Piotr Świstowski Index 59220

Dokumentacja Projektu: Webowy Sterownik LED (ESP32 + NeoPixel)

Wykorzystane komponenty:

- **ESP32** – mikrokontroler odpowiedzialny za logikę i obsługę sieci Wi-Fi.
- **Zasilacz 5V 20A** – wydajne źródło zasilania dla dużej macierzy LED.
- **Kondensator 1000µF 6.3V+** – zabezpiecza układ LED przed nagłym skokiem napięcia przy uruchomieniu.
- **Rezystor 330Ω** – ogranicza prąd na linii danych sygnałowych do LED.
- **Panel LED NeoPixel 32x8 (256 diod)** – adresowalna macierz WS2812B, sterowana przez ESP32.
- **Przewody połączeniowe + płytka stykowa** – elementy montażowe.

Opis funkcjonalności:

Urządzenie tworzy punkt dostępowy Wi-Fi, przez który użytkownik może sterować różnymi efektami na macierzy LED za pomocą intuicyjnego interfejsu webowego. Obsługiwane tryby:

1. **Tekst statyczny** – wyświetlenie ciągłego komunikatu pionowo.
2. **Przewijanie tekstu (scroll)** – pionowe przesuwanie tekstu w górę.
3. **Flash** – miganie panelem w kolorach czerwony/niebieski.
4. **Tęcza (rainbow)** – efekt płynnego przechodzenia kolorów.
5. **Zegar** – wyświetlanie zegara na podstawie ustawionego czasu.
6. **Stoper** – licznik czasu w górę, z przyciskami Start/Stop/Reset.
7. **Timer** – odliczanie czasu w dół, z konfigurowanym czasem.
8. **Tabata** – interwałowy tryb treningowy (work/rest x8).
9. **Wyłączenie** – zgaszenie wszystkich LED-ów.

Interfejs użytkownika:

Po podłączeniu do sieci Wi-Fi o nazwie `PanelLED`, użytkownik może wejść na adres `192.168.10.1`, gdzie znajdzie formularz pozwalający:

- ustawić tryb pracy,
- zmienić kolor tekstu,

- wprowadzić nowy tekst przewijany/statyczny,
- ustawić czas zegara,
- sterować stoperem, timerem i tabatą.

Schemat połączeń:

Komponent	Połączenie z ESP32
NeoPixel DIN	GPIO22 (przez rez. 330Ω)
NeoPixel VCC	5V z zasilacza
NeoPixel GND	GND zasilacza i ESP32
Kondensator 1000µF	między VCC i GND LED

Uwaga: zasilanie 5V LED nie może być brane z ESP32 – konieczne jest zewnętrzne źródło.

WAŻNE: Minus (GND) zasilacza **musi być połączony** z masą (GND) ESP32. Wspólna masa jest konieczna do prawidłowej transmisji sygnału sterującego do panelu NeoPixel.

Dodatkowe informacje:

