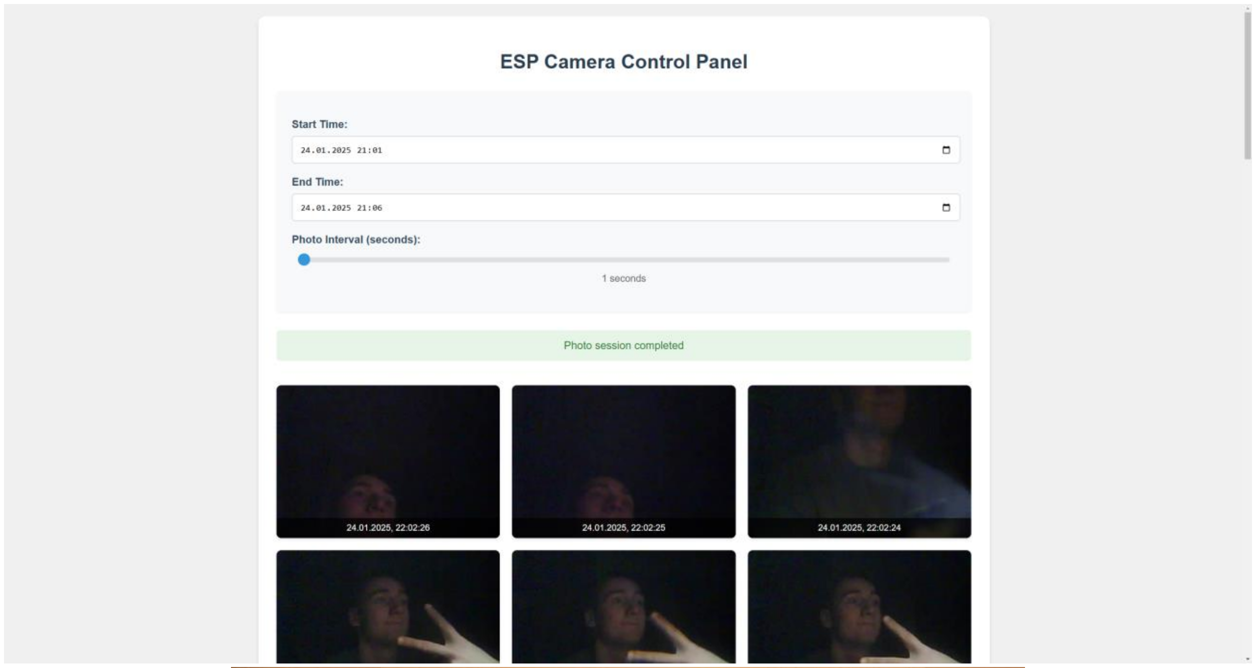
Artem Zakharov, grupa A1

Systemy wbudowane

Zadanie 8

2024



ESP Camera Control Panel

Start Time:

24.01.2025 21:01

End Time:

24.01.2025 21:06

Photo Interval (seconds):

