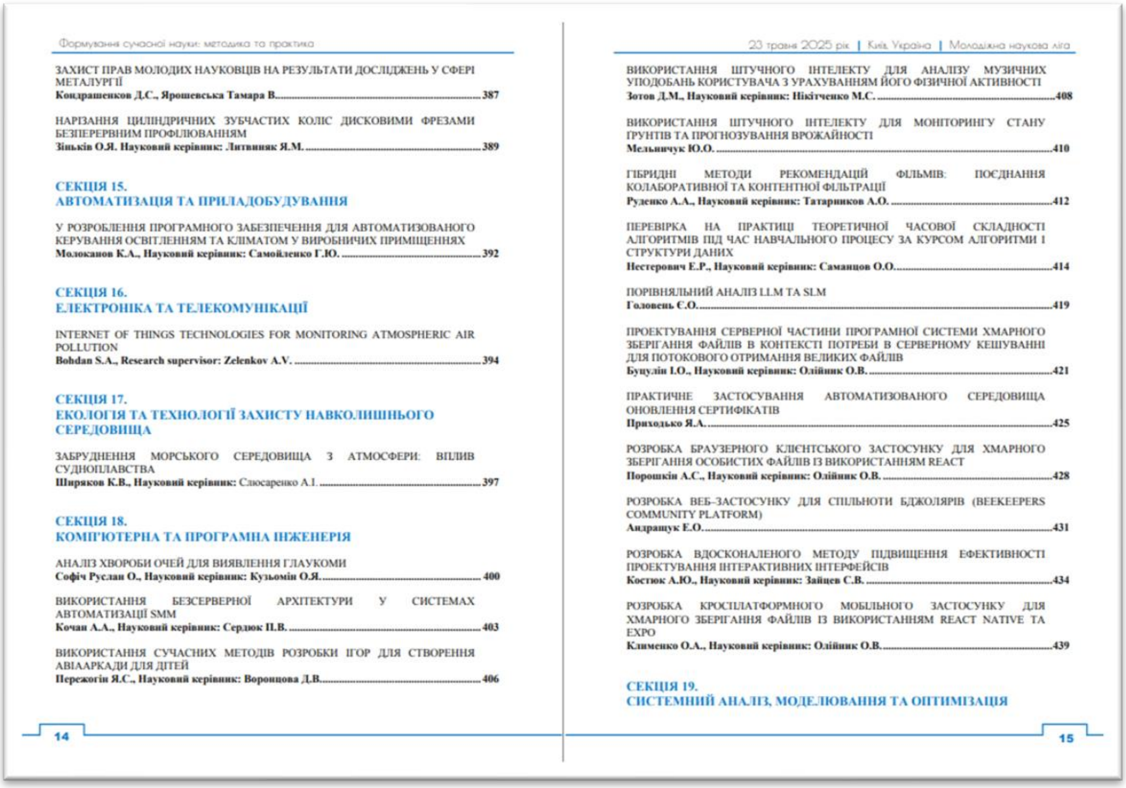
ДОДАТОК А

Апробація результатів роботи

Рисунок А.1 – Титульний аркуш електронного видання



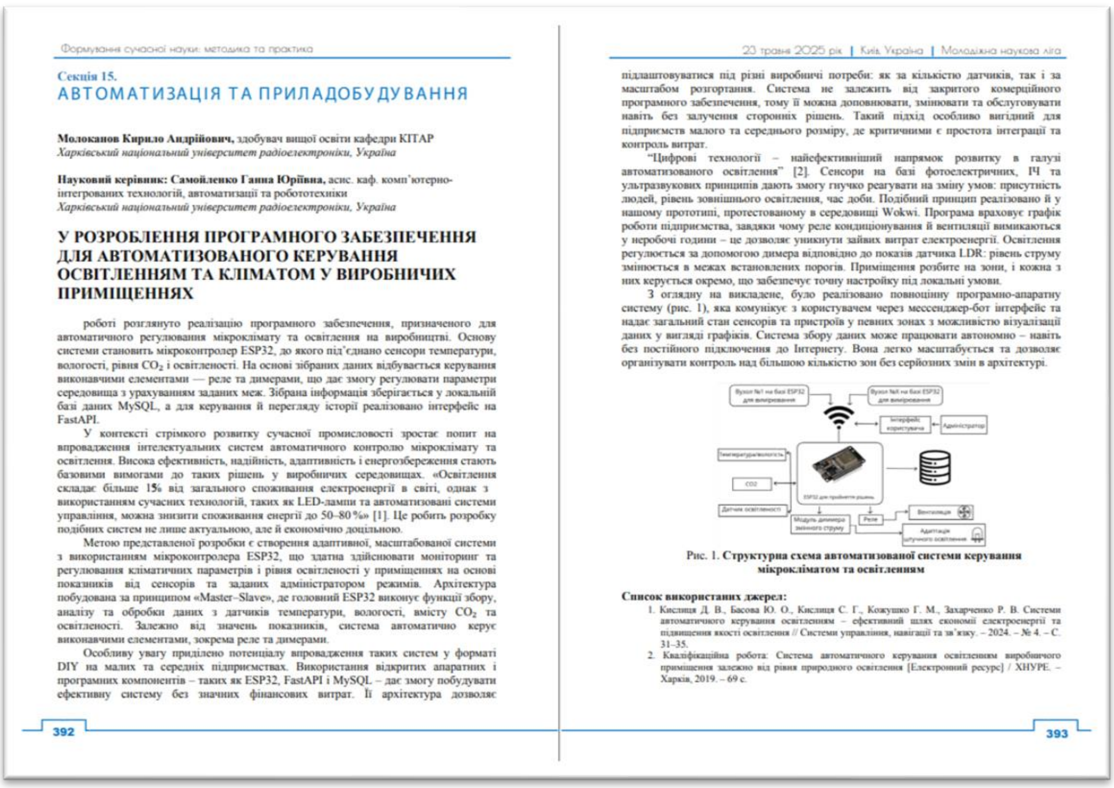
Формування сучасної науки: методика та практика

14

23 травня 2025 рік | Київ, Україна | Молодіжна наукова ліга

15

Рисунок А.2 – Зміст конференції



Формування сучасної науки: методика та практика

Секція 15.
АВТОМАТИЗАЦІЯ ТА ПРИЛАДОБУДУВАННЯ

Молоканов Кирило Андрійович, здобувач вищої освіти кафедри КІТАР
Харківський національний університет радіоелектроніки, Україна

Науковий керівник: Самойленко Ганна Юріївна, асис. каф. комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
Харківський національний університет радіоелектроніки, Україна

У РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО КЕРУВАННЯ ОСВІТЛЕННЯМ ТА КЛІМАТОМ У ВИРОБНИЧИХ ПРИМІЩЕННЯХ

роботі розглянуто реалізацію програмного забезпечення, призначеного для автоматичного регулювання мікроклімату та освітлення на виробництві. Основу системи становить мікроконтролер ESP32, до якого під'єднано сенсори температури, вологості, рівня CO_2 і освітленості. На основі зібраних даних відбувається керування виконавчими елементами — реле та димерами, що дає змогу регулювати параметри середовища з урахуванням заданих меж. Зібрана інформація зберігається у локальній базі даних MySQL, а для керування й перегляду історії реалізовано інтерфейс на FastAPI.