- Czcionka 5x7 dla znaków alfanumerycznych jest zapisana w kodzie.
- Efekty wizualne są renderowane z częstotliwością ~10-30ms.
- Projekt wykorzystuje bibliotekę FastLED do zarządzania macierzą.

```
#include <WiFi.h>
#include <WebServer.h>
#include <FastLED.h>
```

```
// --- LED Konfiguracja ---
#define LED_PIN     22
#define WIDTH       8
#define HEIGHT      32
#define NUM_LEDS    (WIDTH * HEIGHT)
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB
#define BRIGHTNESS  100

CRGB leds[NUM_LEDS];
CRGB* matrix[WIDTH][HEIGHT];

// --- Serwer i dane ---
WebServer server(80);
int effectMode = 0;
unsigned long lastEffect = 0;
bool blinkState = false;
int scrollY = HEIGHT;

// --- Teksty i kolory ---
String text = "Lorem";
String staticText = "Lorem Ipsum is simply dummy text of the printing and typesetting
industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500";
CRGB scrollColor = CRGB::Blue;
CRGB staticColor = CRGB::Blue;
String htmlForm;
String manualTime = "12:34:56";  // Domyślny czas
unsigned long timeSetMillis = 0;   // Kiedy ustawiono czas
// --- Zmienne globalne ---
unsigned long stopwatchStart = 0;
bool stopwatchRunning = false;
unsigned long stopwatchElapsed = 0;

unsigned long timerDuration = 0;
unsigned long myTimerStart = 0;
bool timerRunning = false;

// Tabata: domyślnie 20s pracy, 10s przerwy, 8 rund
int tabataRound = 0;
bool tabataWork = true;
unsigned long tabataStart = 0;
bool tabataRunning = false;
int tabataWorkSec = 20;
int tabataRestSec = 10;



// --- Czcionka 5x7 ---
const uint8_t font[][5] = {
  // 0-9
  {0x3E,0x45,0x49,0x51,0x3E}, // 0
  {0x00,0x41,0x7F,0x40,0x00}, // 1
```

```
  {0x62,0x51,0x49,0x49,0x46}, // 2
  {0x22,0x41,0x49,0x49,0x36}, // 3
  {0x18,0x14,0x12,0x7F,0x10}, // 4
  {0x2F,0x49,0x49,0x49,0x31}, // 5
  {0x3E,0x49,0x49,0x49,0x30}, // 6
  {0x01,0x71,0x09,0x05,0x03}, // 7
  {0x36,0x49,0x49,0x49,0x36}, // 8
  {0x06,0x49,0x49,0x49,0x3E}, // 9

  // A-Z
  {0x7E,0x11,0x11,0x11,0x7E},{0x7F,0x49,0x49,0x49,0x36},
  {0x3E,0x41,0x41,0x41,0x22},{0x7F,0x41,0x41,0x22,0x1C},
  {0x7F,0x49,0x49,0x49,0x41},{0x7F,0x09,0x09,0x09,0x01},
  {0x3E,0x41,0x49,0x49,0x7A},{0x7F,0x08,0x08,0x08,0x7F},
  {0x00,0x41,0x7F,0x41,0x00},{0x20,0x40,0x41,0x3F,0x01},
  {0x7F,0x08,0x14,0x22,0x41},{0x7F,0x40,0x40,0x40,0x40},
  {0x7F,0x02,0x04,0x02,0x7F},{0x7F,0x04,0x08,0x10,0x7F},
  {0x3E,0x41,0x41,0x41,0x3E},{0x7F,0x09,0x09,0x09,0x06},
  {0x3E,0x41,0x51,0x21,0x5E},{0x7F,0x09,0x19,0x29,0x46},
  {0x46,0x49,0x49,0x49,0x31},{0x01,0x01,0x7F,0x01,0x01},
  {0x3F,0x40,0x40,0x40,0x3F},{0x1F,0x20,0x40,0x20,0x1F},
  {0x7F,0x20,0x18,0x20,0x7F},{0x63,0x14,0x08,0x14,0x63},
  {0x07,0x08,0x70,0x08,0x07},{0x61,0x51,0x49,0x45,0x43},

  // Dwukropek
  {0x00,0x36,0x36,0x00,0x00},

  // Pusty znak
  {0x00,0x00,0x00,0x00,0x00}
};
// --- Pomocnicze ---
int XY(int x, int y) {
  return (y % 2 == 0) ? y * WIDTH + x : y * WIDTH + (WIDTH - 1 - x);
}

void setupMatrix() {
  for (int x = 0; x < WIDTH; x++)
    for (int y = 0; y < HEIGHT; y++)
      matrix[x][y] = &leds[XY(x, y)];
}

void clearMatrix() {
  fill_solid(leds, NUM_LEDS, CRGB::Black);
}
void drawCharRotated180(char c, int offsetY, CRGB color) {
  c = toupper(c);  // Zamień na wielką literę jeśli to możliwe

  int index;
  if (c >= '0' && c <= '9') {
    index = c - '0';  // 0-9 at index 0-9
  } else if (c >= 'A' && c <= 'Z') {
```

```
    index = 10 + (c - 'A');  // A–Z at index 10–35
  } else if (c == ':') {
    index = 36;  // ':' at index 36
  } else {
    index = 37;  // fallback to blank
  }

  for (int col = 0; col < 5; col++) {
    byte colData = font[index][4 - col];
    for (int row = 0; row < 7; row++) {
      if (colData & (1 << (6 - row))) {
        int x = row;
        int y = offsetY + col;
        if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT)
          *matrix[x][y] = color;
      }
    }
  }
}
void drawTextVerticalReversed(const char* t, int offsetY, CRGB color) {
  clearMatrix();
  int len = strlen(t);
  for (int i = 0; i < len; i++)
