Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«ІОТ-СИСТЕМА МОНІТОРИНГУ ПАРАМЕТРІВ МІКРОКЛІМАТУ ПРИМІЩЕННЯ НА БАЗІ МІКРОКОНТРОЛЕРА ESP32»

Спеціальність ***151 – Автоматизація та комп'ютерно-інтегровані технології***
Освітня програма ***Автоматизація та комп'ютерно інтегровані технології***

Виконавець _____ **Богдан ЧУЄВ**
(підпис)

Науковий керівник _____ **Артем БЕРЕЖНИЙ**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 85 сторінок, із яких 43 сторінки основної частини з 9 рисунками, 6 таблицями, 26 найменувань у списку використаних джерел.

Метою кваліфікаційної роботи є вдосконалення методів і моделей побудови комп'ютерно-інтегрованих систем моніторингу за допомогою сенсорів BME280, BH1750, MQ-135 із багатозадачною обробкою, хмарною візуалізацією та Telegram-інтерфейсом для підвищення оперативності й достовірності дистанційного інформування про стан мікроклімату.

Об'єкт дослідження – процеси автоматизованого збору, локального оброблення та передачі даних за допомогою сенсорів BME280, BH1750, MQ-135, мікроконтролера ESP32, сервісу ThingSpeak і платформи Telegram.

Предмет дослідження – моделі трирівневої архітектури системи моніторингу, багатозадачної обробки (FreeRTOS), двосторонньої взаємодії через веб-інтерфейс і Telegram-бот, а також методи цифрової фільтрації.

Проблема, яка вирішується в кваліфікаційній роботі полягає у необхідності своєчасного виявлення відхилень мікроклімату від санітарних норм шляхом вдосконалення методів побудови відмовостійких комп'ютерно-інтегрованих систем.

Область застосування – системи «розумний дім», офісні, житлові та складські приміщення, екологічний моніторинг, освітній процес за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології».

Ключові слова: автоматизація та комп'ютерно-інтегровані технології, інтернет речей, мікроклімат, мікроконтролер ESP32, Arduino IDE, моніторинг, FreeRTOS, ThingSpeak, Telegram-бот.

ABSTRACT

The explanatory note to the bachelor's qualification work consists of an introduction, four sections, conclusions, a list of references, and four appendices. The total volume of the work is 85 pages, of which 43 pages constitute the main body, containing 9 figures, 6 tables, and 26 items in the list of references.

The purpose of the qualification work is to improve the methods and models for building computer-integrated monitoring systems using BME280, BH1750, and MQ-135 sensors with multitasking processing, cloud visualization, and a Telegram interface to increase the promptness and reliability of remote microclimate status reporting.

The object of research is the processes of automated collection, local processing, and data transmission using BME280, BH1750, and MQ-135 sensors, the ESP32 microcontroller, the ThingSpeak service, and the Telegram platform.

The subject of research is the models of a three-level monitoring system architecture, multitasking processing (FreeRTOS), two-way interaction via a web interface and a Telegram bot, as well as methods of digital filtering.

The problem solved in the qualification work is the need for timely detection of microclimate deviations from sanitary standards by improving the methods of building fault-tolerant computer-integrated systems.

The area of application includes "smart home" systems, office, residential, and warehouse premises, environmental monitoring, and the educational process in specialty 151 "Automation and computer-integrated technologies".

Keywords: automation and computer-integrated technologies, Internet of Things, microclimate, ESP32 microcontroller, Arduino IDE, monitoring, FreeRTOS, ThingSpeak, Telegram bot.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ ІОТ-СИСТЕМ МОНІТОРИНГУ МІКРОКЛІМАТУ ПРИМІЩЕНЬ 10
1.1 Поняття мікроклімату та його вплив на здоров'я людини 10
1.2 Огляд існуючих IoT-систем моніторингу мікроклімату 11
1.3 Технології бездротової передачі даних у IoT-системах 14
1.4 Хмарні платформи для IoT-моніторингу 15
Висновки за розділом 1 15
РОЗДІЛ 2. ВИБІР АПАРАТНОЇ ПЛАТФОРМИ ТА ДАТЧИКІВ 17
2.1 Обґрунтування вибору мікроконтролера ESP32 17
2.2 Вибір та характеристики датчиків мікроклімату 19
2.2.1 Датчик температури, вологості та тиску BME280 20
2.2.2 Датчик освітленості BH1750 21
2.2.3 Датчик якості повітря MQ-серії 21
2.3 Вибір допоміжних компонентів системи 22
Висновки за розділом 2 25
РОЗДІЛ 3. ПРОЄКТУВАННЯ СТРУКТУРИ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ 26
3.1 Структурно-функціональна організація комплексу 26
3.2 Архітектура програмного забезпечення на базі FreeRTOS 27
3.3 Алгоритми збору, обробки та аналізу даних 29
3.4 Організація мережевої взаємодії та хмарної інтеграції 31

3.5 Розробка інтерфейсів користувача: локальний веб-сервер та Telegram-бот 32

Висновки за розділом 3 36

РОЗДІЛ 4. ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ 37

4.1 Методика тестування системи 37

4.2 Результати тестування та аналіз 38

4.3 Порівняння з існуючими рішеннями 41

Висновки за розділом 4 42

ВИСНОВКИ 44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ДОДАТКИ 49

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

API	–	Application Programming Interface (програмний інтерфейс застосунку)
BH1750	–	цифровий датчик освітленості виробництва ROHM Semiconductor
BME280	–	цифровий датчик температури, вологості та тиску виробництва Bosch
ESP32	–	двоядерний мікроконтролер виробництва Espressif Systems
FreeRTOS	–	вільна операційна система реального часу для мікроконтролерів
GPIO	–	General-Purpose Input/Output (інтерфейс вводу/виводу загального призначення)
HTTP	–	HyperText Transfer Protocol (протокол передачі гіпертексту)
HTTPS	–	HyperText Transfer Protocol Secure (безпечний протокол передачі гіпертексту)
I2C	–	Inter-Integrated Circuit (двопровідний послідовний інтерфейс)
IoT	–	Internet of Things (Інтернет речей)
JSON	–	JavaScript Object Notation (текстовий формат обміну даними)
LEDC	–	LED Controller (контролер ШІМ-сигналів мікроконтролера ESP32)
LittleFS	–	файлова система для мікроконтролерів з Flash-пам'яттю
OLED	–	Organic Light-Emitting Diode (органічний світлодіодний дисплей)
PWM (ШІМ)	–	широтно-імпульсна модуляція (Pulse Width Modulation)
RGB LED	–	триколірний світлодіод (Red-Green-Blue)
RTOS	–	Real-Time Operating System (операційна система реального часу)

SSD1306	–	контролер OLED-дисплею виробництва Solomon Systech
TVOC	–	Total Volatile Organic Compounds (загальний вміст летких органічних сполук)
Wi-Fi	–	Wireless Fidelity (стандарт бездротової передачі даних IEEE 802.11)
UPS	–	Uninterruptible Power Supply (джерело безперебійного живлення; див. ДБЖ)
АЦП	–	Аналого-цифровий перетворювач
ДБЖ	–	Джерело безперебійного живлення (див. UPS)
ДСН	–	Державні санітарні норми України
ПЗ	–	Програмне забезпечення

ВСТУП

Актуальність роботи. Якість мікроклімату виробничих та житлових приміщень безпосередньо впливає на здоров'я, самопочуття та продуктивність людини. Відповідно до Закону України «Про охорону праці» [2] та Державних санітарних норм ДСН 3.3.6.042-99 [1], дотримання нормативних параметрів температури, відносної вологості та освітленості є обов'язковим для приміщень різного призначення. Традиційні методи контролю передбачають епізодичні вимірювання окремими приладами, що не дозволяє фіксувати динаміку змін у реальному часі та своєчасно реагувати на порушення встановлених норм.

Розвиток технологій Інтернету речей (IoT) відкрив нові можливості для організації безперервного автоматизованого моніторингу параметрів навколишнього середовища. Мікроконтролери з інтегрованими модулями Wi-Fi, зокрема ESP32 виробництва Espressif Systems, у поєднанні з точними цифровими датчиками та хмарними платформами дозволяють побудувати ефективні, масштабовані та доступні за вартістю системи моніторингу [6].

Метою дослідження є вдосконалення методів і моделей побудови комп'ютерно-інтегрованих систем моніторингу за допомогою сенсорів BME280, BH1750, MQ-135 із багатозадачною обробкою, хмарною візуалізацією та Telegram-інтерфейсом для підвищення оперативності й достовірності дистанційного інформування про стан мікроклімату.

Об’єкт дослідження – процеси автоматизованого збору, локального оброблення та передачі даних за допомогою сенсорів BME280, BH1750, MQ-135, мікроконтролера ESP32, сервісу ThingSpeak і платформи Telegram.

Предмет дослідження – моделі трирівневої архітектури системи моніторингу, багатозадачної обробки (FreeRTOS), двосторонньої взаємодії через веб-інтерфейс і Telegram-бот, а також методи цифрової фільтрації.

Завдання дослідження:

1. Аналіз сучасних IoT-систем моніторингу параметрів мікроклімату приміщень.

2. Дослідження та обґрунтування вибору апаратної платформи і датчиків для вимірювання параметрів мікроклімату.

3. Розробка структурної та функціональної моделі IoT-системи моніторингу на базі мікроконтролера ESP32.

4. Розробка програмного забезпечення збору, обробки та передачі даних у бездротовій мережі.

5. Реалізація засобів відображення та дистанційного моніторингу параметрів мікроклімату.

6. Проведення тестування системи та аналіз ефективності її роботи.

Методи дослідження: системний аналіз апаратних платформ та програмних рішень для IoT; метод порівняльного аналізу при виборі компонентів системи; методи тестування вбудованих систем реального часу; аналіз санітарних норм ДСН 3.3.6.042-99.

Практична значущість роботи полягає у вдосконаленні методів побудови та практичній реалізації повнофункціонального прототипу IoT-системи моніторингу мікроклімату, придатного для безпосереднього використання в офісних, навчальних та житлових приміщеннях. Система реалізує автоматичний аналіз відповідності адаптованим пороговим значенням на основі ДСН та надсилає тривожні сповіщення через Telegram-бот.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (26 найменувань) та додатків. Загальний обсяг роботи становить 85 сторінок, з них 43 сторінки основного тексту.

РОЗДІЛ 1.
АНАЛІЗ ІОТ-СИСТЕМ МОНІТОРИНГУ МІКРОКЛІМАТУ ПРИМІЩЕНЬ

1.1 Поняття мікроклімату та його вплив на здоров'я людини

Мікроклімат виробничих і житлових приміщень — це сукупність метеорологічних умов внутрішнього середовища, що характеризується показниками температури повітря та поверхонь, відносної вологості, швидкості руху повітря та інтенсивності теплового опромінення [1]. В Україні нормування параметрів мікроклімату здійснюється відповідно до Державних санітарних норм ДСН 3.3.6.042-99, які регламентують допустимі та оптимальні значення кліматичних показників залежно від категорії важкості робіт та пори року [3].

Для категорії робіт Іа (легкі фізичні роботи, що виконуються сидячи з незначним фізичним навантаженням) у табл. 1.1 наведено адаптовані порогові значення параметрів мікроклімату, прийняті в системі для алгоритмів моніторингу та формування тривожних сповіщень.

Таблиця 1.1

Адаптовані порогові значення параметрів мікроклімату, прийняті в системі на основі ДСН 3.3.6.042-99 (категорія Іа)

Параметр	Пора року	Мін. знач.	Макс. знач.
Температура повітря, °C	Тепла	21,0	28,0
Температура повітря, °C	Холодна	20,0	25,0
Відносна вологість, %	Обидві	15	75
Освітленість, лк	Обидві	300	500

Примітка. Нижні межі температурних діапазонів знижено на 1 °C відносно допустимих значень ДСН 3.3.6.042-99 для зменшення частоти хибних тривог при незначних коливаннях температури. Діапазон відносної вологості (15–75 %)

прийнято як узагальнений, спираючись на максимально допустиме значення для холодного періоду (до 75 %) за ДСН, та поширено на всі режими роботи системи для уникнення хибних спрацювань і спрощення програмної логіки.