У контексті стрімкого розвитку сучасної промисловості зростає попит на впровадження інтелектуальних систем автоматичного контролю мікроклімату та освітлення. Висока ефективність, надійність, адаптивність і енергозбереження стають базовими вимогами до таких рішень у виробничих середовищах. «Освітлення складає більше 15% від загального споживання електроенергії в світі, однак з використанням сучасних технологій, таких як LED-лампи та автоматизовані системи управління, можна знизити споживання енергії до 50–80 %» [1]. Це робить розробку подібних систем не лише актуальною, але й економічно доцільною.

Метою представленої розробки є створення адаптивної, масштабованої системи з використанням мікроконтролера ESP32, що здатна здійснювати моніторинг та регулювання кліматичних параметрів і рівня освітленості у приміщеннях на основі показників від сенсорів та заданих адміністратором режимів. Архітектура побудована за принципом «Master–Slave», де головний ESP32 виконує функції збору, аналізу та обробки даних з датчиків температури, вологості, вмісту CO_2 та освітленості. Залежно від значень показників, система автоматично керує виконавчими елементами, зокрема реле та димерами.

Особливу увагу приділено потенціалу впровадження таких систем у форматі DIY на малих та середніх підприємствах. Використання відкритих апаратних і програмних компонентів – таких як ESP32, FastAPI і MySQL – дає змогу побудувати ефективну систему без значних фінансових витрат. Її архітектура дозволяє

392

23 травня 2025 рік | Київ, Україна | Молодіжна наукова ліга

підлаштовуватися під різні виробничі потреби: як за кількістю датчиків, так і за масштабом розгортання. Система не залежить від закритого комерційного програмного забезпечення, тому її можна доповнювати, змінювати та обслуговувати навіть без залучення сторонніх рішень. Такий підхід особливо вигідний для підприємств малого та середнього розміру, де критичними є простота інтеграції та контроль витрат.

"Цифрові технології – найефективніший напрямок розвитку в галузі автоматизованого освітлення" [2]. Сенсори на базі фотоелектричних, ІЧ та ультразвукових принципів дають змогу гнучко реагувати на зміну умов: присутність людей, рівень зовнішнього освітлення, час доби. Подібний принцип реалізовано й у нашому прототипі, протестованому в середовищі Wokwi. Програма враховує графік роботи підприємства, завдяки чому реле кондиціонування й вентиляції вимикаються у неробочі години – це дозволяє уникнути зайвих витрат електроенергії. Освітлення регулюється за допомогою димера відповідно до показів датчика LDR: рівень струму змінюється в межах встановлених порогів. Приміщення розбите на зони, і кожна з них керується окремо, що забезпечує точну настройку під локальні умови.

З огляду на викладене, було реалізовано повноцінну програмно-апаратну систему (рис. 1), яка комунікує з користувачем через мессенджер-бот інтерфейс та надає загальний стан сенсорів та пристроїв у певних зонах з можливістю візуалізації даних у вигляді графіків. Система збору даних може працювати автономно – навіть без постійного підключення до Інтернету. Вона легко масштабується та дозволяє організувати контроль над більшою кількістю зон без серйозних змін в архітектурі.

Рис. 1. **Структурна схема автоматизованої системи керування мікрокліматом та освітленням**

Список використаних джерел:

1. Кислиця Д. В., Басова Ю. О., Кислиця С. Г., Кожушко Г. М., Захарченко Р. В. Системи автоматичного керування освітленням – ефективний шлях економії електроенергії та підвищення якості освітлення // Системи управління, навігації та зв'язку. – 2024. – № 4. – С. 31–35.
2. Кваліфікаційна робота: Система автоматичного керування освітленням виробничого приміщення залежно від рівня природного освітлення [Електронний ресурс] / ХНУРЕ. – Харків, 2019. – 69 с.