    drawCharRotated180(t[len - 1 - i], offsetY + i * 6, color);
  FastLED.show();
}

void drawScrollEffect() {
  if (millis() - lastEffect > 100) {
    lastEffect = millis();
    drawTextVerticalReversed(text.c_str(), scrollY++, scrollColor);
    if (scrollY > HEIGHT)
      scrollY = -((int)text.length() * 6);
  }
}

void staticPolizeiEffect() {
  drawTextVerticalReversed(staticText.c_str(), 0, staticColor);
}

void flashPanelEffect() {
  if (millis() - lastEffect > 300) {
    lastEffect = millis();
    CRGB color = blinkState ? CRGB::Red : CRGB::Blue;
    fill_solid(leds, NUM_LEDS, color);
    blinkState = !blinkState;
    FastLED.show();
  }
}

CRGB htmlColorToCRGB(const String& hex) {
```

```
  long number = strtol(hex.c_str() + 1, NULL, 16);  // Pomija #
  return CRGB((number >> 16) & 0xFF, (number >> 8) & 0xFF, number & 0xFF);
}

String getCurrentClock() {
  unsigned long secondsPassed = (millis() - timeSetMillis) / 1000;

  int h = manualTime.substring(0, 2).toInt();
  int m = manualTime.substring(3, 5).toInt();
  int s = manualTime.substring(6, 8).toInt();

  s += secondsPassed;
  if (s >= 60) { m += s / 60; s %= 60; }
  if (m >= 60) { h += m / 60; m %= 60; }
  if (h >= 24) h %= 24;

char buffer[6];
sprintf(buffer, "%02d:%02d", h, m);

  return String(buffer);
}

// --- Funkcja efektu zegara:
void drawClockEffect() {
  static unsigned long lastDraw = 0;
  if (millis() - lastDraw > 1000) {
    lastDraw = millis();
    String now = getCurrentClock();
    drawTextVerticalReversed(now.c_str(), 0, scrollColor);
  }
}

String formatTime(unsigned long seconds) {
  int m = seconds / 60;
  int s = seconds % 60;
  char buf[6];
  sprintf(buf, "%02d:%02d", m, s);
  return String(buf);
}

// --- Stoper ---
void drawStopwatchEffect() {
unsigned long elapsed = stopwatchElapsed;
if (stopwatchRunning) {
  elapsed += (millis() - stopwatchStart) / 1000;
}
String timeStr = formatTime(elapsed);
drawTextVerticalReversed(timeStr.c_str(), 0, CRGB::Green);
}

// --- Timer ---
```

```
void drawTimerEffect() {
  if (timerRunning) {
    unsigned long elapsed = (millis() - myTimerStart) / 1000;
    if (elapsed >= timerDuration) {
      timerRunning = false;
    }
  }
  unsigned long remaining = timerRunning ? (timerDuration - (millis() - myTimerStart) /
1000) : 0;
  String timeStr = formatTime(remaining);
  drawTextVerticalReversed(timeStr.c_str(), 0, CRGB::Red);
}
// --- Tabata ---
void drawTabataEffect() {
  if (!tabataRunning) return;
  unsigned long now = millis();
  unsigned long phaseTime = now - tabataStart;
  unsigned long duration = tabataWork ? tabataWorkSec * 1000 : tabataRestSec * 1000;

  if (phaseTime >= duration) {
    tabataWork = !tabataWork;
    tabataStart = now;
    if (!tabataWork) tabataRound++;
    if (tabataRound >= 8) {
      tabataRunning = false;
      tabataRound = 0;
    }
  }

  unsigned long remaining = (duration - (now - tabataStart)) / 1000;
  String label = tabataWork ? "WORK" : "REST";
  String timeStr = formatTime(remaining);
  drawTextVerticalReversed((label + " " + timeStr).c_str(), 0, tabataWork ? CRGB::Orange :
CRGB::Blue);
}



// --- Formularz HTML ---
void generateHtmlForm() {
  char staticHex[8], scrollHex[8];
  sprintf(staticHex, "#%02X%02X%02X", staticColor.r, staticColor.g, staticColor.b);
  sprintf(scrollHex, "#%02X%02X%02X", scrollColor.r, scrollColor.g, scrollColor.b);

  htmlForm = "<!DOCTYPE html><html><head><title>Panel LED</title><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">";
  htmlForm += R"rawliteral(
  <style>
    body {
      background-color: #121212;
      color: #ffffff;
      font-family: Arial, sans-serif;
```

```
      text-align: center;
      padding: 20px;
    }
    h2 {
      color: #00bcd4;
    }
    h3 {
      margin-top: 20px;
      color: #00acc1;
    }
    form {
      background-color: #1e1e1e;
      border-radius: 10px;
      padding: 20px;
      display: inline-block;
      box-shadow: 0 0 15px rgba(0, 188, 212, 0.4);
    }