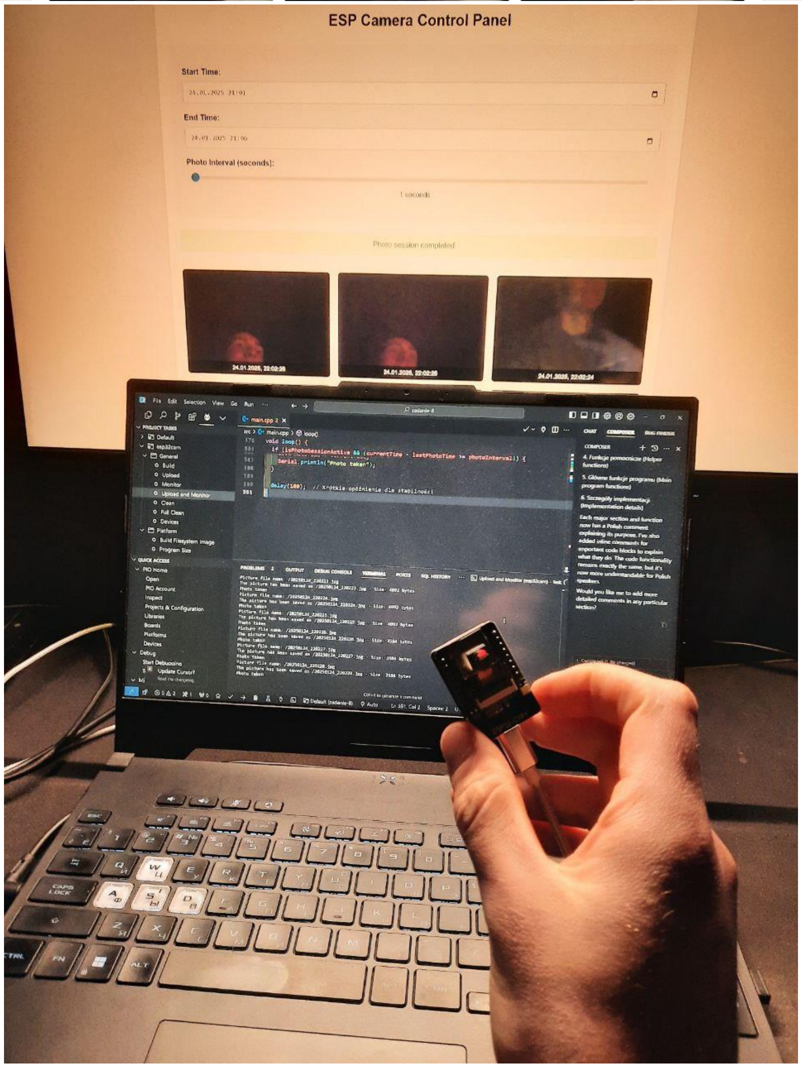
1 seconds

Photo session completed

24.01.2025, 22:02:26

24.01.2025, 22:02:25

24.01.2025, 22:02:24



```
void loop() {
  if (isPhotoSessionActive && (currentTime - lastPhotoTime >= photoInterval)) {
    Serial.println("Photo taken");
  }

  delay(100);  // Krótkie opóźnienie dla stabilności
```