393

Рисунок А.3 – Опублікована стаття

ДОДАТОК Б

Код мессенджер-бота

```
import logging
import requests
import sqlite3
import asyncio
import os
from dotenv import load_dotenv
import matplotlib.pyplot as plt
from io import BytesIO
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder, CommandHandler, MessageHandler, ContextTypes,
    filters
)
from config import TELEGRAM_TOKEN, ALLOWED_USERS
from datetime import datetime
import pytz

from db import get_latest_data, log_user_override

ZONE_STATUS_URL = "http://127.0.0.1:8000/api/status"
MASTER_DATA_URL = "http://127.0.0.1:8000/api/last-sensor-data"
OVERRIDE_URL = "http://127.0.0.1:8000/api/override"

ROOMS = ["🏠 Приміщення №1"]
MAIN_MENU = [["📊 Статус", "⚙️ Керування"], ["📈 Графіки"], ["📂 Дані з БД"], ["↩️ Назад"]]
LIGHT_MENU = [["💡 Зона 1", "💡 Зона 2", "💡 Зона 3"], ["↩️ Назад"]]
GRAPH_MENU = [["📈 Температура за день"], ["💧 Вологість за день", "🔥 CO₂ за день"], ["↩️ Назад"]]
DB_MENU = [["📋 Журнал керування"], ["📊 Середні показники"], ["📂 Кількість записів"], ["↩️ Назад"]]
DIMMING_MENU = [["0%", "25%", "50%", "75%", "100%"], ["↩️ Назад"]]
DURATION_MENU = [["1 хвилина", "2 хвилини"], ["↩️ Назад"]]
RELAY_MENU = [["🌀 Вентиляція", "❄️ Кондиціонер"], ["↩️ Назад"]]

load_dotenv()
SECRET_CODE = os.getenv("SECRET_CODE")
```

```
# ──────────────────────────────────────────────────────────────
# АВТОРИЗАЦІЯ І ДОСТУП
# ──────────────────────────────────────────────────────────────

def is_authorized(user_id: int) -> bool:
    return user_id in ALLOWED_USERS

# ──────────────────────────────────────────────────────────────
# ЧАС І СТАТУС
# ──────────────────────────────────────────────────────────────

def format_time_kyiv(ts: str) -> str:
    try:
        utc_dt = datetime.fromisoformat(ts.replace("Z", "+00:00"))
        kyiv_dt = utc_dt.astimezone(pytz.timezone("Europe/Kyiv"))
        return kyiv_dt.strftime("%H:%M:%S %d.%m.%Y")
    except:
        return ts

def get_last_db_update_time():
    try:
        conn = sqlite3.connect("sensors.db")
        cursor = conn.cursor()
        cursor.execute("SELECT timestamp FROM sensor_data ORDER BY id DESC LIMIT 1")
        ts = cursor.fetchone()
        conn.close()
        if ts:
            kyiv_dt = datetime.fromisoformat(ts[0]).astimezone(pytz.timezone("Europe/Kyiv"))
            return kyiv_dt.strftime("📦 Останнє оновлення БД: %H:%M:%S %d.%m.%Y")
    except:
        return "⚠️ Не вдалося отримати час оновлення БД"
def fetch_full_status():
    status = ""
    try:
        resp = safe_get(ZONE_STATUS_URL)
        is_work = resp.get("is_work_time", False)
        status += "⚡ Робочий час\n" if is_work else "🌙 Неробочий час\n"
    except:
        status += "🌐 Статус режиму недоступний\n"
```

```
    try:
        master = safe_get(MASTER_DATA_URL)
        if "error" not in master:
            now_kyiv = datetime.now(pytz.timezone("Europe/Kyiv")).strftime("%H:%M:%S %d.%m.%Y")
            status += (
                "\n📡 Дані від Master ESP:\n"
                f"🌡 Температура: {master['temperature']}°C\n"
                f"💧 Вологість: {master['humidity']}%\n"
                f"🔥 CO₂: {master['co2']} ppm\n"
            )
            if not is_work:
                status += "💡 Автоматичне керування вимкнене — енергозбереження\n"
            status += (
                f"❄️ Кондиціонер: {'🟢 (вручну)' if master.get('aircon_manual') else '🔵 авто'} - {'увімкнено' if master['relay_aircon'] else 'вимкнено'}\n"
                f"🌀 Вентиляція: {'🟢 (вручну)' if master.get('vent_manual') else '🔵 авто'} - {'увімкнено' if master['relay_ventilation'] else 'вимкнено'}\n"
                f"🕒 Поточний час: {now_kyiv}\n"
            )
            status += "\n" + get_last_db_update_time()
    except:
        status += "⚠️ Дані Master ESP недоступні\n"

    try:
        zones = resp.get("zones", [])
        status += "\n💡 Зони освітлення:\n"
        for z in zones:
            m = " (вручну)" if z["manual"] else ""
            online = "🟢" if z["online"] else "🔴"
            status += f"{z['zone']}: {z['light_level']} lx | {z['dimming']}%{m} {online}\n"
    except:
        status += "⚠️ Дані зон недоступні\n"

    return status

# ————————————————————————————————————————————
# HTTP-ЗАПИТИ
# ————————————————————————————————————————————
```

```
def safe_get(url):
    try:
        return requests.get(url).json()
    except Exception as e:
        logging.warning(f"⚠ GET {url} → {e}")
        return {}

def safe_post(url, params=None, json=None):
    try:
        return requests.post(url, params=params, json=json)
    except Exception as e:
        logging.warning(f"⚠ POST {url} → {e}")
        return None

# ────────────────────────────────────────────────
# ГЕНЕРАЦІЯ ГРАФІКІВ
# ────────────────────────────────────────────────
def generate_plot(param: str, label: str, title_prefix: str):
    conn = sqlite3.connect("sensors.db")
    cursor = conn.cursor()
    today = datetime.now(pytz.timezone("Europe/Kyiv")).strftime("%Y-%m-%d")
    cursor.execute(f"""
        SELECT timestamp, {param} FROM sensor_data
        WHERE DATE(timestamp) = ?
        ORDER BY timestamp
    """, (today,))
    rows = cursor.fetchall()
    conn.close()

    if not rows:
        return None

    times = [datetime.fromisoformat(ts).strftime("%H:%M") for ts, _ in rows]
    values = [v for _, v in rows]
    date_str = datetime.fromisoformat(rows[0][0]).strftime("%d.%m.%Y")

    with plt.style.context("seaborn-v0_8-pastel"):
        fig, ax = plt.subplots(figsize=(9, 4.5))
        ax.plot(times, values, marker="o", linestyle="-", linewidth=1.8)
```

```
        ax.set_title(f"{title_prefix} за день: {date_str}", fontsize=13)
        ax.set_xlabel("Час", fontsize=11)
        ax.set_ylabel(label, fontsize=11)
        ax.grid(True)
        plt.xticks(rotation=45)
        plt.tight_layout()

        buffer = BytesIO()
        plt.savefig(buffer, format="png")
        buffer.seek(0)
        plt.close()
        return buffer

def generate_temperature_plot():
    conn = sqlite3.connect("sensors.db")
    cursor = conn.cursor()
    today = datetime.now(pytz.timezone("Europe/Kyiv")).strftime("%Y-%m-%d")
    cursor.execute("""
        SELECT timestamp, temperature FROM sensor_data
        WHERE DATE(timestamp) = ?
        ORDER BY timestamp
    """, (today,))
    rows = cursor.fetchall()
    conn.close()

    if not rows:
        return None

    times = [datetime.fromisoformat(ts).strftime("%H:%M") for ts, _ in rows]
    temps = [t for _, t in rows]
    date_str = datetime.fromisoformat(rows[0][0]).strftime("%d.%m.%Y")

    with plt.style.context("seaborn-v0_8-pastel"):
        fig, ax = plt.subplots(figsize=(9, 4.5))
        ax.plot(times, temps, marker="o", linestyle="-", color="#003366", linewidth=1.8)
        ax.set_title(f"Температура за день: {date_str}", fontsize=13)
        ax.set_xlabel("Час", fontsize=11)
        ax.set_ylabel("Температура (°C)", fontsize=11)
        ax.grid(True)
        plt.xticks(rotation=45)
        plt.tight_layout()
```

```
        buffer = BytesIO()
        plt.savefig(buffer, format="png")
        buffer.seek(0)
        plt.close()
        return buffer

# ──────────────────────────────────────────────
# ЛОГІКА БОТА
# ──────────────────────────────────────────────

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if not is_authorized(update.effective_user.id):
        await update.message.reply_text("⛔ Доступ заборонено.")
        return
    keyboard = ReplyKeyboardMarkup([["▶️ Старт"]], resize_keyboard=True)
    await update.message.reply_text("Натисніть ▶️ Старт, щоб почати", reply_markup=keyboard)

async def temp_graph(update: Update, context: ContextTypes.DEFAULT_TYPE):
    plot = generate_temperature_plot()
    if plot:
        await update.message.reply_photo(photo=plot, caption="📈 Температурний графік за сьогодні")
    else:
        await update.message.reply_text("📭 Немає даних температури за сьогодні.")
async def humidity_graph(update: Update, context: ContextTypes.DEFAULT_TYPE):
    plot = generate_plot("humidity", "Вологість (%)", "💧 Вологість")
    if plot:
        await update.message.reply_photo(photo=plot, caption="Графік вологості за сьогодні")
    else:
        await update.message.reply_text("📭 Немає даних вологості за сьогодні.")

async def co2_graph(update: Update, context: ContextTypes.DEFAULT_TYPE):
    plot = generate_plot("co2", "CO₂ (ppm)", "🔥 CO₂")
    if plot:
        await update.message.reply_photo(photo=plot, caption="Графік CO₂ за сьогодні")
    else:
        await update.message.reply_text("📭 Немає даних CO₂ за сьогодні.")
```

```
async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    msg = update.message.text
    uid = update.effective_user.id

    if not is_authorized(uid):
        await update.message.reply_text("⛔ Доступ заборонено.")
        return

    if msg == "▶️ Старт":
        await update.message.reply_text("Оберіть приміщення:", reply_markup=ReplyKeyboardMarkup([ROOMS], resize_keyboard=True))
        return

    if msg == "🏠 Приміщення №1":
        context.user_data.clear()
        context.user_data["zone"] = "room1"
        await update.message.reply_text("✅ Обрано: Приміщення №1", reply_markup=ReplyKeyboardMarkup(MAIN_MENU, resize_keyboard=True))