    input[type="text"], input[type="number"], input[type="color"], input[type="time"] {
      width: 80%;
      padding: 10px;
      margin: 10px 0;
      border: none;
      border-radius: 5px;
      font-size: 16px;
    }
    input[type="radio"] {
      margin-right: 5px;
    }
    input[type="range"] {
      width: 80%;
    }
    input[type="color"] {
  appearance: none;
  width: 80%;
  height: 40px;
  border: 2px solid #00bcd4;
  border-radius: 5px;
  padding: 0;
}

    input[type="submit"] {
      background-color: #00bcd4;
      color: white;
      padding: 10px 20px;
      margin-top: 10px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      font-size: 16px;
    }
    input[type="submit"]:hover {
      background-color: #0097a7;
```

```
    }
    label {
      display: block;
      margin-top: 10px;
    }
  </style></head><body>
  <h2>Panel Sterowania LED</h2>
  <form action="/set" method="GET">
  )rawliteral";

// Static text
htmlForm += "<h3>Static text</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"0\"> Select</label>";
htmlForm += "<label for=\"static\">Text:</label>";
htmlForm += "<input type=\"text\" name=\"static\" value=\"" + staticText + "\">";
htmlForm += "<label for=\"staticColor\">Color:</label>";
htmlForm += "<input type=\"color\" name=\"staticColor\" value=\"" + String(staticHex) +
"\">";
htmlForm += "<input type=\"submit\" value=\"Apply\">";
htmlForm += "<hr>";

// Scroll text
htmlForm += "<h3>Scroll text</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"1\"> Select</label>";
htmlForm += "<label for=\"message\">Content:</label>";
htmlForm += "<input type=\"text\" name=\"message\" value=\"" + text + "\">";
htmlForm += "<label for=\"scrollColor\">Color:</label>";
htmlForm += "<input type=\"color\" name=\"scrollColor\" value=\"" + String(scrollHex) +
"\">";
htmlForm += "<input type=\"submit\" value=\"Apply\">";
htmlForm += "<hr>";

// Flash
htmlForm += "<h3>Flash</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"2\"> Select</label>";
htmlForm += "<input type=\"submit\" value=\"Apply\">";
htmlForm += "<hr>";

// Rainbow
htmlForm += "<h3>Rainbow</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"3\"> Select</label>";
htmlForm += "<input type=\"submit\" value=\"Apply\">";
htmlForm += "<hr>";

// Clock
htmlForm += "<h3>Clock</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"4\"> Select</label>";
htmlForm += "<label for=\"appt\">Set time:</label>";
htmlForm += "<input type=\"time\" name=\"appt\" value=\"" + manualTime.substring(0,5) +
"\">";
htmlForm += "<input type=\"submit\" value=\"Apply\">";
```

```
htmlForm += "<hr>";

// Stopwatch
htmlForm += "<h3>Stopwatch</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"5\">
Select</label><br>";
htmlForm += "<input type=\"submit\" name=\"stopwatch\" value=\"Start\">";
htmlForm += "<input type=\"submit\" name=\"stopwatch\" value=\"Stop\">";
htmlForm += "<input type=\"submit\" name=\"stopwatch\" value=\"Reset\">";
htmlForm += "<hr>";

// Timer
htmlForm += "<h3>Timer</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"6\"> Select</label>";
htmlForm += "<label for=\"timerMin\">Minutes: <span id=\"timerMinVal\">1</span></label>";
htmlForm += "<input type=\"range\" id=\"timerMin\" name=\"timerMin\" min=\"1\" max=\"60\"
value=\"1\" oninput=\"timerMinVal.innerText=this.value\">";
htmlForm += "<input type=\"hidden\" name=\"timerSec\" value=\"0\">";
htmlForm += "<br>";
htmlForm += "<input type=\"submit\" name=\"timerctrl\" value=\"Start\">";
htmlForm += "<input type=\"submit\" name=\"timerctrl\" value=\"Reset\">";
htmlForm += "<hr>";

// Tabata
htmlForm += "<h3>Tabata</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"7\"> Select</label>";
htmlForm += "<label>Work time (s): <span id=\"workVal\">20</span></label>";
htmlForm += "<input type=\"range\" name=\"tabataWork\" min=\"5\" max=\"60\" value=\"20\"
oninput=\"workVal.innerText=this.value\">";
htmlForm += "<label>Rest time (s): <span id=\"restVal\">10</span></label>";
htmlForm += "<input type=\"range\" name=\"tabataRest\" min=\"5\" max=\"60\" value=\"10\"
