otlet.txt

Projektoldal:
- Projektötlethez az új file
        - a weboldal tetejére is kerüljön be: projektkód, disclaimer, a két arculati elem
Csapat: Interviewot beszúrni és a Brassai Eminescut helyettesíteni
Okosház összeszerelés: kidsblock segédlet és az összeszerelési videók (amit mi csináltunk)
Saját okosház: tervezés, 3d nyomtatás, vezél·rendszer.
Kézikönyv: feladatgy·jtemény (nincs még meg)

Hiányzik:
- feladatgyüjtemény
SZÖVEG:
A program python környezetben fut (letöltési link), app.py filial futtatható.

```
### ujak/smart-street/app.py ###

from flask import Flask, render_template, jsonify
from pymodbus.client import ModbusTcpClient
import threading, time

app = Flask(__name__)

IPS = ["192.168.191.211", "192.168.0.11", "192.168.0.12"]

houses = []

for ip in IPS:
    houses.append({
        "ip": ip,
        "client": ModbusTcpClient(host=ip, port=502, timeout=0.2),

        "led": False,
        "pir_led": False,
        "btn": False,
        "pir": False,

        "temp": 0,
        "hum": 0,
        "light": 0,
        "pot": 0,
        "dist": 0,

        "online": False,
        "fail": 0
    })


# · IMPULZUS (mint gomb)
def fast_write(i):
    h = houses[i]

    try:
        c = ModbusTcpClient(host=h["ip"], port=502, timeout=0.2)

        if not c.connect():
            return

        c.write_coil(address=2, value=True)
        time.sleep(0.05)
        c.write_coil(address=2, value=False)

        c.close()

    except:
        pass


def plc_cycle(i):

    h = houses[i]
    c = h["client"]

    while True:
        try:
            rr = c.read_coils(address=0, count=2)
            if rr is None or rr.isError():
                raise Exception()

            h["led"] = rr.bits[0]
            h["pir_led"] = rr.bits[1]

            rr = c.read_discrete_inputs(address=0, count=2)
            if rr is None or rr.isError():
                raise Exception()

            h["btn"] = rr.bits[0]
            h["pir"] = rr.bits[1]

            rr = c.read_holding_registers(address=0, count=5)
            if rr is None or rr.isError():
                raise Exception()
```

```
            regs = rr.registers

            h["temp"]  = regs[0] / 10
            h["hum"]   = regs[1] / 10
            h["light"] = round((regs[2] / 4095) * 100, 1)
            h["pot"]   = round((regs[3] / 4095) * 100, 1)
            h["dist"]  = regs[4]

            h["online"] = True
            h["fail"] = 0

        except:
            h["fail"] += 1
            if h["fail"] > 3:
                h["online"] = False

        time.sleep(0.2)


for i in range(len(houses)):
    threading.Thread(target=plc_cycle, args=(i,), daemon=True).start()


@app.route("/")
def index():
    return render_template("../../index.html")


@app.route("/status/<int:i>")
def status(i):
    h = houses[i]
    return jsonify({k:v for k,v in h.items() if k!="client"})


@app.route("/toggle/<int:i>", methods=["POST"])
def toggle(i):

    h = houses[i]

    if not h["online"]:
        return "OFFLINE", 400

    threading.Thread(target=fast_write, args=(i,), daemon=True).start()

    return "OK"


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

ujak/smart-street/ino.txt

```
#include <WiFi.h>
#include <ModbusTCP.h>
#include <DHT.h>
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// ================= MODBUS =================
ModbusTCP mb;

// ================= HARDVER =================
#define BTN 17
#define LED 2
#define PIR_PIN 16
#define PIR_LED 23

#define LIGHT_SENSOR 34
#define POT_PIN 35

#define DHTPIN 4
#define DHTTYPE DHT11

#define TRIG 18
#define ECHO 19

// · RGB LED
#define R_PIN 25
#define G_PIN 26
#define B_PIN 27

DHT dht(DHTPIN, DHTTYPE);
hd44780_I2Cexp lcd;