        return

    if msg == "↩️ Назад":
        await update.message.reply_text("⬅️ Назад до меню:", reply_markup=ReplyKeyboardMarkup([ROOMS], resize_keyboard=True))
        return

    if msg == "📊 Статус":
        await update.message.reply_text(fetch_full_status())
        return

    if msg == "📈 Графіки":
        await update.message.reply_text("Оберіть графік:", reply_markup=ReplyKeyboardMarkup(GRAPH_MENU, resize_keyboard=True))
        return

    if msg == "📈 Температура за день":
        await temp_graph(update, context)
        return
```

```
    if msg == "💧 Вологість за день":
        await humidity_graph(update, context)
        return

    if msg == "🔥 CO₂ за день":
        await co2_graph(update, context)
        return

    if msg == "📁 Дані з БД":
        await update.message.reply_text("Оберіть дію:", reply_markup=ReplyKeyboardMarkup(DB_MENU,
resize_keyboard=True))
        return

    if msg == "📋 Журнал керування":
        conn = sqlite3.connect("sensors.db")
        cursor = conn.cursor()
        cursor.execute("""
            SELECT timestamp, user_id, target, action, duration_sec
            FROM user_override_log
            ORDER BY id DESC LIMIT 5
        """)
        rows = cursor.fetchall()
        conn.close()

        if not rows:
            await update.message.reply_text("⛔ Журнал порожній.")
            return

        text = "📋 Останні дії користувачів:\n\n"
        for ts, uid, target, act, dur in rows:
            t = datetime.fromisoformat(ts).astimezone(pytz.timezone("Europe/Kyiv")).strftime("%H:%M:%S %d.%m.%Y")
            text += f"👤 {uid}\n🎯 {target} → {act} на {dur // 60} хв.\n🕒 {t}\n\n"

        await update.message.reply_text(text.strip())
        return

    if msg == "📊 Середні показники":
        conn = sqlite3.connect("sensors.db")
```

```
    cursor = conn.cursor()
    today = datetime.now(pytz.timezone("Europe/Kyiv")).strftime("%Y-%m-%d")
    cursor.execute("""
        SELECT
            ROUND(AVG(temperature), 2),
            ROUND(MAX(temperature), 2),
            ROUND(MIN(temperature), 2),
            ROUND(AVG(humidity), 2),
            ROUND(MAX(humidity), 2),
            ROUND(AVG(co2), 2),
            ROUND(MAX(co2), 2)
        FROM sensor_data
        WHERE DATE(timestamp) = ?
    """, (today,))
    row = cursor.fetchone()
    conn.close()

    if not row or row[0] is None:
        await update.message.reply_text("📭 Дані відсутні за сьогодні.")
        return

    avg_temp, max_temp, min_temp, avg_hum, max_hum, avg_co2, max_co2 = row
    await update.message.reply_text(
        f"📊 Статистика за {today}:\n\n"
        f"🌡 Температура:\n"
        f"   ↳ Середня: {avg_temp}°C\n"
        f"   ↳ Макс: {max_temp}°C\n"
        f"   ↳ Мін: {min_temp}°C\n\n"
        f"💧 Вологість:\n"
        f"   ↳ Середня: {avg_hum}%\n"
        f"   ↳ Макс: {max_hum}%\n\n"
        f"🔥 CO₂:\n"
        f"   ↳ Середнє: {avg_co2} ppm\n"
        f"   ↳ Макс: {max_co2} ppm"
    )
    return



if msg == "📁 Кількість записів":
```

```
        conn = sqlite3.connect("sensors.db")
        cursor = conn.cursor()
        cursor.execute("SELECT name, seq FROM sqlite_sequence")
        rows = cursor.fetchall()
        conn.close()

        if not rows:
            await update.message.reply_text("📭 Немає інформації про записи.")
            return

        text = "📁 Кількість записів у таблицях:\n\n"
        for name, seq in rows:
            text += f"📌 {name}: {seq} записів\n"
        await update.message.reply_text(text.strip())
        return

    if msg == "⚙️ Керування":
        if not context.user_data.get("auth", False):
            await update.message.reply_text("🔐 Введіть пароль для доступу до керування")
            context.user_data["awaiting_password"] = True
            return
        await update.message.reply_text("Оберіть дію:", reply_markup=ReplyKeyboardMarkup([["💡 Освітлення", "🔌 Реле"], ["↩️ Назад"]], resize_keyboard=True))
        return

    if context.user_data.get("awaiting_password"):
        if msg == SECRET_CODE:
            context.user_data["auth"] = True
            context.user_data.pop("awaiting_password")
            await update.message.reply_text("🔓 Доступ дозволено. Оберіть дію:", reply_markup=ReplyKeyboardMarkup([["💡 Освітлення", "🔌 Реле"], ["↩️ Назад"]], resize_keyboard=True))
        else:
            await update.message.reply_text("❌ Невірний пароль.")
        return
    if msg == "💡 Освітлення":
        await update.message.reply_text("Оберіть зону:", reply_markup=ReplyKeyboardMarkup(LIGHT_MENU, resize_keyboard=True))
        return
```

```
    if msg.startswith("💡 Зона"):
        zone = int(msg.split()[-1])
        context.user_data["control_zone"] = zone
        await update.message.reply_text(f"🔆 Зона {zone}. Встановіть рівень освітлення:",
reply_markup=ReplyKeyboardMarkup(DIMMING_MENU, resize_keyboard=True))
        return

    if msg.endswith("%"):
        try:
            level = int(msg.strip("%"))
            context.user_data["pending_level"] = level
            context.user_data["pending_action"] = "light"
            await update.message.reply_text("⏱ Оберіть тривалість:",
reply_markup=ReplyKeyboardMarkup(DURATION_MENU, resize_keyboard=True))
        except:
            await update.message.reply_text("⚠️ Помилка параметрів.")
        return

    if msg == "🔌 Реле":
        await update.message.reply_text("Оберіть реле:", reply_markup=ReplyKeyboardMarkup(RELAY_MENU,
resize_keyboard=True))
        return

    if msg in ["🌀 Вентиляція", "❄️ Кондиціонер"]:
        context.user_data["pending_action"] = "relay"
        context.user_data["pending_relay"] = "ventilation" if "Вентиляція" in msg else "aircon"
        await update.message.reply_text("⏱ Оберіть тривалість:",
reply_markup=ReplyKeyboardMarkup(DURATION_MENU, resize_keyboard=True))
        return

    if msg.startswith("1 хвилина") or msg.startswith("2 хвилини"):
        duration = 60 if "1" in msg else 120
        action = context.user_data.get("pending_action")

        if action == "light":
            level = context.user_data.get("pending_level")
            zone = context.user_data.get("control_zone", 1)
            res = requests.post(f"{OVERRIDE_URL}/light", params={
                "zone": zone,
                "level": level,
```

```
            "duration_sec": duration,
            "user_id": uid
        })
        if res.status_code == 200:
            await update.message.reply_text(f"🔧 Освітлення в зоні {zone} встановлено на {level}% на
{duration//60} хв.")
        else:
            await update.message.reply_text("❌ Помилка встановлення освітлення.")
        return

    elif action == "relay":
        target = context.user_data.get("pending_relay")
        res = requests.post(f"{OVERRIDE_URL}/relay", params={
            "target": target,
            "state": "on",
            "duration_sec": duration,
            "user_id": uid
        })
        if res.status_code == 200:
            await update.message.reply_text(f"🔧 {'🌀 Вентиляція' if target == 'ventilation' else '❄️ Кондиціонер'}
увімкнено вручну на {duration//60} хв.")
        else:
            await update.message.reply_text("❌ Помилка увімкнення реле.")
        return

    await update.message.reply_text(
        "🤖 Команду не розпізнано. Виберіть дію з меню:",
        reply_markup=ReplyKeyboardMarkup(MAIN_MENU, resize_keyboard=True)
    )
# ────────────────────────────────────────────────────────────
# МОНИТОРИНГ ЗВ’ЯЗКУ
# ────────────────────────────────────────────────────────────
async def monitor_slave_connection(app):
    notified = False
    while True:
        await asyncio.sleep(30)
        try:
            resp = safe_get("http://127.0.0.1:8000/api/status")
            now = datetime.now(pytz.utc)
            zones = resp.get("zones", [])
```

```
            online = any(z["online"] for z in zones)
            if not online and not notified:
                for uid in ALLOWED_USERS:
                    await app.bot.send_message(chat_id=uid, text="🚨 *Увага!* Slave ESP не надсилає дані!",
parse_mode="Markdown")
                notified = True
            elif online:
                notified = False
        except Exception as e:
            logging.error(f"[ESP Monitor] Помилка: {e}")

async def show_main_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("📋 Головне меню:", reply_markup=ReplyKeyboardMarkup(MAIN_MENU,
resize_keyboard=True))
# ─────────────────────────────────────────────────────────────
# БОТ: ГОЛОВНА ФУНКЦІЯ
# ─────────────────────────────────────────────────────────────
def main():
    logging.basicConfig(level=logging.INFO)
    app = ApplicationBuilder().token(TELEGRAM_TOKEN).build()
    app.add_handler(CommandHandler("menu", show_main_menu))
    app.job_queue.run_once(lambda *_: asyncio.create_task(monitor_slave_connection(app)), 0)
    app.add_handler(CommandHandler("start", start))
    app.add_handler(CommandHandler("temp_graph", temp_graph))
    app.add_handler(CommandHandler("humidity_graph", humidity_graph))
    app.add_handler(CommandHandler("co2_graph", co2_graph))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))
    app.run_polling()

if __name__ == "__main__":
    main()
```