oninput=\"restVal.innerText=this.value\">";
htmlForm += "<input type=\"submit\" name=\"tabata\" value=\"Start\">";
htmlForm += "<input type=\"submit\" name=\"tabata\" value=\"Reset\">";
htmlForm += "<hr>";

// Off
htmlForm += "<h3>Turn off</h3>";
htmlForm += "<label><input type=\"radio\" name=\"effect\" value=\"8\"> Select</label>";
htmlForm += "<br><input type=\"submit\" value=\"Apply\">";

  htmlForm += "</form></body></html>";
}



uint8_t rainbowHue = 0;

void rainbowEffect() {
  EVERY_N_MILLISECONDS(30) {
    for (int y = 0; y < HEIGHT; y++) {
      CHSV color = CHSV(rainbowHue + y * 8, 255, 255);
```

```
      for (int x = 0; x < WIDTH; x++) {
        *matrix[x][y] = color;
      }
    }
    FastLED.show();
    rainbowHue++;
  }
}



void setup() {
  Serial.begin(115200);
  delay(1000);

  FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS);
  FastLED.setBrightness(BRIGHTNESS);
  setupMatrix();
  clearMatrix();
  FastLED.show();

  IPAddress local_IP(192, 168, 10, 1);
  IPAddress gateway(192, 168, 10, 1);
  IPAddress subnet(255, 255, 255, 0);
  WiFi.softAPConfig(local_IP, gateway, subnet);
  WiFi.softAP("PanelLED");

  Serial.println("Access Point uruchomiony!");
  Serial.println(WiFi.softAPIP());

  server.on("/", []() {
    generateHtmlForm();
    server.send(200, "text/html", htmlForm);
  });

  server.on("/set", []() {
    if (server.hasArg("effect")) {
      effectMode = server.arg("effect").toInt();
      scrollY = -((int)text.length() * 6);
    }
    if (server.hasArg("message")) text = server.arg("message");
    if (server.hasArg("static")) staticText = server.arg("static");
    if (server.hasArg("scrollColor")) scrollColor =
htmlColorToCRGB(server.arg("scrollColor"));
    if (server.hasArg("staticColor")) staticColor =
htmlColorToCRGB(server.arg("staticColor"));
 if (server.hasArg("appt")) {
  String t = server.arg("appt"); // format "HH:MM"
  if (t.length() == 5 && t.charAt(2) == ':') {
    manualTime = t + ":00";
    timeSetMillis = millis();
```

```
  }
}

if (server.hasArg("stopwatch")) {
  String cmd = server.arg("stopwatch");
if (cmd == "Start") {
  stopwatchStart = millis();
  stopwatchRunning = true;
}
else if (cmd == "Stop") {
  stopwatchElapsed += (millis() - stopwatchStart) / 1000;
  stopwatchRunning = false;
}
else if (cmd == "Reset") {
  stopwatchElapsed = 0;
  stopwatchRunning = false;
}

}

if (server.hasArg("timerMin")) {
  int min = server.arg("timerMin").toInt();
  timerDuration = min * 60;
}

// --- Handler formularza ---
if (server.hasArg("timerctrl")) {
  String cmd = server.arg("timerctrl");
  if (cmd == "Start") {
    myTimerStart = millis();
    timerRunning = true;
  } else if (cmd == "Reset") {
    timerRunning = false;
    myTimerStart = 0;
  }
}

if (server.hasArg("tabata")) {
  String cmd = server.arg("tabata");
  if (server.hasArg("tabataWork")) tabataWorkSec = server.arg("tabataWork").toInt();
  if (server.hasArg("tabataRest")) tabataRestSec = server.arg("tabataRest").toInt();

  if (cmd == "Start") {
    tabataRunning = true;
    tabataStart = millis();
    tabataRound = 0;
    tabataWork = true;
  } else if (cmd == "Reset") {
    tabataRunning = false;
    tabataRound = 0;
  }
```

```
}



    generateHtmlForm();
    server.send(200, "text/html", htmlForm);
  });

  server.begin();
  Serial.println("Serwer HTTP uruchomiony!");
}

void loop() {
  server.handleClient();

switch (effectMode) {
  case 0: staticPolizeiEffect(); break;        // Static text
  case 1: drawScrollEffect(); break;            // Scroll
  case 2: flashPanelEffect(); break;            // Flash
  case 3: rainbowEffect(); break;               // Rainbow
  case 4: drawClockEffect(); break;             // Manual Clock
  case 5: drawStopwatchEffect(); break;         // Stopwatch
  case 6: drawTimerEffect(); break;             // Timer
  case 7: drawTabataEffect(); break;            // Tabata
  case 8: clearMatrix(); FastLED.show(); break; // Off
}

}
```