// ================= WIFI =================
const char* ssid = "plc";
const char* pass = "12345678";

IPAddress local_IP(192,168,0,11);
IPAddress gateway(192,168,0,1);
IPAddress subnet(255,255,255,0);

// ================= ÁLLAPOT =================
bool ledState = false;
bool buttonToggle = false;
bool lastButton = false;

bool pir = false;
bool pirLedState = false;

float temp = 0;
float hum = 0;
float distance = 0;

bool wifiConnected = false;

// virtuális gomb
bool lastVirtual = false;

// RGB sz·rés
float smoothDist = 0;

// ================= ID·ZÍTÉS =================
unsigned long lastDHT = 0;
unsigned long lastUltra = 0;
unsigned long lastLCD = 0;

// ================= SETUP =================
void setup() {

  Serial.begin(115200);

  pinMode(BTN, INPUT);
  pinMode(LED, OUTPUT);
  pinMode(PIR_PIN, INPUT);
  pinMode(PIR_LED, OUTPUT);
```

```
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);

  pinMode(R_PIN, OUTPUT);
  pinMode(G_PIN, OUTPUT);
  pinMode(B_PIN, OUTPUT);

  dht.begin();

  // LCD
  Wire.begin(21,22);
  lcd.begin(16,2);
  lcd.backlight();
  lcd.print("Smart Farm Node");
  delay(1500);
  lcd.clear();

  // WIFI
  WiFi.mode(WIFI_STA);
  WiFi.config(local_IP, gateway, subnet);
  WiFi.begin(ssid, pass);

  // MODBUS
  mb.server();

  mb.addCoil(0); // LED
  mb.addCoil(1); // PIR LED
  mb.addCoil(2); // virtuális gomb

  mb.addIsts(0);
  mb.addIsts(1);

  mb.addHreg(0);
  mb.addHreg(1);
  mb.addHreg(2);
  mb.addHreg(3);
  mb.addHreg(4);
}

// ================= LOOP =================
void loop() {

  mb.task();

  wifiConnected = (WiFi.status() == WL_CONNECTED);

  // ===== FIZIKAI GOMB =====
  bool reading = digitalRead(BTN);
  bool physicalEdge = (reading && !lastButton);
  lastButton = reading;

  // ===== VIRTUÁLIS GOMB =====
  bool virtualPress = mb.Coil(2);
  bool virtualEdge = (virtualPress && !lastVirtual);
  lastVirtual = virtualPress;

  // ===== GOMB LOGIKA =====
  if (physicalEdge || virtualEdge) {
    buttonToggle = !buttonToggle;
    ledState = buttonToggle;
  }

  if (virtualPress) {
    mb.Coil(2, false); // impulzus reset
  }

  // ===== ANALÓG =====
  int light = analogRead(LIGHT_SENSOR);
  int pot   = analogRead(POT_PIN);

  float lightPercent = (light / 4095.0) * 100;
  float potPercent   = (pot / 4095.0) * 100;

  // ===== PIR =====
  pir = digitalRead(PIR_PIN);

  // ===== AUTOMATA =====
  bool autoLight = (lightPercent < potPercent);
```

```
if (!(physicalEdge || virtualEdge)) {
  if (autoLight) ledState = true;
}

digitalWrite(LED, ledState);

// ===== PIR LED =====
pirLedState = pir ? (millis() % 500 < 250) : false;
digitalWrite(PIR_LED, pirLedState);

// ===== DHT =====
if (millis() - lastDHT > 2000) {
  lastDHT = millis();

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (!isnan(h) && !isnan(t)) {
    temp = t;
    hum = h;
  }
}

// ===== ULTRASONIC =====
if (millis() - lastUltra > 200) {

  lastUltra = millis();

  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);