Тривале перебування в умовах підвищеної температури (понад 28 °C у теплу пору) призводить до теплового перевантаження організму, зниження концентрації уваги та продуктивності праці на 15–25 % [22]. Підвищений рівень вологості сприяє розвитку грибкових уражень поверхонь та загостренню алергічних реакцій; водночас надмірно сухе повітря спричиняє пересихання слизових оболонок та подразнення дихальних шляхів [12].

Якість повітря оцінюється за вмістом летких органічних сполук (TVOC). Відповідно до рекомендацій Всесвітньої організації охорони здоров'я, концентрація TVOC понад 400 мкг/м³ класифікується як підвищена та потребує провітрювання приміщення. Датчики MQ-серії вимірюють інтегральний показник загальної концентрації летких сполук та формують аналоговий сигнал, що зручно обробляється 12-бітним АЦП мікроконтролера ESP32 [26].

Таким чином, комплексний безперервний моніторинг усіх зазначених параметрів є необхідною умовою підтримання нормативного мікроклімату та своєчасного виявлення відхилень. Ручний контроль із застосуванням окремих приладів є неефективним і не дозволяє відстежувати динаміку змін, що обумовлює актуальність розробки автоматизованої IoT-системи.

1.2 Огляд існуючих IoT-систем моніторингу мікроклімату

Для розуміння принципів побудови сучасних засобів контролю навколишнього середовища доцільно розглянути їх базову архітектуру. Узагальнену структурну схему типової IoT-системи моніторингу параметрів мікроклімату наведено на рис. 1.1.



Рисунок 1.1 – Узагальнена структурна схема IoT-системи моніторингу параметрів мікроклімату.

Наведена конфігурація є базовою для більшості сучасних пристроїв побутового та промислового екологічного контролю.

На сучасному ринку представлені рішення різних класів — від автономних датчиків до комплексних інтегрованих платформ моніторингу. Аналіз існуючих систем дозволяє виявити їх переваги та недоліки, що є підставою для обґрунтування власного рішення.

Комерційні рішення класу «розумний будинок», зокрема датчики Xiaomi Mi Temperature and Humidity Monitor або Sonoff SNZB-02, орієнтовані на моніторинг базових параметрів — температури та вологості. Ключові недоліки: закрита екосистема виробника, відсутність інструментів для аналізу якості повітря (TVOC) та необхідність придбання додаткового апаратного шлюзу (Gateway) для підключення до мережі Інтернет [23]. Рішення елітного сегмента, такі як Netatmo Smart Indoor Climate Module, вимірюють розширений спектр параметрів, але їхня вартість перевищує 100 USD.

Наукові та любительські розробки на базі Arduino UNO з Ethernet-шилдом [20] або Raspberry Pi [25] забезпечують ширший набір параметрів, але мають обмеження: Arduino UNO не має вбудованого Wi-Fi і потребує додаткового модуля; Raspberry Pi є надмірно потужним і дорогим рішенням для задач простого моніторингу з постійно увімкненим Linux.

Системи на базі ESP8266 набули широкого поширення в IoT-застосуваннях завдяки інтегрованому Wi-Fi та низькій вартості. Проте ESP8266 має одне ядро

процесора, що унеможливлює паралельне виконання задач обробки даних та мережевих операцій — мережевий запит блокує основний цикл.

Для систематизації виявлених недоліків та обґрунтування вибору оптимальної апаратної бази проведено комплексне оцінювання розглянутих аналогів. До уваги бралися такі критичні критерії: наявність вбудованого Wi-Fi, кількість обчислювальних ядер та вимірюваних параметрів, підтримка Telegram-сповіщень і норм ДСН, а також відкритість програмного забезпечення та загальна вартість комплексу. Порівняльний аналіз існуючих рішень наведено у табл. 1.2.

Таблиця 1.2

Порівняльний аналіз існуючих рішень IoT-моніторингу

Критерій	**Комерційні (Xiaomi/Sonoff)**	**Arduino+ETH**	**ESP8266**	**ESP32 (запропон.)**
Вбудований Wi-Fi	Ні (потребують Hub)	Ні	Так	Так
Кількість ядер ЦП	—	1	1	2
Кількість параметрів	2	до 5	до 5	5
Telegram-сповіщення	Ні	Ні	Часткове	Так
Норми ДСН	Ні	Ні	Ні	Так
Відкрите ПЗ	Ні	Так	Так	Так
Вартість, USD	~25–35 (Датчик + Hub)	25–50	10–20	~30 (із ДБЖ)

Як видно з табл. 1.2, запропоноване рішення на базі ESP32 поєднує переваги наявних підходів. Хоча вартість компонентів становить близько 30 USD (що є еквівалентним вартості базового комерційного датчика разом із необхідним

мережевим шлюзом), у цю суму вже включено модуль безперебійного живлення (ДБЖ, англ. UPS), локальний OLED-дисплей та систему релейної комутації. Це перетворює пристрій на повноцінну автономну станцію. Унікальною характеристикою є також підтримка адаптованих порогових значень на основі ДСН 3.3.6.042-99 та їх сезонне перемикання.

1.3 Технології бездротової передачі даних у IoT-системах

Вибір технології бездротової передачі є одним із ключових рішень при проектуванні IoT-системи. Для задач моніторингу мікроклімату в приміщеннях розглядаються такі технології: Wi-Fi (IEEE 802.11 b/g/n), Bluetooth Low Energy (BLE 4.2/5.0), Zigbee (IEEE 802.15.4) та LoRaWAN [6].

Wi-Fi (IEEE 802.11 b/g/n) є найбільш поширеним стандартом для IoT-пристроїв у приміщеннях завдяки широкій доступності інфраструктури, підтримці повного стека TCP/IP і протоколів HTTP/HTTPS та можливості прямого підключення до мережі Інтернет. Стандарт забезпечує швидкість передачі до 150 Мбіт/с у діапазоні 2,4 ГГц на відстані до 50–100 м у приміщенні [14]. Відносно висока споживана потужність (до 240 мА при активній передачі) є прийнятною для стаціонарних систем, що живляться від мережі.

Bluetooth Low Energy (BLE) забезпечує мінімальне енергоспоживання та підходить для носимих пристроїв, але обмежений радіус дії (до 10 м) та відсутність прямого доступу до мережі Інтернет без проміжного шлюзу роблять його менш придатним для систем із хмарним моніторингом.

Zigbee орієнтований на мережі типу «mesh» у промислових системах або «розумному будинку». Потребує окремого шлюзу для підключення до Інтернет та спеціалізованих радіомодулів, що збільшує складність і вартість системи.

Технологія LoRaWAN забезпечує великий радіус дії (до кількох кілометрів) і низьке енергоспоживання, проте характеризується вкрай низькою швидкістю передачі даних. Крім того, її розгортання вимагає наявності специфічної інфраструктури (шлюзів LoRa), що унеможливлює пряму роботу з протоколами HTTP/HTTPS та є економічно недоцільним для локального моніторингу окремого приміщення.

Для розробленої системи обрано технологію Wi-Fi як оптимальний варіант зв'язку. Основними перевагами цього рішення є пряме підключення до мережі Інтернет, забезпечення роботи локального веб-сервера та взаємодії з Telegram API, достатня пропускна здатність, а також наявність готової інфраструктури в більшості цільових приміщень. Протокол HTTP використовується для реалізації веб-інтерфейсу та відправки телеметрії на сервіс ThingSpeak, тоді як обмін даними з Telegram Bot API здійснюється через захищений протокол HTTPS [16].

1.4 Хмарні платформи для IoT-моніторингу

Хмарні платформи IoT надають сервіси зберігання часових рядів, візуалізації та аналізу даних без необхідності власної серверної інфраструктури. Серед популярних платформ: ThingSpeak, Blynk, Adafruit IO та AWS IoT Core [13].

ThingSpeak — це спеціалізована IoT-аналітична платформа від компанії MathWorks, що забезпечує накопичення часових рядів у каналах з кількома полями, візуалізацію у вигляді інтерактивних графіків у реальному часі та аналітику через MATLAB Online. Платформа підтримує протокол HTTP/HTTPS для запису даних за допомогою стандартних GET-запитів. Базовий тарифний план дозволяє надсилати дані не частіше ніж раз на 15 секунд [17].

Як платформу хмарної аналітики для розробленої системи обрано сервіс ThingSpeak. Основними критеріями вибору стали: доступність базового функціоналу для освітніх проєктів без обов'язкової реєстрації фінансових даних, можливість інтеграції за допомогою стандартного протоколу HTTP та наявність надійних засобів візуалізації вимірювальних каналів (температура, вологість, рівень TVOC).

Висновки за розділом 1

За результатами проведеного аналізу встановлено, що параметри мікроклімату суттєво впливають на здоров'я та продуктивність людини, а їх нормування в Україні здійснюється відповідно до ДСН 3.3.6.042-99. Огляд існуючих комерційних рішень показав, що вони не здатні забезпечити комплексний моніторинг усіх нормованих показників за умов відкритого

програмного коду та доступної вартості, що обґрунтовує необхідність вдосконалення підходів до побудови власної системи.

Для розв'язання цього завдання оптимальною технологією бездротової передачі даних визначено Wi-Fi, яка гарантує прямий доступ до мережі Інтернет та підтримку стандартних протоколів HTTP/HTTPS. Це дозволяє легко інтегрувати пристрій із хмарною платформою ThingSpeak, що надає необхідні інструменти для безкоштовного зберігання та візуалізації часових рядів. Як апаратну базу для розробки найдоцільніше використовувати мікроконтролер ESP32, який завдяки двоядерній архітектурі під керуванням FreeRTOS, вбудованому Wi-Fi модулю та широкій екосистемі бібліотек Arduino повністю задовольняє технічні вимоги до проєктування сучасних IoT-систем.

РОЗДІЛ 2.
ВИБІР АПАРАТНОЇ ПЛАТФОРМИ ТА ДАТЧИКІВ

2.1 Обґрунтування вибору мікроконтролера ESP32

Мікроконтролер ESP32 (конкретна апаратна ревізія чіпа ESP32-D0WD-V3, Silicon Revision 301) виробництва Espressif Systems є двоядерним 32-бітним процесором архітектури Xtensa LX6 з тактовою частотою 240 МГц. Ключовою особливістю, що визначила його вибір для даної системи, є апаратна двоядерність, яка в рамках середовища розробки Arduino дозволяє чітко розмежувати обчислювальні потоки. Основне ядро (Core 1) виконує головний циклічний алгоритм застосунку loop() (періодичне опитування датчиків, цифрову фільтрацію сигналів, розрахунок тривожних станів, оновлення OLED-дисплея та локальну світлозвукову індикацію), тоді як допоміжне ядро (Core 0) під керуванням операційної системи FreeRTOS повністю виділене під обслуговування мережевого Wi-Fi стека, фонову обробку запитів асинхронного веб-сервера та виконання паралельної задачі Telegram-бота [11]. Технічні характеристики обраної ревізії ESP32 наведено у табл. 2.1, а графічну схему розташування та призначення його апаратних виводів (пінаут) представлено на рис. 2.1 [19].