ДОДАТОК В

Код серверної частини FastApi

main.py

```
from fastapi import FastAPI
from contextlib import asynccontextmanager
from db import init_db
from routes import router
import asyncio
import logging

logging.basicConfig(level=logging.INFO)

@asynccontextmanager
async def lifespan(app: FastAPI):
    logging.info("🛠 Ініціалізація бази даних...")
    init_db()
    logging.info("🤖 Запуск Telegram-бота...")
    yield

# Ініціалізація FastAPI з життєвим циклом
app = FastAPI(lifespan=lifespan)
# Підключення всіх маршрутів
app.include_router(router)

# (опційно) Кореневий маршрут для перевірки
@app.get("/")
def read_root():
    return {"status": "API is running"}
```

routes.py

```
        from fastapi import APIRouter, Query
from pydantic import BaseModel, Field
import sqlite3
from datetime import datetime, timedelta, timezone
import pytz
```

```
import logging
import json
import requests

from typing import Optional
from threshold import TEMP_MIN, TEMP_MAX, CO2_THRESHOLD, HUMIDITY_THRESHOLD

router = APIRouter()
DB_NAME = "sensors.db"
ZONE_TIMEOUT = 30
MASTER_ESP_URL = "http://127.0.0.1:8001"  # адреса майстер ESP API

zone_dimming = {1: 0, 2: 0, 3: 0}
zone_last_change = {1: datetime.utcnow(), 2: datetime.utcnow(), 3: datetime.utcnow()}
zone_override_until = {}
relay_override = {"ventilation": None, "aircon": None}
esp_last_seen = {}

import os
BASE_DIR = os.path.dirname(os.path.abspath(__file__))

with open(os.path.join(BASE_DIR, "work_schedule.json"), "r", encoding="utf-8") as f:
    WORK_SCHEDULE = json.load(f)

def now_utc():
    return datetime.utcnow().replace(tzinfo=timezone.utc)

def is_work_time():
    kyiv = datetime.now(pytz.timezone("Europe/Kyiv"))
    weekday = kyiv.isoweekday()
    if weekday not in WORK_SCHEDULE["workdays"]:
        return False
    start = datetime.strptime(WORK_SCHEDULE["start"], "%H:%M").time()
    end = datetime.strptime(WORK_SCHEDULE["end"], "%H:%M").time()
    return start <= kyiv.time() <= end

class SensorData(BaseModel):
    temperature: float = Field(..., ge=-50, le=100)
    humidity: float = Field(..., ge=0, le=100)
    co2: int = Field(..., ge=0, le=5000)
    relay_ventilation: Optional[bool] = None
```

```
    relay_aircon: Optional[bool] = None

class ZoneData(BaseModel):
    zone: int
    light_level: int
    timestamp: str

class PingData(BaseModel):
    esp_id: int
    timestamp: str

@router.post("/api/update")
async def update_sensor_data(data: SensorData):
    now = now_utc()

    if not is_work_time():
        logging.info("⛔ Неробочий час — автоматичне керування реле заблоковано")
        data.relay_aircon = False
        data.relay_ventilation = False
    else:
        data.relay_aircon = True if relay_override["aircon"] and now < relay_override["aircon"] else data.temperature > TEMP_MAX or data.temperature < TEMP_MIN
        data.relay_ventilation = True if relay_override["ventilation"] and now < relay_override["ventilation"] else data.co2 > CO2_THRESHOLD or data.humidity > HUMIDITY_THRESHOLD

    from db import insert_data
    insert_data(data.dict())

    return {
        "status": "ok",
        "relay_aircon": data.relay_aircon,
        "relay_ventilation": data.relay_ventilation
    }

@router.post("/api/update-zone")
async def update_zone_data(data: ZoneData):
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS zone_data (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            zone INTEGER,
            light_level INTEGER,
            dimming_level INTEGER,
            timestamp TEXT
        )
    """)

    lux = data.light_level
    if lux < 100:
        new_level = 100
    elif lux < 150:
        new_level = 75
    elif lux < 200:
        new_level = 50
    elif lux < 250:
        new_level = 25
    else:
        new_level = 0

    if not is_work_time():
        logging.info(f"💡 Неробочий час — освітлення зони {data.zone} вимкнено (енергозбереження)")
        new_level = 0

    if data.zone in zone_override_until and now_utc() < zone_override_until[data.zone]:
        new_level = zone_dimming[data.zone]
    else:
        zone_override_until.pop(data.zone, None)

    cursor.execute("""
        INSERT INTO zone_data (zone, light_level, dimming_level, timestamp)
        VALUES (?, ?, ?, ?)
    """, (data.zone, lux, new_level, data.timestamp))
    conn.commit()
    conn.close()

    zone_dimming[data.zone] = new_level
    zone_last_change[data.zone] = now_utc()

@router.post("/api/esp-ping")
def ping_slave(data: PingData):
    esp_last_seen[data.esp_id] = datetime.fromisoformat(data.timestamp)
```

```
    return {"status": "pong", "esp_id": data.esp_id}

@router.get("/api/status")
def get_zone_status():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT zone, light_level, dimming_level, timestamp
        FROM zone_data
        WHERE (zone, timestamp) IN (
            SELECT zone, MAX(timestamp)
            FROM zone_data
            GROUP BY zone
        )
    """)
    rows = cursor.fetchall()
    conn.close()

    now = now_utc()
    zone_names = {1: "Зона 1", 2: "Зона 2", 3: "Зона 3"}
    status = []

    esp_online = True
    for esp_id, last_seen in esp_last_seen.items():
        if (now - last_seen.replace(tzinfo=timezone.utc)).total_seconds() > ZONE_TIMEOUT:
            esp_online = False
            logging.warning(f"❗ Втрачено зв’язок із ESP ID={esp_id}")
        else:
            logging.debug(f"✅ ESP {esp_id} онлайн")

    for zone, lux, dimming, ts in rows:
        last_seen = datetime.fromisoformat(ts).astimezone(pytz.timezone("Europe/Kyiv"))
        ts_dt = datetime.fromisoformat(ts)
        if ts_dt.tzinfo is None:
            ts_dt = ts_dt.replace(tzinfo=timezone.utc)
        offline = (now - ts_dt).total_seconds() > ZONE_TIMEOUT
        manual = zone in zone_override_until and now < zone_override_until[zone]
        status.append({
            "zone": zone_names.get(zone, f"Зона {zone}"),
            "light_level": lux,
```

```
            "dimming": dimming,
            "last_seen": last_seen.strftime("%H:%M:%S %d.%m.%Y"),
            "online": not offline,
            "manual": manual
        })
    lighting_dict = {f"zone{z['zone'][-1]}": z["dimming"] for z in status}

    return {
        "zones": status,
        "is_work_time": is_work_time(),
        "esp_online": esp_online,
        "lighting": lighting_dict  # 👈 додаємо
    }


@router.get("/api/last-sensor-data")
def get_latest_sensor_data():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT temperature, humidity, co2,
               relay_ventilation, relay_aircon, timestamp
        FROM sensor_data
        ORDER BY id DESC LIMIT 1
    """)
    row = cursor.fetchone()
    conn.close()

    if not row:
        return {"error": "Немає даних"}

    now = now_utc()
    vent_manual = relay_override["ventilation"] and now < relay_override["ventilation"]
    aircon_manual = relay_override["aircon"] and now < relay_override["aircon"]

    return {
        "temperature": row[0],
        "humidity": row[1],
        "co2": row[2],
        "relay_ventilation": True if vent_manual else bool(row[3]),
        "relay_aircon": True if aircon_manual else bool(row[4]),
```

```
        "timestamp": row[5],
        "vent_manual": vent_manual,
        "aircon_manual": aircon_manual
    }
@router.post("/api/set-light-level")
def set_light_level(zone: str = Query(...), level: int = Query(...)):
    zone_map = {"zone1": 1, "zone2": 2, "zone3": 3}
    zone_num = zone_map.get(zone)
    if not zone_num:
        return {"error": "невідома зона"}

    until = now_utc() + timedelta(seconds=60)
    zone_override_until[zone_num] = until
    zone_dimming[zone_num] = level

    from db import log_user_override
    log_user_override(488400832, zone, f"{level}%", 60)

    return {"status": "ok", "zone": zone, "level": level, "until": until.isoformat()}

@router.post("/api/override/light")
def override_light(zone: int = Query(...), level: int = Query(...), duration_sec: int = 60, user_id: int = 0):
    until = now_utc() + timedelta(seconds=duration_sec)
    zone_override_until[zone] = until
    zone_dimming[zone] = level

    from db import log_user_override
    log_user_override(user_id, f"zone_{zone}", f"{level}%", duration_sec)

    return {"status": "ok", "zone": zone, "until": until.isoformat()}

@router.post("/api/override/relay")
def override_relay(target: str = Query(...), state: str = Query(...), duration_sec: int = 60, user_id: int = 0):
    until = now_utc() + timedelta(seconds=duration_sec)
    if target in relay_override:
        relay_override[target] = until

        from db import log_user_override, insert_data, get_latest_data
        log_user_override(user_id, target, state, duration_sec)

        # 🧠 Оновлення останніх даних у БД для відображення в боті
```

```
    last_data = get_latest_data()
    if last_data:
        if target == "ventilation":
            last_data["relay_ventilation"] = True
        elif target == "aircon":
            last_data["relay_aircon"] = True
        insert_data(last_data)

    return {"status": "ok", "target": target, "until": until.isoformat()}
return {"error": "невірний параметр"}
```