Kod(zad-8): https://github.com/ZAKHAROV-Artem/systemy-wbudowane

```
#include <WiFiClient.h>

#include <WiFi.h>
#include <ESPmDNS.h>
#include <esp_camera.h>
#include <Arduino.h>
#include <esp_timer.h>
#include <FS.h>
#include "ESPAsyncWebServer.h"
#include "time.h"

#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"

#include <SD.h>
#include <SPIFFS.h>
#define DIODA 33



const char* ssid = "Сюда я";
const char* password = "lalalala";

AsyncWebServer server(80);  //użycie serwera asynchronicznego http na porcie 80

//prosta strona www z miejscem na obraz z kamery
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 1200px;
      margin: 0 auto;
      padding: 20px;
      background-color: #f0f0f0;
    }
    .container {
      background-color: white;
      padding: 30px;
    border-radius: 12px;
    box-shadow: 0 4px 6px rgba(0,0,0,0.1);
  }
  h2 {
    color: #2c3e50;
    text-align: center;
    margin-bottom: 30px;
    font-size: 2em;
  }
  .controls {
    margin: 30px 0;
    padding: 25px;
    background-color: #f8f9fa;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
  }
  .control-group {
    margin: 20px 0;
  }
  label {
    display: block;
    margin-bottom: 8px;
    color: #34495e;
    font-weight: bold;
    font-size: 1.1em;
  }
  input[type="datetime-local"] {
    width: calc(100% - 20px);
    padding: 12px;
    border: 2px solid #dde1e3;
    border-radius: 6px;
    font-size: 1.1em;
    transition: border-color 0.3s;
  }
  input[type="datetime-local"]:focus {
    border-color: #3498db;
    outline: none;
  }
  input[type="range"] {
    width: calc(100% - 20px);
    margin: 10px 0;
  }
  .status {
    text-align: center;
    margin: 20px 0;
```

```css
    padding: 15px;
    border-radius: 6px;
    font-size: 1.1em;
    font-weight: 500;
    background-color: #e8f5e9;
    color: #2e7d32;
  }
  .error {
    background-color: #ffebee;
    color: #c62828;
  }
  .photo-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 20px;
    padding: 20px 0;
  }
  .photo-item {
    position: relative;
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    background-color: #fff;
    transition: transform 0.3s;
  }
  .photo-item:hover {
    transform: translateY(-5px);
  }
  .photo-item img {
    width: 100%;
    height: 250px;
    object-fit: cover;
    display: block;
  }
  .photo-timestamp {
    position: absolute;
    bottom: 0;
    left: 0;
    right: 0;
    background: rgba(0,0,0,0.7);
    color: white;
    padding: 8px;
    font-size: 0.9em;
    text-align: center;
  }
```

```
    #interval-value {
      text-align: center;
      color: #666;
      margin-top: 8px;
      font-size: 1em;
    }
    .slider-container {
      padding: 0 10px;
    }
    input[type="range"] {
      -webkit-appearance: none;
      width: 100%;
      height: 8px;
      border-radius: 4px;
      background: #dde1e3;
      outline: none;
    }
    input[type="range"]::-webkit-slider-thumb {
      -webkit-appearance: none;
      width: 20px;
      height: 20px;
      border-radius: 50%;
      background: #3498db;
      cursor: pointer;
      transition: background .3s;
    }
    input[type="range"]::-webkit-slider-thumb:hover {
      background: #2980b9;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>ESP Camera Control Panel</h2>

    <div class="controls">
      <div class="control-group">
        <label for="start-time">Start Time:</label>
        <input type="datetime-local" id="start-time">
      </div>

      <div class="control-group">
        <label for="end-time">End Time:</label>
        <input type="datetime-local" id="end-time">
      </div>
      <div class="control-group">
        <label for="interval">Photo Interval (seconds):</label>
        <div class="slider-container">
          <input type="range" id="interval" min="1" max="60" value="10">
          <div id="interval-value">10 seconds</div>
        </div>
      </div>
    </div>

    <div class="status" id="status">Waiting to start...</div>

    <div class="photo-grid" id="photo-grid">
      <!-- Photos will be added here -->
    </div>
  </div>

  <script>
    const intervalSlider = document.getElementById('interval');
    const intervalValue = document.getElementById('interval-value');
    const startTime = document.getElementById('start-time');
    const endTime = document.getElementById('end-time');
    const photoGrid = document.getElementById('photo-grid');
    const statusElement = document.getElementById('status');

    let photos = new Set();
    let currentInterval = 10000; // Default 10 seconds

    function addNewPhoto(src) {
      const timestamp = new Date().toLocaleString();
      const photoItem = document.createElement('div');
      photoItem.className = 'photo-item';

      const img = new Image();
      img.src = src;

      const timestampDiv = document.createElement('div');
      timestampDiv.className = 'photo-timestamp';
      timestampDiv.textContent = timestamp;

      photoItem.appendChild(img);
      photoItem.appendChild(timestampDiv);