Таблиця 2.1

Ключові технічні характеристики мікроконтролера ESP32

Характеристика	Значення
Модель та архітектура ЦП	ESP32-D0WD-V3 (Rev 301), Xtensa LX6, 32-біт, 2 ядра
Тактова частота	240 МГц
Flash-пам'ять	4 МБ (зовнішня SPI NOR Flash на частоті 80 МГц)

Продовження таблиці 2.1

Характеристика	Значення
SRAM	520 КБ фізично (369 КБ доступно для застосунків)
Wi-Fi стандарт	IEEE 802.11 b/g/n, 2,4 ГГц
Bluetooth	BT 4.2, BLE
Інтерфейси	I2C, SPI, UART, I2S, ADC (12-біт, 18 кан.), DAC, LEDC PWM
Напруга живлення	2,2 – 3,6 В
Споживана потужність (Wi-Fi активний)	до 240 мА
Вбудована ОС	FreeRTOS
Підтримка файлової системи	LittleFS, SPIFFS

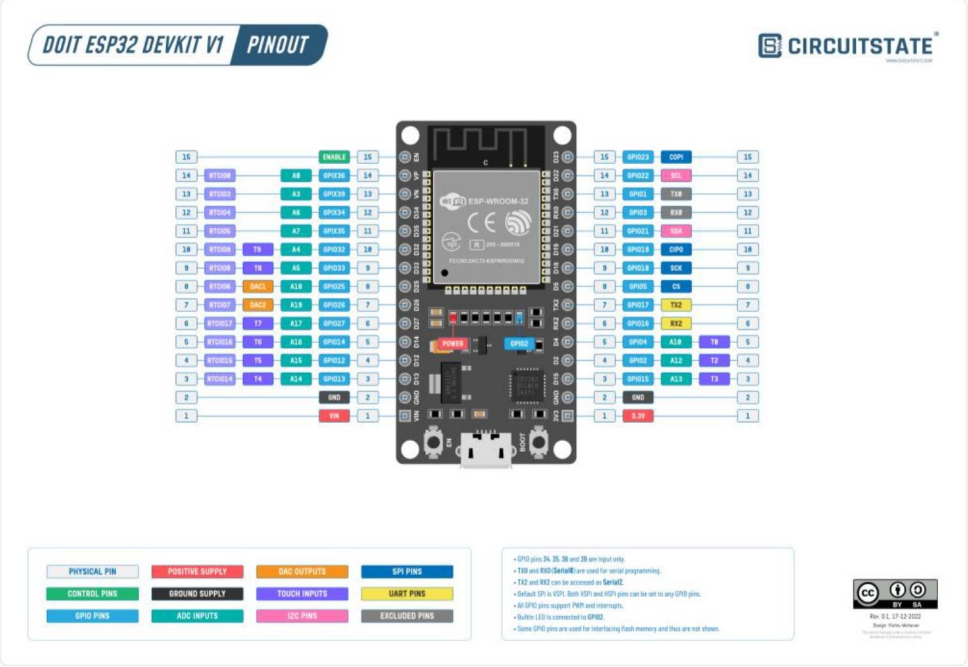


Рисунок 2.1 – Схема розташування та призначення виводів мікроконтролера ESP32 NodeMCU DevKit v1.

Наведена на рис. 2.1 схема підтверджує наявність усіх необхідних апаратних інтерфейсів (I2C, АЦП, ШІМ) для прямого підключення обраних датчиків та дисплея.

Практична реалізація згаданого розподілу обчислень здійснюється за допомогою функції xTaskCreatePinnedToCore(), що жорстко розносить мережеві процеси та опитування датчиків (з інтервалом у 3 секунди) по різних ядрах. Таке архітектурне рішення унеможливлює блокування циклу керування при виконанні HTTP-запитів тривалістю кілька секунд [7].

Для розмежування доступу до спільних даних між ядрами використовується примітив portMUX_TYPE з FreeRTOS API ESP32. Критичні секції, що охоплюють читання/запис глобальних змінних датчиків, захищаються парою portENTER_CRITICAL() / portEXIT_CRITICAL(). Мережеві операції виконуються виключно поза критичними секціями з використанням локально скопійованих даних.

Файлова система LittleFS, розміщена у Flash-пам'яті ESP32, використовується для збереження конфігураційних параметрів системи: токена Telegram-бота, ідентифікатора чату (Chat ID), назви приміщення, а також поточного робочого сезону ДСН та інтервалу генерації автозвітів. Базові налаштування Telegram та назва приміщення задаються користувачем через веб-портал WiFiManager під час первинного підключення до мережі, що дозволяє легко переносити пристрій без перепрошивки [24]. Управління робочими режимами (сезон, інтервали) здійснюється безпосередньо через команди Telegram-бота. Водночас ключ доступу до хмарної платформи ThingSpeak (API Key) інтегрований у програмне забезпечення на етапі компіляції для забезпечення нерозривної прив'язки прототипу до аналітичного каналу.

2.2 Вибір та характеристики датчиків мікроклімату

Система вимірює п'ять параметрів мікроклімату. Для кожного обрано датчик з урахуванням вимог до точності, інтерфейсу та вартості.

2.2.1 Датчик температури, вологості та тиску BME280

Датчик BME280 виробництва Bosch Sensortec є комплексним цифровим сенсором з інтерфейсом I2C (адреса 0x76). Технічні характеристики датчика наведено в табл. 2.2.

Таблиця 2.2

Технічні характеристики датчика BME280

Параметр	Значення
Діапазон вимірювання температури	–40 … +85 °C
Точність вимірювання температури	±0,5 °C (діапазон 0–65 °C)
Діапазон вимірювання вологості	0 … 100 %
Точність вимірювання вологості	±3 %
Діапазон вимірювання тиску	300 … 1100 гПа
Точність вимірювання тиску	±1 гПа (відносна ±0,12 гПа)
Інтерфейс	I2C (до 3,4 МГц), SPI (до 10 МГц)
Напруга живлення	1,71 … 3,6 В
Споживана потужність (норм. режим)	3,6 мкА

Сенсорний модуль має стандартну контактну колодку для підключення по апаратній шині I2C. Отримані через цей інтерфейс значення температури (у градусах Цельсія) та відносної вологості (у відсотках) зчитуються системою безпосередньо і не потребують додаткових перетворень. Водночас значення атмосферного тиску перераховуються з Паскалів у мм рт. ст. за формулою:

$$P(\text{мм рт. ст.}) = P(\text{Па}) \times 0{,}00750062 \qquad (2.1)$$

що відповідає прийнятим в Україні метеорологічним одиницям вимірювання. Також у програмному коді реалізовано захист від збоїв на шині:

якщо датчик повертає значення NaN (Not a Number) або не відповідає, поточний цикл опитування пропускається без оновлення статистики та генерації хибних тривог [4, 9].

2.2.2 Датчик освітленості BH1750

Датчик BH1750 виробництва ROHM Semiconductor є цифровим фотоелектричним сенсором з 16-бітним АЦП. Діапазон вимірювань становить 1–65535 лк, точність — ±20 %, роздільна здатність — 1 лк [15, 21]. Сенсор підтримує автоматичний вибір режиму високої роздільної здатності (H-resolution mode).

Даний модуль має стандартну п'ятиконтактну групу виводів, що дозволило легко об'єднати його із BME280 на спільній апаратній шині I2C. Для забезпечення стабільності роботи системи моніторингу у програмному коді реалізовано захист від збоїв на лінії зв'язку: при поверненні сенсором значення −1 (помилка читання) зберігається попереднє коректне значення освітленості для запобігання хибним тривогам.

2.2.3 Датчик якості повітря MQ-серії

Для вимірювання рівня TVOC використовується аналоговий датчик якості повітря MQ-серії (MQ-135), підключений до аналогового входу GPIO34. Датчик чутливий до широкого спектра летких органічних сполук — спирту, бензолу, аміаку та продуктів горіння. Сигнал обробляється 12-бітним АЦП ESP32 (4096 рівнів квантування, діапазон 0–3,3 В), результат відображається у відсотках від максимального рівня забруднення [26].

Живлення модуля MQ-135 комутується через модуль реле (GPIO18), що вмикається одразу при старті системи, забезпечуючи безперервний прогрів датчика.

Для зменшення впливу апаратного шуму та раптових стрибків концентрації застосовується програмний алгоритм цифрової фільтрації з асиметричним часом реакції (детальний математичний опис алгоритму наведено у розділі 3). Конструктивне виконання вимірювального модуля (наявність

готової плати з необхідною електронною обв'язкою) дозволяє зручно інтегрувати його в загальну апаратну частину розроблюваної системи.

2.3 Вибір допоміжних компонентів системи

OLED-дисплей SSD1306 (128×64 пікс., діагональ 0,96 дюйма) забезпечує місцеве відображення поточних показників. Пристрій підключається через апаратну шину I2C (SDA — GPIO21, SCL — GPIO22, адреса 0x3C) спільно із датчиками BME280 та BH1750. Програмне керування екраном реалізовано за допомогою бібліотеки Adafruit_SSD1306 [5]. Реалізовано чотири режими відображення: IP-адреса, температура та вологість; атмосферний тиск та освітленість; рівень TVOC; стани RGB. При невдалій ініціалізації дисплея на старті система автоматично виконує функцію ESP.restart() після 5 секунд очікування. Використання даного модуля забезпечує повноцінну автономну локальну візуалізацію всіх параметрів мікроклімату без обов'язкового підключення до хмарного інтерфейсу.

RGB-світлодіод із загальним катодом підключений до виводів мікроконтролера GPIO26 (R), GPIO27 (G) та GPIO14 (B), причому кожна сигнальна лінія підключена через струмообмежувальний резистор номіналом 220 Ом. Управління здійснюється через апаратний інтерфейс LEDC PWM API ESP32 Core 3.x [10]: функція ledcAttach() прикріплює пін до ШІМ-каналу (частота 5 кГц, 8-бітна розрядність — 256 рівнів), а ledcWrite() встановлює яскравість. Колірне кодування RGB-індикатора наведено в табл. 2.3.

Таблиця 2.3

Колірне кодування RGB-індикатора

Колір / значення ШІМ (R, G, B)	Стан системи
Зелений (0, 80, 0)	Ідеальна норма: TVOC < 20 %, T і H мають буферний відступ від адаптованих порогових меж (±1.0 °C та ±5 % відповідно)

Продовження таблиці 2.3

Колір / значення ШІМ (R, G, B)	**Стан системи**
Синій (0, 0, 80)	Штатний режим: параметри в межах норми, але поблизу порогів
Жовтогарячий миготливий → постійний (80, 40, 0)	Тривога температури (вихід за порогові межі)
Пурпурний миготливий → постійний (80, 0, 80)	Тривога вологості (вихід за порогові межі)
Червоний миготливий → постійний (80, 0, 0)	Критична тривога якості повітря (TVOC > 35 %)

Освітленість не має окремого кольору RGB-індикатора — при відхиленні (< 100 лк або > 800 лк) система надсилає виключно Telegram-сповіщення. Звуковим сигналом зуммера супроводжується виключно критична тривога якості повітря (перевищення рівня TVOC > 35 %). У цьому випадку зуммер генерує переривчастий звуковий сигнал протягом перших 7 секунд, після чого вимикається. Тривоги температури та вологості мають лише візуальну RGB-індикацію (миготіння відповідним кольором: для температури — перші 10 секунд, для вологості — 5 секунд, після чого відбувається перехід у постійне світіння до нормалізації) та Telegram-сповіщення, що запобігає зайвому шумовому забрудненню в приміщенні.

Зуммер низького рівня (GPIO25) формує звукове сповіщення при тривогах. HIGH = вимкнено; LOW = увімкнено, що відповідає специфікації low-level buzzer.

Тактова кнопка (GPIO13) підтягнута через резистор 10 кОм до GND; пін налаштований в INPUT_PULLDOWN. Коротке натискання — перемикання режиму відображення на OLED-дисплеї. Тривале утримання кнопки реалізовано у двох програмних варіантах: при старті пристрою (утримання 3 с через блокуючий цикл ініціалізації) та під час звичайної роботи (утримання понад 4 с із неблокуючим відліком через функцію millis(), що супроводжується візуальним зворотним відліком на екрані). В обох випадках це призводить до повного скидання пристрою до заводських налаштувань: форматування файлової системи LittleFS, видалення збережених параметрів Wi-Fi та перезавантаження мікроконтролера.