ДОДАТОК Г

Код симуляторів Master та Slave ESP32

master_sim.py

```
from fastapi import FastAPI, Request
from pydantic import BaseModel
import requests

app = FastAPI()

API_UPDATE_URL = "http://127.0.0.1:8000/api/update"
API_ZONE_URL = "http://127.0.0.1:8000/api/update-zone"

# 💬 Приймає повний пакет даних від slave ESP
@app.post("/api/from-slave")
async def receive_from_slave(req: Request):
    data = await req.json()
    print(f"📥 Отримано від slave: {data}")

    # 🔶 Надсилання сенсорних даних
    try:
        sensor_data = {
            "temperature": data["temperature"],
            "humidity": data["humidity"],
            "co2": data["co2"],
        }
        r1 = requests.post(API_UPDATE_URL, json=sensor_data)
        print(f"➡️ Sensor API статус: {r1.status_code}")
    except Exception as e:
        print(f"❌ Помилка надсилання сенсорних даних: {e}")

    # 🔷 Надсилання зон
    for z in data.get("zones", []):
        try:
            zone_data = {
                "zone": z["zone"],
```

```
                "light_level": z["light_level"],
                "timestamp": data["timestamp"]
            }
            r2 = requests.post(API_ZONE_URL, json=zone_data)
            print(f"➡️ Zone {z['zone']} API статус: {r2.status_code}")
        except Exception as e:
            print(f"❌ Помилка зони {z['zone']}: {e}")

    return {"status": "ok"}

# 📦 Модель для relay override
class RelayOverride(BaseModel):
    target: str
    state: str
    duration_sec: int
    user_id: int

# ⚙️ Обробка запиту на override реле з боку сервера
@app.post("/api/relay-override")
async def relay_override(payload: RelayOverride):
    print(f"🔧 Запит на реле: {payload.target} → {payload.state.upper()} на {payload.duration_sec} сек.")
    # Тут може бути логіка керування GPIO, або просто зберігання стану
    return {"status": "accepted", "target": payload.target, "state": payload.state, "duration": payload.duration_sec}
```

slave_sim.py

```
import requests
import time
import random
from datetime import datetime, timezone
from routes import is_work_time

MASTER_URL = "http://127.0.0.1:8001/api/from-slave"
ZONES = [1, 2, 3]

def now_iso():
    return datetime.now(timezone.utc).isoformat()

def simulate_lux(): return random.randint(50, 300)
```

```
def simulate_temp(): return round(random.uniform(18, 26), 1)
def simulate_humidity(): return random.randint(40, 90)
def simulate_co2(): return random.randint(400, 900)

def main():
    print("📡 Slave ESP запущено. Передача всіх даних до Master ESP...\n")
    while True:
        payload = {
            "timestamp": now_iso(),
            "temperature": simulate_temp(),
            "humidity": simulate_humidity(),
            "co2": simulate_co2(),
            "zones": [
                {"zone": z, "light_level": simulate_lux()} for z in ZONES
            ]
        }

        try:
            res = requests.post(MASTER_URL, json=payload)
            print(f"📤 Дані відправлено: {payload} | Статус: {res.status_code}")
        except Exception as e:
            print(f"❌ Помилка відправки: {e}")

        # Інтервал залежно від робочого часу
        interval = 5 if is_work_time() else 15
        time.sleep(interval)

if __name__ == "__main__":
    main()
```