      if (photoGrid.firstChild) {
        photoGrid.insertBefore(photoItem, photoGrid.firstChild);
      } else {
        photoGrid.appendChild(photoItem);
      }

      photos.add(src);
    }

    // Sync time with server
    async function syncTime() {
      try {
        const response = await fetch('/get-time');
        const serverTime = await response.text();
        const now = new Date(serverTime);

        // Set min datetime for inputs to current time
        const minDateTime = now.toISOString().slice(0, 16);
        startTime.min = minDateTime;
        endTime.min = minDateTime;

        // Update status
        updateStatus();
      } catch (error) {
        console.error('Error syncing time:', error);
      }
    }

    function updateStatus() {
      const now = new Date();
      const start = startTime.value ? new Date(startTime.value) : null;
      const end = endTime.value ? new Date(endTime.value) : null;

      if (start && end) {
        if (now < start) {
          statusElement.textContent = 'Waiting for start time...';
          statusElement.classList.remove('error');
        } else if (now > end) {
          statusElement.textContent = 'Photo session completed';
          statusElement.classList.remove('error');
        } else {
          statusElement.textContent = 'Taking photos...';
          statusElement.classList.remove('error');
        }
      } else {
        statusElement.textContent = 'Please set start and end times';
        statusElement.classList.add('error');
    }
}

// Initialize
syncTime();
setInterval(syncTime, 60000); // Sync time every minute

// Set default times (current time and current time + 5 minutes)
async function setDefaultTimes() {
  try {
    const response = await fetch('/get-time');
    const serverTime = await response.text();
    const now = new Date(serverTime);
    const fiveMinutesLater = new Date(now.getTime() + 5 * 60000);

    startTime.value = now.toISOString().slice(0, 16);
    endTime.value = fiveMinutesLater.toISOString().slice(0, 16);

    // Trigger the change events to update server
    startTime.dispatchEvent(new Event('change'));
    endTime.dispatchEvent(new Event('change'));
  } catch (error) {
    console.error('Error setting default times:', error);
  }
}

// Call setDefaultTimes after page loads
setDefaultTimes();

intervalSlider.addEventListener('input', function() {
  const seconds = parseInt(this.value);
  intervalValue.textContent = seconds + ' seconds';
  currentInterval = seconds * 1000; // Convert to milliseconds
  fetch('/set-interval?value=' + seconds)
    .then(response => {
      if (!response.ok) {
        console.error('Failed to set interval');
      }
    });
});

startTime.addEventListener('change', function() {
  if (endTime.value && new Date(this.value) >= new Date(endTime.value)) {
    alert('Start time must be before end time');
    this.value = '';
```

```
        return;
      }
      fetch('/set-start-time?value=' + this.value);
      updateStatus();
    });

    endTime.addEventListener('change', function() {
      if (startTime.value && new Date(this.value) <= new Date(startTime.value)) {
        alert('End time must be after start time');
        this.value = '';
        return;
      }
      fetch('/set-end-time?value=' + this.value);
      updateStatus();
    });

    let photoTimer = null;

    function startPhotoTimer() {
      if (photoTimer) {
        clearInterval(photoTimer);
      }

      photoTimer = setInterval(() => {
        const now = new Date();
        const start = startTime.value ? new Date(startTime.value) : null;
        const end = endTime.value ? new Date(endTime.value) : null;

        if (start && end && now >= start && now <= end) {
          const photoUrl = 'fotka?' + new Date().getTime();
          addNewPhoto(photoUrl);
        }
      }, currentInterval);
    }

    // Start the timer initially
    startPhotoTimer();

    // Update timer when interval changes
    intervalSlider.addEventListener('change', function() {
      startPhotoTimer();
    });
  </script>
</body>
</html>)rawliteral";
```

```
// Konfiguracja serwera czasu NTP
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;  // Przesunięcie strefy czasowej (w sekundach)
const int   daylightOffset_sec = 3600;  // Korekta czasu letniego
struct tm timeinfo;

// Zmienne globalne do kontroli czasowej wykonywania zdjęć
unsigned long photoInterval = 10000; // Domyślny interwał - 10 sekund
String startTimeStr = "";  // Czas rozpoczęcia sesji
String endTimeStr = "";    // Czas zakończenia sesji
bool isPhotoSessionActive = false;  // Flaga aktywności sesji
unsigned long lastPhotoTime = 0;    // Czas ostatniego zdjęcia

// Funkcja konwertująca czas UTC na lokalny string czasowy
String getLocalTimeString() {
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return "";
  }
  char timeString[30];
  strftime(timeString, sizeof(timeString), "%Y-%m-%dT%H:%M:%S", &timeinfo);
  return String(timeString);
}

// Funkcja sprawdzająca poprawność zapisanego zdjęcia