Тумблер MTS-102 встановлено на лінії живлення 5 В між UPS-модулем та рештою схеми — головний вимикач пристрою.

Модуль реле 5 В / 10 А з оптороз'язкою (GPIO18) комутує живлення датчика MQ-135. Реле є low-level: LOW на керуючому виводі вмикає реле. При ініціалізації GPIO18 одразу встановлюється в LOW, забезпечуючи безперервний прогрів чутливого елемента MQ-135 з моменту увімкнення пристрою (виробник рекомендує 24–48 год початкового прогріву). Оптороз'язка захищає ESP32 від зворотних ЕРС. Перспективою є реалізація циклічного режиму: вмикання датчика лише за 30 с до вимірювання для зниження енергоспоживання.

UPS-модуль на базі двох акумуляторів 18650 забезпечує резервне живлення 5 В. Це гарантує безперервну роботу системи при короткочасних перебоях електроживлення та дозволяє автономне розміщення пристрою.

WiFiManager реалізує налаштування мережі та базової конфігурації системи без перепрошивки мікроконтролера. При відсутності збережених мережевих даних ESP32 розгортає точку доступу «Climate_Monitor» (пароль: Admin12345) із вбудованим веб-порталом (Captive Portal). Окрім стандартного вибору Wi-Fi мережі, портал містить додаткові поля для введення користувацьких параметрів: токена Telegram-бота, ідентифікатора чату (Chat ID) та назви приміщення. Після успішного підключення всі ці параметри

серіалізуються у формат JSON та надійно зберігаються у Flash-пам'яті (файлова система LittleFS) [24].

Висновки за розділом 2

У другому розділі здійснено комплексне обґрунтування та вибір апаратної бази розроблюваної системи моніторингу. Головним обчислювальним модулем обрано мікроконтролер ESP32, що обумовлено його двоядерною архітектурою з підтримкою операційної системи FreeRTOS, вбудованим модулем Wi-Fi та достатньою кількістю апаратних інтерфейсів (I2C, ADC, LEDC). Сенсорний рівень реалізовано на основі цифрового датчика BME280, який забезпечує точне вимірювання температури, відносної вологості та атмосферного тиску при мінімальному енергоспоживанні, а також цифрового датчика BH1750 для надійного моніторингу рівня освітленості із прямим підключенням до спільної шини I2C.

Для інтегральної оцінки якості повітря застосовано аналоговий сенсор MQ-135, який дозволяє визначати концентрацію летких сполук (TVOC) із подальшою програмною обробкою сигналу алгоритмом асиметричного ковзного середнього. Повноцінний локальний інтерфейс користувача для візуалізації телеметрії та багаторівневого сповіщення про відхилення сформовано на базі графічного OLED-дисплея, багатоколірного RGB-індикатора та звукового зуммера. Надійність функціонування прототипу забезпечується підсистемою живлення, яка поєднує UPS-модуль на базі акумуляторів 18650 для автономності та модуль реле 5 В з опторозв'язкою для стабільного прогріву сенсора якості повітря.

РОЗДІЛ 3.
ПРОЄКТУВАННЯ СТРУКТУРИ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Структурно-функціональна організація комплексу

Запропонована IoT-система моніторингу навколишнього середовища побудована за трирівневою архітектурою, яка включає рівень збору даних (Edge-рівень), рівень локальної обробки та відображення (рівень мікроконтролера) і рівень дистанційного моніторингу (хмарний рівень).

Рівень збору даних реалізовано за допомогою високоточних цифрових датчиків BME280 (температура, вологість, атмосферний тиск) та BH1750 (рівень освітленості), які інтегровані в систему через апаратну шину I2C (виводи SDA — GPIO21, SCL — GPIO22). Сенсор якості повітря MQ-135, який визначає концентрацію летких органічних сполук (TVOC), підключений до аналогового входу GPIO34. Для забезпечення стабільного прогріву чутливого елемента живлення цього сенсора безперервно комутується модулем реле (GPIO18). Опитування сенсорів виконується циклічно кожні 3000 мс на першому ядрі мікроконтролера. Після зчитування здійснюється автоматична перевірка валідності даних: обробка помилок читання NaN для I2C-пристроїв та відфільтровування помилкових від'ємних значень для фотодатчика.

Рівень обробки на базі мікроконтролера ESP32 здійснює цифрову фільтрацію аналогового сигналу, порівняння виміряних значень із адаптованими пороговими межами на основі ДСН 3.3.6.042-99, управління локальною RGB-індикацією та звуковим зуммером (GPIO25), а також формування кадру для оновлення OLED-дисплея SSD1306.

Рівень дистанційного моніторингу включає вбудований асинхронний веб-сервер, який надає HTML-дашборд та JSON API для локального зчитування даних. Інтегрований Telegram-бот забезпечує віддалений доступ через команди та автоматичні сповіщення, а хмарна платформа ThingSpeak накопичує часові ряди для тривалого аналізу динаміки мікроклімату приміщення.

Принципову структурну схему взаємодії описаних рівнів та фізичного підключення периферійних компонентів до 30-пінової плати мікроконтролера наведено на рис. 3.1.

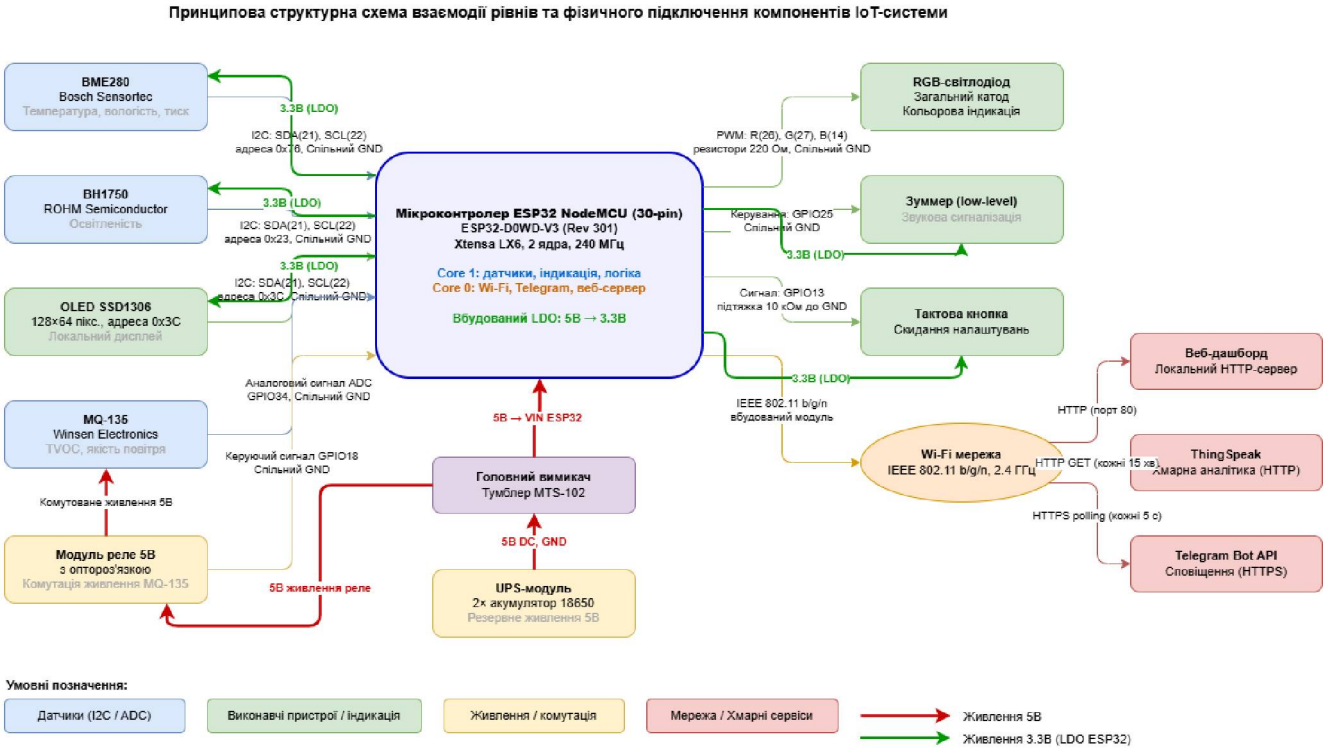


Рисунок 3.1 – Принципова структурна схема підключення компонентів IoT-системи.

Запропонована структура забезпечує модульність комплексу, дозволяючи незалежно модернізувати як апаратні компоненти сенсорного рівня, так і програмні сервіси хмарного рівня.

3.2 Архітектура програмного забезпечення на базі FreeRTOS

Програмне забезпечення розроблено в середовищі Arduino IDE з використанням ядра ESP32 Arduino Core 3.x (мова стандарту C++11/14). Код організовано у вигляді єдиного монолітного файлу з чітким розподілом на функціональні блоки. Для реалізації функціоналу використано бібліотеки: Adafruit_BME280 [4], BH1750 [15], Adafruit_SSD1306 [5], ESPAsyncWebServer [18], UniversalTelegramBot [16], ArduinoJson [8], WiFiManager [24] та вбудовану файлову систему LittleFS.

Загальна структура програми побудована на базі двоядерної архітектури операційної системи реального часу FreeRTOS і включає такі блоки:

1. Ініціалізація setup(): одноразова конфігурація периферії та дисплея, обробка апаратного скидання налаштувань через кнопку (GPIO13) із 3-секундним зворотним відліком, монтування LittleFS, десеріалізація конфігураційного JSON-файлу (токен, Chat ID, сезон ДСН), підключення до Wi-Fi та запуск окремої задачі для Telegram на ядрі Core 0.

2. Основний цикл loop() (Core 1): періодичне опитування датчиків, оновлення загального знімка даних SensorSnapshot під захистом м'ютекса dataMux (тип portMUX_TYPE), логіка генерації тривожних станів та управління світлозвуковою периферією (ledc-керування RGB на виводах GPIO26, GPIO27, GPIO14).

3. Фонова задача telegramTaskCode() (Core 0): моніторинг вільної динамічної пам'яті (Free Heap) для запобігання виснаженню ресурсів, атомарне копіювання прапорців AlertFlags, поллінг Telegram Bot API та обробка вхідних команд користувача. Для забезпечення стабільного HTTPS-з'єднання з серверами Telegram цій задачі виділено збільшений обсяг стека (20480 байт), що обґрунтовано високими вимогами до пам'яті з боку криптографічної бібліотеки TLS (WiFiClientSecure) та необхідністю динамічного формування текстових рядків.

4. Обробники асинхронного веб-сервера: генерація HTML-сторінки та формування JSON-відповідей (/data) за допомогою статичних буферів (snprintf) для оптимізації та запобігання фрагментації оперативної пам'яті.

Взаємодія між ядрами мікроконтролера базується на принципі «єдиного власника»: глобальні змінні датчиків та статистика оновлюються виключно на Core 1. Ядро Core 0 зчитує ці дані тільки через захищену структуру SensorSnapshot. Передача станів тривог між ядрами здійснюється через структуру AlertFlags, яка встановлюється в критичній секції на Core 1 через системні виклики portENTER_CRITICAL(&dataMux) та portEXIT_CRITICAL(&dataMux), атомарно копіюється та скидається на Core 0.

3.3 Алгоритми збору, обробки та аналізу даних

Модуль збору даних виконується кожні 3000 мс поза критичною секцією, оскільки апаратні інтерфейси I2C та ADC у розробленій архітектурі доступні тільки для Core 1. Після успішного зчитування та фільтрації апаратних збоїв усі змінні синхронно оновлюються в єдиній portMUX-секції.