ДОДАТОК Д
Код роботи з базою даних

```
import sqlite3
from datetime import datetime
import pytz

DB_NAME = "sensors.db"

def init_db():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS sensor_data (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp TEXT,
            temperature REAL,
            humidity REAL,
            co2 INTEGER,
            relay_ventilation INTEGER,
            relay_aircon INTEGER
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS zone_data (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            zone INTEGER,
            light_level INTEGER,
            dimming_level INTEGER,
            timestamp TEXT
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS user_override_log (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
```

```
            target TEXT,
            action TEXT,
            duration_sec INTEGER,
            timestamp TEXT
        )
    """)

    conn.commit()
    conn.close()

def insert_data(data: dict):
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO sensor_data (
            timestamp, temperature, humidity, co2,
            relay_ventilation, relay_aircon
        ) VALUES (?, ?, ?, ?, ?, ?)
    """, (
        datetime.now(pytz.timezone("Europe/Kyiv")).isoformat(),
        data["temperature"],
        data["humidity"],
        data["co2"],
        int(data["relay_ventilation"]),
        int(data["relay_aircon"])
    ))
    conn.commit()
    conn.close()

def get_latest_data():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT temperature, humidity, co2,
               relay_ventilation, relay_aircon, timestamp
        FROM sensor_data
        ORDER BY id DESC LIMIT 1
    """)
    row = cursor.fetchone()
    conn.close()
```

```
    if row:
        return {
            "temperature": row[0],
            "humidity": row[1],
            "co2": row[2],
            "relay_ventilation": bool(row[3]),
            "relay_aircon": bool(row[4]),
            "timestamp": row[5]
        }
    return None

def log_user_override(user_id: int, target: str, action: str, duration_sec: int):
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO user_override_log (user_id, target, action, duration_sec, timestamp)
        VALUES (?, ?, ?, ?, ?)
    """, (
        user_id,
        target,
        action,
        duration_sec,
        datetime.now(pytz.timezone("Europe/Kyiv")).isoformat()
    ))
    conn.commit()
    conn.close()
```

ДОДАТОК Е

Код альтернативного інтерфейсу користувача

```
from fastapi import FastAPI, Request, Form
from fastapi.responses import HTMLResponse, RedirectResponse
import requests

app = FastAPI()

@app.get("/", response_class=HTMLResponse)
async def dashboard():
    try:
        data = requests.get("http://127.0.0.1:8000/api/last-sensor-data", timeout=3).json()
        status = requests.get("http://127.0.0.1:8000/api/status", timeout=3).json()
        lighting = status.get("lighting", {})
        work_status = "🟢 Робочий час" if status.get("is_work_time") else "🔴 Неробочий час"
    except:
        return HTMLResponse("""
        <html>
        <head>
            <style>
                body { font-family: Arial, sans-serif; background: #fce4ec; text-align: center; padding: 40px; }
                h2 { color: #b71c1c; }
                .error-box { padding: 20px; border: 2px solid #b71c1c; border-radius: 8px; background: #ffebee; display: inline-block; }
            </style>
        </head>
        <body>
            <div class='error-box'>
                <h2>❌ API недоступне</h2>
                <p>Спробуйте пізніше...</p>
            </div>
        </body>
        </html>
        """, status_code=500)

    html = f"""
    <html>
```

```
<head>
  <style>
    body {{ font-family: Arial, sans-serif; background: #f7f5f2; padding: 40px; }}
    h2 {{ color: #c75c5c; margin-top: 40px; margin-left: 10px; }}
    ul {{ list-style: none; padding: 0; }}
    li {{ background: #ffffff; padding: 12px; margin: 8px 0; border-radius: 10px; box-shadow: 0 1px 3px
rgba(0,0,0,0.05); }}
    .container {{ max-width: 650px; margin: auto; padding: 0 10px; }}
    form {{ background: #ffffff; padding: 24px; border-radius: 12px; margin-top: 36px; box-shadow: 0 2px 5px
rgba(0,0,0,0.04); }}
    select, input, button {{
      padding: 10px;
      margin-top: 14px;
      width: 100%;
      font-size: 16px;
      border-radius: 6px;
      border: 1px solid #ccc;
    }}
    label {{
      display: block;
      margin-bottom: 6px;
      margin-top: 20px;
      font-weight: 500;
    }}
    button {{ background: #81c784; color: white; border: none; cursor: pointer; }}
    button:hover {{ background: #66bb6a; }}
    .divider {{ margin-top: 50px; }}
  </style>
</head>
<body>
  <div class="container">
    <h2>🌡 Поточні дані</h2>
    <p><strong>Статус:</strong> {work_status}</p>
    <ul>
      <li><strong>Температура:</strong> {data.get("temperature", "н/д")} °C</li>
      <li><strong>Вологість:</strong> {data.get("humidity", "н/д")} %</li>
      <li><strong>CO₂:</strong> {data.get("co2", "н/д")} ppm</li>
    </ul>

    <div class="divider"></div>
```

```
<h2>🔧 Керування реле</h2>
<form action="/control" method="post">
  <label for="target">Пристрій:</label>
  <select name="target">
    <option value="ventilation">Вентиляція</option>
    <option value="aircon">Кондиціонер</option>
  </select>
  <label for="duration">Тривалість:</label>
  <select name="duration">
    <option value="60">1 хвилина</option>
    <option value="120" selected>2 хвилини</option>
  </select>
  <button type="submit">Увімкнути</button>
</form>

<div class="divider"></div>
<h2>💡 Освітлення</h2>
<ul>
  <li><strong>Зона 1:</strong> {lighting.get("zone1", "н/д")}%</li>
  <li><strong>Зона 2:</strong> {lighting.get("zone2", "н/д")}%</li>
  <li><strong>Зона 3:</strong> {lighting.get("zone3", "н/д")}%</li>
</ul>
<form action="/light-control" method="post">
  <label for="zone">Оберіть зону:</label>
  <select name="zone">
    <option value="zone1">Зона 1</option>
    <option value="zone2">Зона 2</option>
    <option value="zone3">Зона 3</option>
  </select>
  <label for="level">Рівень освітлення:</label>
  <select name="level">
    <option value="0">0%</option>
    <option value="25">25%</option>
    <option value="50">50%</option>
    <option value="75">75%</option>
    <option value="100">100%</option>
  </select>
  <label for="duration">Тривалість:</label>
  <select name="duration">
    <option value="60">1 хвилина</option>
```

```
                <option value="120" selected>2 хвилини</option>
            </select>
            <button type="submit">Застосувати</button>
        </form>

        <div class="divider"></div>
        <p><a href="/status">📊 Переглянути JSON-статус</a></p>
    </div>
</body>
</html>
"""
    return HTMLResponse(html)

@app.post("/control")
async def control(request: Request):
    form = await request.form()
    target = form.get("target")
    duration = int(form.get("duration", 120))
    try:
        requests.post("http://127.0.0.1:8000/api/override/relay", params={
            "target": target,
            "state": "on",
            "duration_sec": duration,
            "user_id": 488400832
        }, timeout=3)
    except:
        pass
    return RedirectResponse("/", status_code=302)

@app.post("/light-control")
async def light_control(
    zone: str = Form(...),
    level: int = Form(...),
    duration: int = Form(...),
):
    try:
        requests.post("http://127.0.0.1:8000/api/override/light", params={
            "zone": int(zone[-1]),  # zone1 -> 1
            "level": level,
            "duration_sec": duration,
            "user_id": 488400832
```

```
        })

    except:
        pass
    return RedirectResponse("/", status_code=302)

@app.get("/status", response_class=HTMLResponse)
async def status():
    try:
        status_data = requests.get("http://127.0.0.1:8000/api/status", timeout=3).json()
        html = f"<pre>{status_data}</pre>"
    except:
        html = "<h2>❌ Статус недоступний</h2>"
    return HTMLResponse(html)
```

