ArduinoOTA.onError([](ota_error_t error) { //   Serial.printf("Error[%u]: ", error);
            if (error == OTA_AUTH_ERROR) { //       Serial.println("Auth Failed");
            } else if (error == OTA_BEGIN_ERROR) { //   Serial.println("Begin Failed");
            } else if (error == OTA_CONNECT_ERROR) { // Serial.println("Connect Failed");
            } else if (error == OTA_RECEIVE_ERROR) { // Serial.println("Receive Failed");
            } else if (error == OTA_END_ERROR) {// Serial.println("End Failed");
            }
    });
    ArduinoOTA.begin();
}
```