Для вимірювання рівня TVOC використовується аналоговий сигнал сенсора MQ-135, що оцифровується 12-бітним АЦП мікроконтролера (4096 рівнів квантування) та масштабується у відсотковий діапазон від 0 до 100%. З метою зменшення впливу електромагнітного шуму та раптових стрибків концентрації летких речовин застосовується алгоритм цифрової фільтрації з асиметричним часом реакції. При нарощуванні концентрації нове значення приймається системою одразу (забезпечення швидкої реакції на загрозу), а при зниженні — плавно усереднюється з поточним за формулою (3.1):

$$P_f = \frac{(P_{cur} \times 3 + P_{new})}{4} \tag{3.1}$$

де P_f – відфільтроване значення; P_{cur} – поточне значення; P_{new} – нове зчитане значення.

Алгоритм генерації тривог реалізує програмний гістерезис для запобігання хибним спрацюванням при коливаннях значень поблизу порогових меж. Тривога активується при виході параметра за адаптовану порогову межу, а скасовується лише при поверненні в нормативну зону із заданим буферним відступом (0,5 °C для температури та 2 % для вологості). Для параметра освітленості також застосовано гістерезис: аварійний стан активується при виході за межі < 100 лк або > 800 лк, а нормалізація стану і скидання прапорців відбувається лише при поверненні значень у комфортний діапазон 150–700 лк.

Адаптовані порогові значення параметрів підтримуються системою динамічно залежно від обраного користувачем сезону (теплий період: 21–28 °C; холодний період: 20–25 °C). Зміна сезону автоматично скидає активні прапорці

тривог для унеможливлення надсилання хибних сповіщень. Окремий статистичний модуль безперервно накопичує екстремуми (мінімуми та максимуми) кожного параметра за звітний період. При генерації аналітичного звіту generateReport() зняття статистики та обнулення лічильників виконується атомарно під захистом критичної секції. Блок-схему алгоритму роботи основного обчислювального циклу програми, що виконується на ядрі Core 1, подано на рис. 3.2.

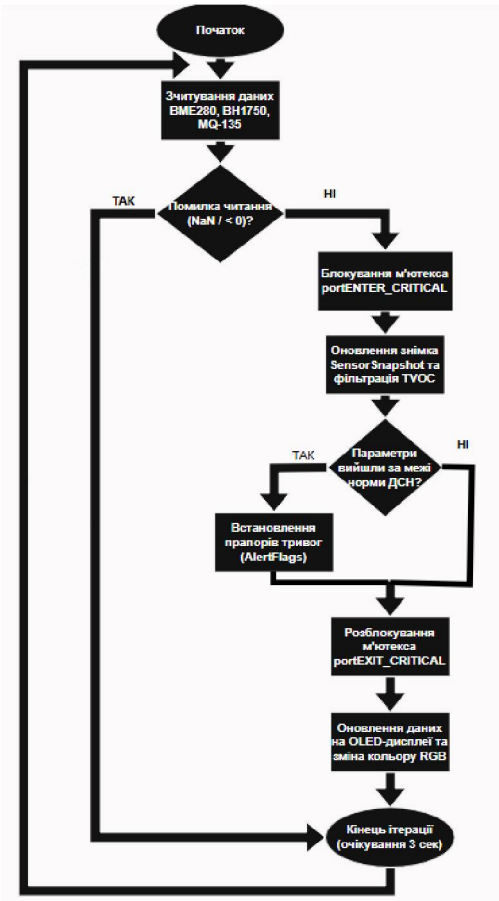


Рисунок 3.2 – Блок-схема алгоритму роботи основного обчислювального циклу мікроконтролера (Core 1).

Наведений на рис. 3.2 алгоритм забезпечує стабільність вимірювального контуру та гарантує миттєву локальну реакцію системи на аварійні стани незалежно від наявності мережевого з'єднання.

3.4 Організація мережевої взаємодії та хмарної інтеграції

Підключення пристрою до локальної мережі Wi-Fi здійснюється за допомогою captive-порталу на базі бібліотеки WiFiManager. При відсутності збережених мережевих реквізитів мікроконтролер переходить у режим Access Point та розгортає бездротову мережу «Climate_Monitor» (пароль: Admin12345). Через веб-портал користувач вводить конфігураційні параметри: токен Telegram-бота, ідентифікатор чату та назву приміщення. Введені дані серіалізуються у JSON-формат через об'єкт StaticJsonDocument та надійно зберігаються у файловій системі LittleFS. У системі реалізовано стійкий механізм автоматичного відновлення зв'язку: при виявленні втрати з'єднання активується автономний режим роботи (isOfflineMode = true), а спроби перепідключення виконуються у фоновому режимі кожні 30 секунд. Для забезпечення максимальної стабільності бездротового з'єднання та уникнення затримок чи втрат пакетів, режим енергозбереження Wi-Fi модуля примусово деактивовано викликом функції WiFi.setSleep(false).

Інтеграція з аналітичною платформою ThingSpeak здійснюється через HTTP GET-запити. Для забезпечення безперебійної роботи ядра алгоритм відправки реалізовано як неблокуючий: локальні копії даних зчитуються під м'ютексом, а сам мережевий запит виконується поза критичною секцією із жорсткими таймаутами з'єднання (setConnectTimeout(5000)) та очікування відповіді (setTimeout(8000)). Блок-схему розробленого алгоритму обробки мережевих подій та передачі даних подано на рис. 3.3.

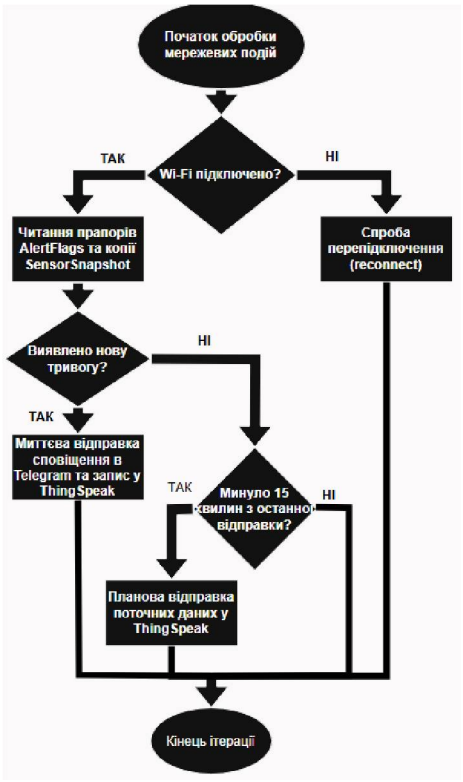


Рисунок 3.3 – Блок-схема алгоритму обробки мережевих подій та логування даних.

Відповідно до наведеного алгоритму, логіка запису телеметрії до ThingSpeak є дворівневою. Плановий запис (Level 1) виконується кожні 15 хвилин (константа LOG_INTERVAL = 900000) незалежно від стану приміщення. Додатковий тривожний запис (Level 2) виконується миттєво при появі нової тривоги, або як нагадування раз на 15 хвилин, якщо критичний стан триває. Це оптимізує мережевий трафік, дозволяє не перевищувати ліміти безкоштовного API ThingSpeak та гарантує негайну фіксацію аварійних відхилень на хмарних графіках. Після нормалізації стану бітовий прапор alarmLoggedFlags обнуляється за допомогою побітової операції &= ~, що забезпечує готовність системи до миттєвої фіксації наступного інциденту.

3.5 Розробка інтерфейсів користувача: локальний веб-сервер та Telegram-бот

Локальний веб-інтерфейс реалізовано у вигляді єдиної HTML-сторінки з вбудованими CSS та JavaScript, яка зберігається як константний рядок

(PROGMEM) у Flash-пам'яті мікроконтролера. Сторінка містить інформаційні картки з поточними показниками та вбудовані iframe-віджети ThingSpeak для візуалізації графіків динаміки. Оновлення даних на сторінці відбувається асинхронно кожні 3 секунди через fetch-запити до ендпоінту /data. Зовнішній вигляд розробленого локального веб-інтерфейсу з картами моніторингу наведено на рис. 3.4.

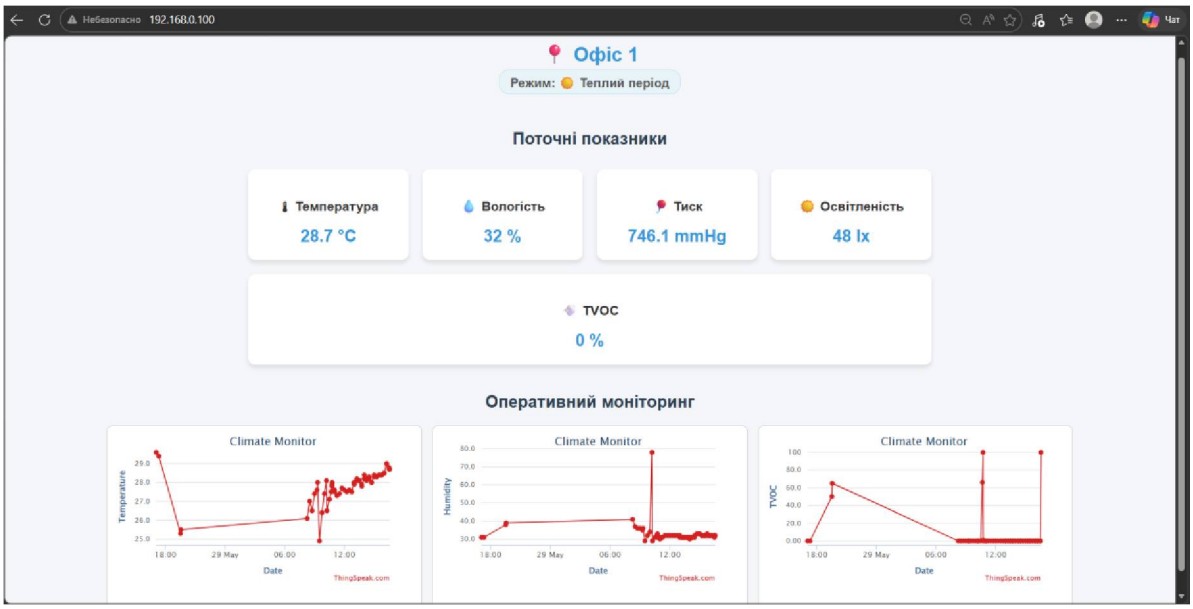


Рисунок 3.4 – Локальний веб-інтерфейс системи моніторингу мікроклімату.

Запропонована структура дашборду забезпечує інтуїтивно зрозуміле сприйняття поточного стану навколишнього середовища без необхідності встановлення додаткового програмного забезпечення.

Вибір месенджера Telegram як основного каналу віддаленої взаємодії обґрунтовується його кросплатформеністю, високою швидкістю доставки push-сповіщень та відсутністю потреби в розробленні й підтримці окремого мобільного застосунку. Для гарантування конфіденційності даних і безпечної авторизації запитів до Telegram API система дистанційного керування реалізована з використанням зашифрованого HTTPS-з'єднання (на базі бібліотеки WiFiClientSecure). Інтерфейс бота підтримує інтуїтивно зрозуміле інтерактивне клавіатурне меню (ReplyKeyboardMarkup) та обробляє 8 базових

команд, що забезпечує повноцінний двосторонній зв'язок із пристроєм. Перелік та призначення команд управління системою наведено в табл. 3.1.

Таблиця 3.1

Команди Telegram-боту системи моніторингу

Команда	Опис
/start	Початок роботи, відображення головного меню з клавіатурою
/status	Отримання поточних значень всіх параметрів мікроклімату приміщення
/alerts	Виведення списку активних тривог та попереджень про відхилення
/season	Перемикання між теплим та холодним профілем норм ДСН
/report	Генерація аналітичного кумулятивного звіту за накопиченими даними
/auto [год]	Встановлення інтервалу автоматичних звітів у годинах (0 — вимкнено)
/ip	Отримання локальної IP-адреси пристрою для доступу до веб-інтерфейсу
/led	Інформаційна довідка щодо режимів колірної індикації RGB-світлодіода

Опитування серверів Telegram (polling) виконується фоновою задачею кожні 5 секунд. Для захисту від зависання мікроконтролера при масовому надходженні повідомлень реалізовано програмний запобіжник (safetyCounter), який обмежує обробку максимум п'ятьма ітераціями за один цикл. Крім того, для захисту від повторної обробки повідомлень після перезавантаження, ідентифікатор останнього обробленого повідомлення зберігається в енергонезалежній RTC-пам'яті мікроконтролера за допомогою атрибута RTC_DATA_ATTR.