ДОДАТОК Є

Конфігураційні файли порогових значень та робочого графіку

threshold.py

```
# Пороги температури (°C)
TEMP_MIN = 18
TEMP_MAX = 26
# --- Порогові значення вологості ---
HUMIDITY_THRESHOLD = 70  # %
# Поріг CO₂ (ppm)
CO2_THRESHOLD = 700

# Пороги освітленості (lx) для зон
LIGHT_THRESHOLDS = [
    (100, 100),
    (150, 75),
    (200, 50),
    (250, 25),
    (float("inf"), 0),
]
```

config.py

```
from dotenv import load_dotenv
import os
load_dotenv('api.env')
TELEGRAM_TOKEN = os.getenv("TELEGRAM_TOKEN")
# ✅ Список дозволених user_id
ALLOWED_USERS = [488400832]  # @pluxurynip
```

work_schedule.json

```
{
  "workdays": [1, 2, 3, 4, 5],
```

"start": "5:30",
"end": "16:00"
}

ДОДАТОК Ж

Приклад коду розробленого макету у середовищі WokWi із можливістю адаптування під реальні пристрої

###Master ESP32

```
from machine import Pin, I2C, ADC
import dht
import time
from oled import SSD1306_I2C
import urequests  #
import ujson

# --- GPIO для виконавчих пристроїв ---
relay_vent = Pin(18, Pin.OUT)
led_vent = Pin(17, Pin.OUT)

relay_ac = Pin(23, Pin.OUT)
led_ac = Pin(19, Pin.OUT)

led_light = Pin(16, Pin.OUT)

# --- Сенсори ---
dht_sensor = dht.DHT22(Pin(13))
mq2 = ADC(Pin(32))
mq2.atten(ADC.ATTN_11DB)

ldr = ADC(Pin(15))
ldr.atten(ADC.ATTN_11DB)

# --- OLED ---
i2c = I2C(0, scl=Pin(22), sda=Pin(21))
oled = SSD1306_I2C(128, 64, i2c)

# --- Порогові значення ---
TEMP_MIN = 18
TEMP_MAX = 26
HUMIDITY_THRESHOLD = 70
```

```
CO2_THRESHOLD = 700

# --- FastAPI URLs ---
# API_URL = "http://192.168.1.100:8000/api/update"      Надсилання даних
# STATUS_URL = "http://192.168.1.100:8000/api/status"    Отримання статусу

# --- Основний цикл ---
while True:
  try:
    # Зчитування сенсорів
    dht_sensor.measure()
    temp = dht_sensor.temperature()
    hum = dht_sensor.humidity()
    gas_adc = mq2.read()
    gas_ppm = int(gas_adc * 0.21)
    light = ldr.read()

    # Формування даних для API
    payload = {
      "temperature": temp,
      "humidity": hum,
      "co2": gas_ppm
    }

    # Надсилання показників на FastAPI
    try:
      res = urequests.post(API_URL, json=payload)
      print("📡 Надіслано на API:", res.status_code)
    except Exception as e:
      print("❌ Помилка надсилання на API:", e)

    # Отримання override-статусу з API
    # try:
      # status = urequests.get(STATUS_URL).json()
      # vent_override = status.get("relay_ventilation", False)
      # ac_override = status.get("relay_aircon", False)
    # except Exception as e:
      # print("⚠️ Неможливо отримати статус. Вмикаємо авто-режим.")
      # vent_override = gas_ppm > CO2_THRESHOLD or hum > HUMIDITY_THRESHOLD
      # ac_override = temp < TEMP_MIN or temp > TEMP_MAX
```

```
        # Керування вентиляцією
        relay_vent.value(1 if vent_override else 0)
        led_vent.value(1 if vent_override else 0)

        # Керування кондиціонером
        relay_ac.value(1 if ac_override else 0)
        led_ac.value(1 if ac_override else 0)

        # Локальне керування освітленням
        led_light.value(1 if light < 2500 else 0)

        # Вивід на OLED
        oled.fill(0)
        oled.text(f"T:{temp}C H:{hum}%", 0, 0)
        oled.text(f"Gas: {gas_ppm} ppm", 0, 10)
        oled.text(f"Light: {light}", 0, 20)
        oled.text(f"Vent: {'ON' if vent_override else 'OFF'}", 0, 30)
        oled.text(f"AC: {'ON' if ac_override else 'OFF'}", 0, 40)
        oled.show()

        time.sleep(2)

    except Exception as e:
        print("🛑 Внутрішня помилка:", e)
        time.sleep(2)
```

###Slave ESP32

```
from machine import Pin, ADC
import dht
import time
import ujson
import urandom
import urequests  #
# --- Сенсори ---
dht_sensor = dht.DHT22(Pin(13))
mq2 = ADC(Pin(32))
mq2.atten(ADC.ATTN_11DB)
ldr = ADC(Pin(34))
ldr.atten(ADC.ATTN_11DB)
```

```
led = Pin(21, Pin.OUT)
# --- Глобальні змінні ---
_temp = 25.0
_hum = 55
_gas = 420
# --- Адреса Master ESP ---
# MASTER_URL = "http://192.168.1.101:8001/api/from-slave"   змінити на реально адресу
# --- Допоміжна функція для обмеження значень ---
def clamp(val, min_val, max_val):
    return max(min_val, min(max_val, val))
# --- Основний цикл ---
while True:
    try:
        # Генерація імітаційних значень
        _temp += urandom.getrandbits(4) / 10 - 0.8
        _hum += urandom.getrandbits(3) % 5 - 2
        _gas += urandom.getrandbits(5) - 15

        _temp = clamp(_temp, 15, 33)
        _hum = clamp(_hum, 35, 90)
        _gas = clamp(_gas, 300, 800)
        light = ldr.read()
        # Формування JSON-даних
        payload = {
            "temp": round(_temp, 1),
            "hum": int(_hum),
            "gas": int(_gas),
            "light": light
        }
        json_data = ujson.dumps(payload)
        print("📤 Дані на MASTER:", json_data)
        # LED індикація передачі
        led.value(1)
        # Відправлення POST-запиту до Master ESP
        try:
            res = urequests.post(MASTER_URL, json=payload)
            print("📡 Статус відповіді:", res.status_code)
        except Exception as e:
            print("❌ Помилка з'єднання з MASTER:", e)
```

```
    led.value(0)
    time.sleep(15)  # ⏱️ Затримка передачі (можна адаптувати під режим роботи)
except Exception as e:
    print("🛑 Помилка виконання:", e)
    time.sleep(2)
```

ДОДАТОК З

Демонстраційний матеріал

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			<u>Текстові документи</u>		
А4	1	ГЮІК XXXXXX.010 ПЗ	Пояснювальна записка	85 с.	
			<u>Інші документи</u>		
А4	2		Апробація результатів роботи	2 с.	
А4	3		Код мессенджер-бота	13 с.	
А4	4		Код серверної частини FastApi	8 с.	
А4	5		Код симуляторів Master та Slave ESP32	3 с.	
А4	6		Код роботи з базою даних	3 с.	
А4	7		Код альтернативного інтерфейсу користувача	5 с.	
А4	8		Конфігураційні файли порогових значень та робочого графіку	2 с.	
А4	9		Приклад коду розробленого макету у середовищі WokWi із можливістю адаптування під реальні пристрої	5 с.	
А4	10		<u>Демонстраційний матеріал</u>	12 с.	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮІК XXXXXX.010 ВД			
Розроб.		Молоканов К. А.			Розроблення програмного забезпечення для автоматизованого керування освітленням та кліматом у виробничих приміщеннях Відомість кваліфікаційної роботи	Літера	Аркуш	Аркушів
Перевір.		Самойленко Г. Ю.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н. П.						
Затв.		Невлюдов І. Ш.						