bool checkPhoto( fs::FS &fs ) {
  File f_pic = fs.open("/photo.jpg");
  unsigned int pic_sz = f_pic.size();
  return ( pic_sz > 100 );
}

// Funkcja generująca znacznik czasowy dla nazwy pliku
String getTimestampString() {
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time for timestamp");
    return "unknown";
  }
  char timestamp[20];
  strftime(timestamp, sizeof(timestamp), "%Y%m%d_%H%M%S", &timeinfo);
  return String(timestamp);
}
```

```
// Funkcja sprawdzająca czy aktualny czas mieści się w zdefiniowanym zakresie
bool isTimeInRange() {
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    return false;
  }

  char currentTime[20];
  strftime(currentTime, sizeof(currentTime), "%Y-%m-%dT%H:%M", &timeinfo);

  // Compare current time with start and end time
  if (startTimeStr.length() > 0 && endTimeStr.length() > 0) {
    String currentTimeStr = String(currentTime);
    return (currentTimeStr >= startTimeStr && currentTimeStr <= endTimeStr);
  }

  return false;
}

// Funkcja wykonująca i zapisująca zdjęcie
void fotka() {
    camera_fb_t * fb = NULL;
    bool ok = 0;
    do {
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            continue;
        }
        String filename = "/" + getTimestampString() + ".jpg";
        Serial.printf("Picture file name: %s\n", filename.c_str());
        File file = SPIFFS.open(filename.c_str(), FILE_WRITE);
        if (!file) {
            Serial.println("Failed to open file in writing mode");
        } else {
            file.write(fb->buf, fb->len);
            Serial.print("The picture has been saved as ");
            Serial.print(filename);
            Serial.print(" - Size: ");
            Serial.print(file.size());
            Serial.println(" bytes");
        }
        file.close();
        esp_camera_fb_return(fb);
        ok = true;
    } while (!ok);
}

// Funkcja konfiguracyjna wywoływana jednorazowo przy starcie
void setup() {
  // Inicjalizacja komunikacji szeregowej
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  // Konfiguracja parametrów kamery
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0; //definicja portów, do których podłączona jes kamera
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // Inicjalizacja systemu plików SPIFFS
  if (!SPIFFS.begin(true)) {
    Serial.println("Błąd montowania systemu plików");
    ESP.restart();
  } else {
    Serial.println("SPIFFS mounted successfully");
    // Format SPIFFS if mounting failed
    if(!SPIFFS.format()) {
      Serial.println("SPIFFS format failed");
      ESP.restart();
    }
  }
  Serial.println("File system ready");

  // Inicjalizacja kamery
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Błąd inicjacji kamery numer: 0x%x", err);
    return;
  }
  Serial.printf("kamera ok");

  // Konfiguracja rozdzielczości sensora kamery
  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAMESIZE_QVGA);

  // Połączenie z siecią WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Połączono z WIFI");

  Serial.print("Kamera gotowa wejdź na adres: 'http://");
  Serial.println(WiFi.localIP());

  // Konfiguracja endpointów serwera HTTP
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });

  server.on("/fotka", HTTP_GET, [](AsyncWebServerRequest *request){
    String filename = "/" + getTimestampString() + ".jpg";
    request->send(SPIFFS, filename, "image/jpg");
  });
```

```
  server.on("/set-interval", HTTP_GET, [](AsyncWebServerRequest *request){
    if(request->hasParam("value")) {
      String intervalStr = request->getParam("value")->value();
      photoInterval = intervalStr.toInt() * 1000; // Convert to milliseconds
      request->send(200, "text/plain", "OK");
    }
  });

  server.on("/set-start-time", HTTP_GET, [](AsyncWebServerRequest *request){
    if(request->hasParam("value")) {
      startTimeStr = request->getParam("value")->value();
      request->send(200, "text/plain", "OK");
    }
  });

  server.on("/set-end-time", HTTP_GET, [](AsyncWebServerRequest *request){
    if(request->hasParam("value")) {
      endTimeStr = request->getParam("value")->value();
      request->send(200, "text/plain", "OK");
    }
  });

  server.on("/get-time", HTTP_GET, [](AsyncWebServerRequest *request){
    String localTime = getLocalTimeString();
    if (localTime.isEmpty()) {
      request->send(500, "text/plain", "Failed to obtain time");
      return;
    }
    request->send(200, "text/plain", localTime);
  });

  server.begin();

  // Inicjalizacja i konfiguracja czasu
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
}

// Główna pętla programu
void loop() {
  // Aktualny czas w milisekundach
  unsigned long currentTime = millis();

  // Sprawdzenie czy sesja fotograficzna jest aktywna
  isPhotoSessionActive = isTimeInRange();
  // Wykonanie zdjęcia jeśli spełnione są warunki czasowe
  if (isPhotoSessionActive && (currentTime - lastPhotoTime >= photoInterval)) {
    fotka();
    lastPhotoTime = currentTime;
    Serial.println("Photo taken");
  }

  delay(100);  // Krótkie opóźnienie dla stabilności
}
```