Програмна логіка Telegram-бота реалізує диференційований підхід до сповіщень. Повідомлення про відхилення температури, вологості та освітленості відправляється користувачу строго одноразово в момент фіксації події, що запобігає спаму в месенджері. Однак, для найбільш критичного параметра системи — рівня TVOC — реалізовано алгоритм циклічного нагадування: сповіщення дублюється кожні 5 хвилин (задано константою TELEGRAM_COOLDOWN = 300000) до моменту усунення небезпеки. При поверненні температури, вологості та рівня TVOC у адаптовані порогові межі із врахуванням гістерезису, система надсилає відповідне повідомлення про нормалізацію стану (повернення до санітарних норм). Для освітленості відправка повідомлення про нормалізацію навмисно не передбачена: система здійснює лише програмне скидання прапорців тривоги при поверненні в комфортний діапазон (150–700 лк), щоб не перевантажувати інтерфейс користувача. Приклад інтерфейсу головного меню та отримання тривожних сповіщень у месенджері продемонстровано на рис. 3.5.

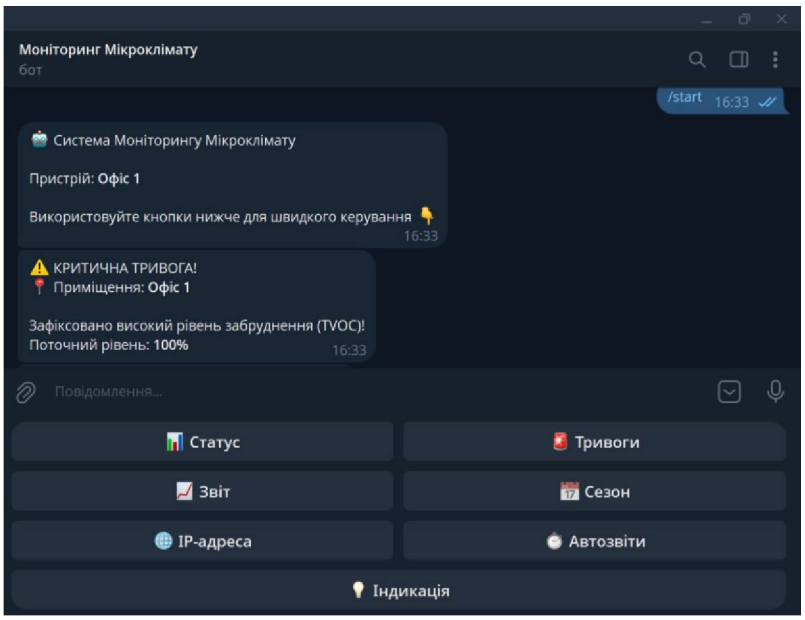


Рисунок 3.5 – Інтерфейс користувача Telegram-бота системи моніторингу.

Як видно з рис. 3.5, реалізований підхід забезпечує ефективний баланс між гарантованим інформуванням про критичні стани та відсутністю інформаційного перевантаження (спаму) в месенджері користувача.

Висновки за розділом 3

У третьому розділі проведено комплекс робіт із проєктування архітектури та безпосередньої програмної реалізації апаратно-програмного забезпечення розроблюваної IoT-системи. На основі отриманих рішень спроєктовано трирівневу структурно-функціональну організацію комплексу, що охоплює Edge-рівень збору даних, рівень локальної апаратної обробки та рівень дистанційного моніторингу. Архітектуру вбудованого програмного забезпечення побудовано на базі операційної системи реального часу FreeRTOS із жорстким розмежуванням обчислювальних потоків між двома ядрами процесора, завдяки чому завдання опитування сенсорів та індикації виконуються на Core 1, а мережева підсистема повністю ізольована на Core 0. Безпечний міжядерний обмін даними при цьому гарантується застосуванням примітивів захисту спільної пам'яті через механізм критичних секцій portMUX.

Разом із цим у програмному забезпеченні реалізовано стійкі алгоритми обробки даних, які включають програмний гістерезис тривог, динамічну підтримку адаптованих порогових значень на основі ДСН 3.3.6.042-99 та асиметричну цифрову фільтрацію показників аналогового сенсора TVOC для усунення шумів. Для взаємодії з хмарними сервісами організовано надійне бездротове підключення з неблокуючим фоновим відновленням Wi-Fi зв'язку та дворівневою логікою відправки телеметрії на платформу ThingSpeak без переривання вимірювального циклу. Фінальним етапом роботи стало створення комплексних інтерфейсів користувача, що складаються з локального веб-дашборду реального часу з періодом асинхронного оновлення 3 секунди та багатофункціонального Telegram-бота із системою інтелектуальних сповіщень і генерацією кумулятивних аналітичних звітів.

РОЗДІЛ 4.
ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методика тестування системи

Експериментальне дослідження запропонованого апаратно-програмного комплексу IoT-системи моніторингу мікроклімату проводилося з метою перевірки та підтвердження його працездатності, оцінки стабільності довготривалого функціонування вбудованого програмного забезпечення на базі операційної системи реального часу FreeRTOS, а також перевірки точності обробки аварійних ситуацій відповідно до адаптованих порогових значень на основі ДСН 3.3.6.042-99.

Тестування комплексу здійснювалося в умовах реального житлового приміщення, мікрокліматична обстановка якого моделює типовий робочий простір (у системі пристрій зареєстровано під тестовою назвою «Офіс 1»), протягом безперервного 8-годинного інтервалу часу. Методика проведення випробувань передбачала моделювання як штатного режиму функціонування об'єкта, так і штучне створення критичних відхилень для перевірки контурів світлозвукового реагування та відправки мережевих сповіщень.

Для реалізації програми випробувань було розроблено таку послідовність дій:

1. Оцінка первинної ініціалізації: аналіз запуску captive-порталу, зчитування конфігураційних файлів JSON із LittleFS та стартового підключення до бездротової мережі Wi-Fi.

2. Контроль та співставлення локальних та мережевих інтерфейсів: порівняльний аналіз даних на локальному OLED-дисплеї та віддаленому асинхронному веб-дашборді.

3. Емуляція критичних станів: короткочасний тепловий вплив на сенсор BME280 та наближення джерела летких органічних сполук до чутливого елемента сенсора MQ-135 для лінійної генерації 100 % рівня забруднення (TVOC).

4. Перевірка алгоритмів аналітики: аналіз формування кумулятивного експертного звіту месенджер-ботом після тривалого накопичення екстремумів.

5. Тестування системи фонового відновлення зв'язку: примусове відключення точки доступу для оцінки поведінки некогерентних FreeRTOS-задач та перевірки переходу системи в автономний режим роботи.

4.2 Результати тестування та аналіз

При первинному ввімкненні апаратної частини пристрою було успішно змонтовано файлову систему LittleFS, зчитано токен бота і Chat ID, після чого мікроконтролер перейшов у режим клієнта мережі Wi-Fi. На вбудованому OLED-екрані відобразилася призначена локальна адреса.

Введення вказаної адреси у веб-браузері персонального комп'ютера підтвердило коректність запуску асинхронного веб-сервера на ядрі Core 0. Веб-сервер миттєво згенерував HTML-сторінку з картами моніторингу параметрів. Дані оновлювалися кожні 3 секунди без перезавантаження сторінки завдяки технології асинхронних запитів, повністю збігаючись із локальним дисплеєм, що свідчить про стабільну роботу критичних секцій обміну пам'яті через об'єкт SensorSnapshot. Зовнішній вигляд розробленого локального веб-дашборду наведено на рис. В.1 (див. Додаток В). Зовнішній вигляд зібраного апаратного прототипу під час проведення стендових випробувань наведено на рис. 4.1.



Рисунок 4.1 – Апаратний прототип IoT-системи під час стендових випробувань.

Наступним етапом випробувань стала перевірка підсистеми генерації тривог та локальної індикації. Як видно з рис. 4.1, прототип виконано у промисловому пластиковому корпусі. На верхній панелі розміщено 0.96-дюймовий OLED-дисплей, багатоколірний RGB-світлодіод візуальної сигналізації, тактову кнопку та отвір для фотоелектричного сенсора. Бічні вентиляційні решітки забезпечують необхідну циркуляцію повітря до внутрішніх датчиків.

При штучному нагріванні повітря понад 28 °C та піднесенні до датчика якості повітря MQ-135 джерела летких речовин, рівень TVOC миттєво зріс до пікових 100 %. Основний обчислювальний цикл на ядрі Core 1 зафіксував вихід за адаптовані порогові межі, активував на 7 секунд звуковий зуммер, а колір RGB-діода змінився на миготливий червоний (аварійний статус).

Паралельно задача на ядрі Core 0 зчитала прапорці тривог та надіслала у месенджер критичне повідомлення. Завдяки реалізованому в коді програмному гістерезису, після усунення джерела забруднення та плавного падіння концентрації до 26 %, система стабілізувала стан і надіслала повідомлення про очищення повітря. Результати цього випробування, що включають інтерфейс бота та відповідні стани пристрою, детально зафіксовані у протоколі випробувань (рис. В.2, Додаток В).

Для перевірки надійності накопичення довготривалої статистики пристрій залишили в робочому стані на 8 годин. Протягом усього часу телеметрія планово логувалася на хмару ThingSpeak кожні 15 хвилин, а при виникненні тривог здійснювався миттєвий дворівневий запис аварійних точок. Динаміку зміни параметрів мікроклімату, яка безперервно фіксувалася на хмарній платформі ThingSpeak протягом усього періоду тестування, відображено на рис. 4.2.

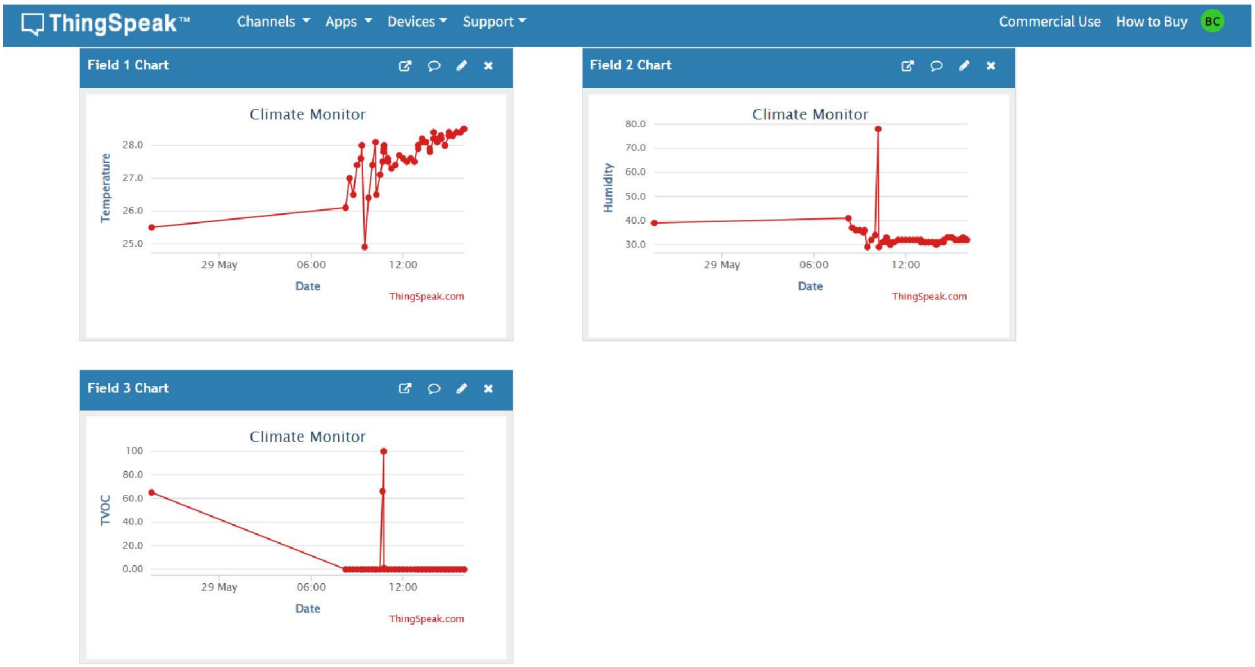


Рисунок 4.2 – Динаміка зміни параметрів мікроклімату на хмарній платформі ThingSpeak за 8 годин.