  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  long duration = pulseIn(ECHO, HIGH, 30000);

  if (duration > 0) {
    distance = duration * 0.034 / 2;
  }
}

// ================================================
// · SIMA RGB SZÍNÁTMENET
// ================================================
smoothDist = smoothDist * 0.8 + distance * 0.2;

float norm = constrain(smoothDist / 200.0, 0.0, 1.0);

int r, g, b;

if (norm < 0.5) {
  float t = norm * 2.0;
  r = 255 * (1 - t);
  g = 255 * t;
  b = 0;
} else {
  float t = (norm - 0.5) * 2.0;
  r = 0;
  g = 255 * (1 - t);
  b = 255 * t;
}

analogWrite(R_PIN, r);
analogWrite(G_PIN, g);
analogWrite(B_PIN, b);

// ===== MODBUS =====
mb.Coil(0, ledState);
mb.Coil(1, pirLedState);

mb.Ists(0, reading);
mb.Ists(1, pir);

mb.Hreg(0, (int)(temp * 10));
mb.Hreg(1, (int)(hum * 10));
mb.Hreg(2, light);
```

```
  mb.Hreg(3, pot);
  mb.Hreg(4, (int)distance);

  // ===== LCD =====
  if (millis() - lastLCD > 500) {

    lastLCD = millis();

    lcd.setCursor(0,0);
    lcd.print(wifiConnected ? "ONLINE  " : "OFFLINE ");

    lcd.setCursor(0,1);
    lcd.print("T:");
    lcd.print(temp,0);
    lcd.print(" H:");
    lcd.print(hum,0);
    lcd.print(" D:");
    lcd.print((int)distance);
  }

  delay(5);
}
```

ujak/smart-street/static/script.js

```
function drawHouse(i,d){
let r=Math.max(0,Math.min(255,255-d.dist*1.2));
let b=Math.max(0,Math.min(255,d.dist*1.2));
return `<h2>· Ház ${i+1}</h2>
<div class="status ${d.online?'on':'off'}">${d.online?'ONLINE':'OFFLINE'}</div>
<div class="row">· H·mérséklet: ${d.temp} °C</div>
<div class="row">· Páratartalom: ${d.hum} %</div>
<div class="row">· Távolság: ${d.dist} cm</div>
<div class="row">· Fény: ${d.light} %</div>
<div class="row">· Potméter: ${d.pot} %</div>
<div class="row">· PIR: ${d.pir?'Mozgás':'Nincs'}</div>
<div class="row">· Gomb: ${d.btn?'Nyomva':'Nincs'}</div>
<div class="row center">· LED<div class="led" style="background:${d.led?'limegreen':'red'}"></div></div>
<div class="row center">· PIR LED<div class="led"
style="background:${d.pir_led?'yellow':'#555'}"></div></div>
<div class="row center">· RGB<div class="rgb" style="background:rgb(${r},0,${b})"></div></div>
<button onclick="toggle(${i})" ${!d.online?'disabled':''}>LED kapcsolás</button>`}

function update(){for(let i=0;i<3;i++){fetch(`/status/${i}`).then(r=>r.json()).then(d=>{document.getEleme
ntById("house"+i).innerHTML=drawHouse(i,d);});}}
function toggle(i){fetch(`/toggle/${i}`,{method:"POST"});}
setInterval(update,300);
```

ujak/smart-street/static/style.css

```
body{margin:0;font-family:Arial;background:linear-
gradient(135deg,#0f2027,#203a43,#2c5364);color:white;text-align:center}
.container{display:flex;justify-content:center;flex-wrap:wrap;gap:20px}
.card{background:#222;padding:20px;width:320px;border-radius:15px;box-shadow:0 0 10px black}
.status.on{color:lime}.status.off{color:red}
.row{background:#2b2b2b;margin:6px 0;padding:8px;border-radius:8px}
.center{text-align:center}
.led{width:30px;height:30px;border-radius:50%;margin:10px auto}
.rgb{width:40px;height:40px;border-radius:50%;margin:10px auto}
button{margin-top:10px;padding:8px;border:none;border-radius:8px;background:#00aaff;color:white}
```