Наведені графіки візуально підтверджують відсутність розривів у передачі телеметрії, що свідчить про високу стабільність налаштованого мережевого каналу та надійність алгоритмів неблокуючої відправки.

Після завершення 8-годинної сесії через Telegram-бот було викликано команду /report. Сформований системою текстовий аналітичний звіт наведено у Додатку В (рис. В.3). Аналіз сформованого звіту підтверджує повну працездатність статистичного модуля: система зафіксувала аптайм (8 год 0 хв) та точні екстремуми (температурний діапазон 23.8...28.7 °C, коливання вологості 26...79 %, тиск 745.7...746.4 мм рт. ст., діапазон освітленості 25...3812 лк та пік TVOC 100 %).

На основі цих даних алгоритм автоматично вивів відповідні експлуатаційні рекомендації щодо необхідності кондиціонування, регулярного провітрювання приміщення та моніторингу рівня інсоляції. Важливо, що пристрій не перезавантажувався від переповнення буферів, що доводить відсутність витоків оперативної пам'яті (memory leaks) у вбудованому ПЗ.

Для перевірки відмовостійкості було примусово знеструмлено Wi-Fi роутер. Вбудоване ПЗ відреагувало штатно: мережевий збій на Core 0 не заблокував вимірювальне ядро Core 1. Пристрій перейшов в автономний режим (isOfflineMode = true), а на OLED-екрані замість адреси з'явилися інженерні нулі (IP: 0.0.0.0). Після відновлення живлення роутера, фонова задача з періодичністю у 30 секунд автоматично виконала функцію перепідключення, відновила зв'єднання та повернула на екран валідну IP-адресу, продовживши трансляцію телеметрії в хмару (процес динамічної зміни мережевого статусу наведено на рис. В.4 у Додатку В).

4.3 Порівняння з існуючими рішеннями

На основі отриманих результатів випробувань проведено оцінку ефективності запропонованого прототипу порівняно з комерційними та аналоговими системами, оглянутими в першому розділі.

Головною перевагою запропонованого комплексу є використання двоядерної архітектури FreeRTOS на мікроконтролері ESP32. У звичайних одноядерних рішеннях (наприклад, на базі ESP8266) відправка важких HTTPS-пакетів до Telegram Bot API або тривале очікування відповіді від хмарного сервера ThingSpeak повністю блокує процесор на 2–5 секунд. У цей час пристрій «зависає»: не зчитуються датчики, застигає екран, не працює світлова сигналізація, що є критично недопустимим для систем автоматизації та безпеки. Впроваджена архітектура паралельних обчислень дозволила повністю ізолювати мережевий стек на ядрі Core 0, завдяки чому вимірювальний цикл на ядрі Core 1 працює безперервно з чітким періодом у 3000 мс.

На відміну від комерційних датчиків закритого типу (Xiaomi, Sonoff), які вимірюють лише два базові параметри та потребують обов'язкової наявності дорогого фірмового шлюзу з прив’язкою до іноземних хмарних серверів, запропонований пристрій є абсолютно автономною станцією. Він поєднує в собі:

– моніторинг розширеного спектра з 5 параметрів, включаючи інтегральну оцінку забруднення повітря летючими речовинами (TVOC) та рівень інсоляції;

– локальний OLED-інтерфейс та інтелектуальну багаторівневу колірну індикацію станів;

– вбудовану енергонезалежну пам'ять LittleFS для локального збереження налаштувань без використання сторонніх серверів;

– модуль резервного живлення (ДБЖ), що гарантує роботу пристрою при вимкненнях електромережі.

З економічного погляду, собівартість компонентів прототипу становить близько 30 USD, що повністю конкурентно з базовими комерційними наборами, але надає користувачу відкрите програмне забезпечення з можливістю автоматичного аналізу відповідності адаптованим пороговим значенням на основі ДСН 3.3.6.042-99 та сезонного перемикання порогових значень.

Висновки за розділом 4

У четвертому розділі проведено повний цикл експериментальних випробувань апаратного прототипу, побудованого на основі вдосконалених методів і моделей. На основі отриманих аналітичних та графічних результатів розроблено й успішно реалізовано комплексну методику довготривалого 8-годинного тестування комплексу в умовах реального приміщення з фіксацією його апаратної та програмної поведінки як у штатному, так і в аварійному режимах роботи. Експериментальні дослідження повністю підтвердили високу стабільність функціонування вбудованого програмного забезпечення на базі двоядерної архітектури операційної системи реального часу FreeRTOS, а також практично довели відсутність витоків оперативної пам'яті (Free Heap) за умов тривалої безперервної обробки важкого мережевого HTTPS-трафіку під час взаємодії з серверами.

Разом із цим під час випробувань верифіковано коректність роботи алгоритмів програмного гістерезису тривог та цифрової фільтрації сигналу TVOC, що дозволило повністю уникнути хибних спрацювань засобів світлозвукової індикації та Telegram-сповіщень при коливаннях вимірювальних значень поблизу адаптованих порогових меж. Дослідження також підтвердили

високу відмовостійкість системи та ефективність неблокуючих алгоритмів відновлення бездротового зв'язку, де примусове відключення мережі Wi-Fi призводить до безпечного переходу пристрою в автономний режим із відображенням нульової IP-адреси на OLED-екрані, а відновлення сигналу маршрутизатора ініціює автоматичний фоновий reconnect без жодного збою чи затримки основного вимірювального циклу на обчислювальному ядрі. У підсумку результати натурних випробувань остаточно засвідчили інженерну та економічну перевагу запропонованого рішення над існуючими комерційними й аматорськими аналогами завдяки комплексності вимірів п'яти параметрів, повній автономності з використанням модуля ДБЖ та впровадженню адаптованих порогових значень на основі вітчизняних санітарних норм.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу вдосконалення методів побудови ефективної та надійної IoT-системи для безперервного моніторингу параметрів мікроклімату приміщень. Відповідно до поставленого завдання отримано такі основні результати:

1. На основі аналізу сучасних IoT-систем моніторингу параметрів мікроклімату приміщень виявлено ключові архітектурні недоліки комерційних пристроїв (закритість коду, обмежений спектр вимірів, висока вартість). Це дозволило теоретично та практично обґрунтувати доцільність вдосконалення методів побудови відкритого автономного комплексу з використанням адаптованих порогових значень на основі ДСН 3.3.6.042-99.

2. Проведено дослідження та обґрунтовано вибір апаратної платформи і датчиків для вимірювання параметрів мікроклімату. Базовим обчислювальним вузлом визначено мікроконтролер ESP32, а для комплексного збору даних інтегровано цифрові сенсори BME280 і BH1750 та аналоговий датчик MQ-135.

3. Розроблено структурну та функціональну модель IoT-системи моніторингу на базі мікроконтролера ESP32. Вона охоплює гнучку трирівневу архітектуру: Edge-рівень первинного збору телеметрії, рівень локального оброблення із застосуванням методів цифрової фільтрації та хмарний рівень віддаленого моніторингу та інформування.

4. Розроблено програмне забезпечення збору, обробки та передачі даних у бездротовій мережі. Вдосконалено модель багатозадачної обробки шляхом розроблення відмовостійкого вбудованого ПЗ мовою C++ на базі ОС FreeRTOS, що повністю ліквідувало проблему блокування вимірювального циклу під час виконання мережевих запитів.

5. Реалізовано засоби відображення та дистанційного моніторингу параметрів мікроклімату. Локальна індикація забезпечується OLED-дисплеєм, зумером і RGB-світлодіодом, а дистанційна взаємодія та хмарна візуалізація

реалізовані через асинхронний веб-сервер, сервіс ThingSpeak та Telegram-бот для оперативного інформування користувача про стан середовища.

6. Проведено тестування системи та аналіз ефективності її роботи під час безперервної 8-годинної стендової сесії в умовах реального приміщення. Проведено комплексні натурні випробування апаратного прототипу, побудованого на основі вдосконалених методів і моделей. Експериментально доведено високу стабільність роботи вбудованого ПЗ, відсутність витоків оперативної пам'яті та точність відпрацювання програмного гістерезису тривог, що підтверджує суттєву інженерну й економічну перевагу запропонованого рішення над існуючими аналогами.

Таким чином, запропонована IoT-система моніторингу параметрів мікроклімату повністю відповідає вимогам офіційного Технічного завдання та придатна для практичної експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень : затв. постановою Головного державного санітарного лікаря України від 01.12.1999 р. № 42 [Електронний ресурс]. – режим доступу: URL: https://zakon.rada.gov.ua/rada/show/va042282-99#Text (дата звернення – 09.03.2026).
2. Закон України «Про охорону праці» від 14 жовтня 1992 р. № 2694-XII. Відомості Верховної Ради України. 1992 [Електронний ресурс]. – режим доступу: URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення – 20.03.2026).
3. Ткачук К. Н., Халімовський М. О., Зацарний В. В. Основи охорони праці : підручник. 2-ге вид. Київ : Основа, 2006. 448 с.
4. Adafruit Industries. Adafruit BME280 Library. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/adafruit/Adafruit_BME280_Library (дата звернення – 10.03.2026).
5. Adafruit Industries. Adafruit SSD1306 OLED Driver Library. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/adafruit/Adafruit_SSD1306 (дата звернення – 10.03.2026).
6. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials. 2015. Vol. 17, No. 4. P. 2347–2376.
7. Barry R. Using the FreeRTOS Real Time Kernel. Real Time Engineers Ltd, 2010. 164 p.
8. Blanchon B. ArduinoJson v6 Documentation [Електронний ресурс]. – режим доступу: URL: https://arduinojson.org/v6/doc (дата звернення – 11.03.2026).
9. Bosch Sensortec. BME280 Combined Humidity and Pressure Sensor. Datasheet Rev. 1.6. Reutlingen : Bosch, 2020. 60 p.

10. Espressif Systems. Arduino-ESP32 Core 3.x Migration Guide [Електронний ресурс]. – режим доступу: URL: https://docs.espressif.com/projects/arduino-esp32/en/latest/migration_guides/2.x_to_3.0.html (дата звернення – 15.03.2026).
11. Espressif Systems. ESP32 Technical Reference Manual v5.3 [Електронний ресурс]. – режим доступу: URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (дата звернення – 05.03.2026).
12. Fang L., Clausen G., Fanger P. O. Impact of Temperature and Humidity on the Perception of Indoor Air Quality. Indoor Air. 1998. Vol. 8, No. 2. P. 80–90.
13. Greengard S. The Internet of Things. Cambridge : MIT Press, 2015. 232 p.
14. IEEE Std 802.11-2016. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems. New York : IEEE, 2016.
15. Laws C. BH1750 Ambient Light Sensor Library. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/claws/BH1750 (дата звернення – 10.03.2026).
16. Lough B. Universal Arduino Telegram Bot Library. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot (дата звернення – 10.03.2026).
17. MathWorks. ThingSpeak IoT Analytics Platform. Documentation [Електронний ресурс]. – режим доступу: URL: https://mathworks.com/help/thingspeak (дата звернення – 12.03.2026).
18. Me No Dev. ESPAsyncWebServer Library for ESP32. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/me-no-dev/ESPAsyncWebServer (дата звернення – 10.03.2026).
19. Mohanan V. DOIT ESP32 DevKit V1 Wi-Fi Development Board – Pinout Diagram & Arduino Reference. CIRCUITSTATE Electronics. 2022 [Електронний ресурс]. – режим доступу: URL:

https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/ (дата звернення – 30.04.2026).

20. Monk S. Programming Arduino: Getting Started with Sketches. 2nd ed. New York : McGraw-Hill Education, 2016. 192 p.
21. ROHM Semiconductor. BH1750FVI Digital 16bit Serial Output Type Ambient Light Sensor IC. Datasheet Rev. D. Tokyo : ROHM, 2011. 13 p.
22. Seppänen O., Fisk W. J., Lei Q. H. Effect of Temperature on Task Performance in Office Environment. Lawrence Berkeley National Laboratory, 2006. 10 p.
23. SONOFF Technologies. SNZB-02 ZigBee Temperature and Humidity Sensor. Офіційна специфікація [Електронний ресурс]. – режим доступу: URL: https://sonoff.tech/product/smart-home-security/snzb-02/ (дата звернення – 15.03.2026).
24. Tzapu, Tabakman T. WiFiManager Library for ESP32/ESP8266. GitHub [Електронний ресурс]. – режим доступу: URL: https://github.com/tzapu/WiFiManager (дата звернення – 10.03.2026).
25. Upton E., Halfacree G. Raspberry Pi User Guide. 4th ed. Hoboken : John Wiley & Sons, 2016. 312 p.
26. Zhengzhou Winsen Electronics Technology Co., Ltd. MQ-135 Gas Sensor: Manual (Ver. 1.4) [Електронний ресурс]. – режим доступу: URL: https://www.winsen-sensor.com/d/files/PDF/Semiconductor%20Gas%20Sensor/MQ135%20(Ver1.4)%20-%20Manual.pdf (дата звернення – 25.05.2026).

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

ЧУЄВА БОГДАНА ІГОРОВИЧА
(прізвище, ім'я, по батькові здобувача)

Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»
1. Тема роботи **ІОТ-СИСТЕМА МОНІТОРИНГУ ПАРАМЕТРІВ МІКРОКЛІМАТУ ПРИМІЩЕННЯ НА БАЗІ МІКРОКОНТРОЛЕРА ESP32**
керівник роботи **Бережний Артем Андрійович, старший викладач,**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання здобувачем роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) технічні характеристики мікроконтролера ESP32 та датчиків параметрів мікроклімату;
   2) програмні засоби розробки вбудованих систем та веб-інтерфейсів;
   3) вимоги до точності вимірювання температури, вологості, тиску, освітленості та рівня забруднення повітря;
   4) вимоги до бездротової передачі даних у реальному часі через мережу Wi-Fi;
   5) персональний комп'ютер для розробки програмного забезпечення та налагодження системи.
4. Перелік питань, які потрібно розробити:
   1) Аналіз сучасних IoT-систем моніторингу параметрів мікроклімату приміщень.
   2) Дослідження та обґрунтування вибору апаратної платформи і датчиків для вимірювання параметрів мікроклімату.
   3) Розробка структурної та функціональної моделі IoT-системи моніторингу на базі мікроконтролера ESP32.
   4) Розробка програмного забезпечення збору, обробки та передачі даних у бездротовій мережі.
   5) Реалізація засобів відображення та дистанційного моніторингу параметрів мікроклімату.
   6) Проведення тестування системи та аналіз ефективності її роботи.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Обґрунтування актуальності дослідження, визначення об'єкта, предмета, мети та завдань роботи	01.11.2025 – 18.11.2025
2.	Аналіз сучасних IoT-систем та технологій моніторингу мікроклімату в приміщеннях	18.11.2025 – 05.12.2025
3.	Огляд технологій бездротової передачі даних та платформ Інтернету речей для систем моніторингу	05.12.2025 – 20.12.2025
4.	Обґрунтування вибору апаратної бази (контролера та датчиків) системи	20.12.2025 – 10.01.2026
5.	Розробка структурної та функціональної моделей системи моніторингу	10.01.2026 – 25.01.2026
6.	Розробка архітектури програмного забезпечення системи Інтернету речей	25.01.2026 – 08.02.2026
7.	Розробка програмних модулів збору та обробки даних із датчиків	08.02.2026 – 25.02.2026
8.	Реалізація передачі інформації у бездротовій мережі Wi-Fi	25.02.2026 – 10.03.2026
9.	Розробка засобів дистанційного моніторингу та візуалізації параметрів	10.03.2026 – 25.03.2026
10.	Складання апаратної частини прототипу та налагодження взаємодії компонентів	25.03.2026 – 10.04.2026
11.	Експериментальне тестування системи та аналіз результатів	10.04.2026 – 20.04.2026
12.	Оцінка ефективності функціонування системи та формування висновків за результатами	20.04.2026 – 28.04.2026
13.	Написання та оформлення матеріалів кваліфікаційної роботи та оформлення пояснювальної записки	28.04.2026 – 10.05.2026
14.	Оформлення звіту про переддипломну практику	30.04.2026 – 10.05.2026
15.	Передзахист кваліфікаційної роботи	10.05.2026 – 20.05.2026
16.	Остаточне оформлення пояснювальної записки роботи та перевірка на антиплагіат	до 25.05.2026
17.	Отримання відгуку керівника та рецензії	до 30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Богдан ЧУЄВ
(підпис) (ім'я, прізвище)

Керівник роботи _____ Артем БЕРЕЖНИЙ
(підпис) (ім'я, прізвище)

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу «IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32»

1.	Введення	1.1. Назва: IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32. 1.2. Галузь застосування: Автоматизація та приладобудування, системи «Розумний дім», екологічний моніторинг.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/1245 від 28.05.2026. (Додаток А до пояснювальної записки).
3.	Призначення розробки	3.1. Мета розробки: вдосконалення методів побудови програмного забезпечення для безперервного моніторингу параметрів мікроклімату. 3.2. Призначення розробки: надає можливість автоматизовано визначати відповідність мікроклімату встановленим пороговим значенням на базі ДСН 3.3.6.042-99, своєчасно інформувати про аварійні відхилення (перегрів, різке погіршення якості повітря) та здійснювати хмарну аналітику. 3.3. Вихідні дані: технічні характеристики мікроконтролера ESP32 та набору датчиків (BME280, BH1750, MQ-135).
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: збір даних з датчиків, відображення на локальному OLED-дисплеї та веб-дашборді, відправка тривожних сповіщень у Telegram-бот, логування даних на платформу ThingSpeak. 4.2. Вимоги до надійності: забезпечувати стабільну роботу на базі ОС FreeRTOS без блокувань; зберігати налаштування у файловій системі LittleFS. 4.3.Вимоги до умов експлуатації: наявність покриття мережі Wi-Fi 2.4 ГГц.

<table>
<tr><td></td><td></td><td>4.4. Вимоги до складу і параметрів технічних засобів: мікроконтролер ESP32, датчики BME280, BH1750, MQ-135, дисплей SSD1306, ДБЖ.
4.5. Вимоги до інформаційної та програмної сумісності: сумісність з Telegram API та REST API ThingSpeak.
4.6. Вимоги до маркування та упаковки: немає
4.7. Вимоги до транспортування і зберігання: немає.
4.8. Спеціальні вимоги: немає.</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td>Програмною документацією до виробу «IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Програму і методику випробувань (у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).
3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td>Програмною документацією до виробу «IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Опис архітектури програмного забезпечення та алгоритмів збору й обробки даних (у вигляді підрозділів 3.2 та 3.3 пояснювальної записки до кваліфікаційної роботи).
3) Аналіз ефективності системи (представити в розділі 4 пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td>
<table>
<tr><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>01.11.2025 – 18.11.2025</td><td>Обґрунтування актуальності дослідження, визначення об’єкта, предмета, мети та завдань роботи</td></tr>
<tr><td>18.11.2025 – 05.12.2025</td><td>Аналіз сучасних IoT-систем та технологій моніторингу мікроклімату в приміщеннях</td></tr>
<tr><td>05.12.2025 – 20.12.2025</td><td>Огляд технологій бездротової передачі даних та платформ</td></tr>
</table>
</td></tr>
</table>

			Інтернету речей для систем моніторингу
		20.12.2025 – 10.01.2026	Обґрунтування вибору апаратної бази (контролера та датчиків) системи
		10.01.2026 – 25.01.2026	Розробка структурної та функціональної моделей системи моніторингу
		25.01.2026 – 08.02.2026	Розробка архітектури програмного забезпечення системи Інтернету речей
		08.02.2026 – 25.02.2026	Розробка програмних модулів збору та обробки даних із датчиків
		25.02.2026 – 10.03.2026	Реалізація передачі інформації у бездротовій мережі Wi-Fi
		10.03.2026 – 25.03.2026	Розробка засобів дистанційного моніторингу та візуалізації параметрів
		25.03.2026 – 10.04.2026	Складання апаратної частини прототипу та налагодження взаємодії компонентів
		10.04.2026 – 20.04.2026	Експериментальне тестування системи та аналіз результатів
		20.04.2026 – 28.04.2026	Оцінка ефективності функціонування системи та формування висновків за результатами
		28.04.2026 – 10.05.2026	Написання та оформлення матеріалів кваліфікаційної роботи та оформлення пояснювальної записки
		30.04.2026 – 10.05.2026	Оформлення звіту про переддипломну практику
		10.05.2026 – 20.05.2026	Передзахист кваліфікаційної роботи
		до 25.05.2026	Остаточне оформлення пояснювальної записки роботи та перевірка на антиплагіат
		до	Отримання відгуку керівника та

		30.05.2026	рецензії
8.	Порядок контролю і приймання програмного продукту	1. Контроль виконання згідно календарного плану; 2. Захист на засіданні Екзаменаційної комісії; 3. Подання пояснювальної записки у паперовому та електронному вигляді.	

Виконавець
студент групи КУ- 41
ЧУЄВ Б. І.

Замовник
старший викладач
БЕРЕЖНИЙ А. А.

Додаток В

Програма і методика випробувань програмного виробу

«IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32»

1. Об'єкт випробувань

1. Назва програмного виробу: «IoT-система моніторингу параметрів мікроклімату приміщення на базі мікроконтролера ESP32».
2. Галузь застосування: Автоматизація та приладобудування, системи «Розумний дім», екологічний моніторинг.
3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності запропонованого апаратно-програмного комплексу заявленим функціональним можливостям та технічним вимогам, наведеним у Технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є затверджене Технічне завдання та календарний план виконання кваліфікаційної роботи.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться в умовах реальної експлуатації (офісне приміщення / комп'ютерний клас кафедри) в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі, що відповідає цій програмі і методиці випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Система повинна задовольняти наступним вимогам:

1. Здійснювати безперервне зчитування даних із датчиків (BME280, BH1750, MQ-135) з періодичністю 3 секунди без блокування основного потоку виконання;
2. забезпечувати стабільну роботу на базі двоядерної архітектури ОС FreeRTOS;
3. формувати та відображати веб-дашборд через вбудований асинхронний веб-сервер при підключенні через локальну мережу Wi-Fi;
4. автоматично керувати локальною світлозвуковою індикацією (RGB-світлодіод та зуммер) залежно від відповідності параметрів адаптованим пороговим значенням на основі ДСН 3.3.6.042-99;
5. здійснювати інтелектуальну відправку тривожних сповіщень та аналітичних звітів через Telegram API;
6. забезпечувати автоматичне відновлення роботи (відмовостійкість) при тимчасовій втраті підключення до бездротової мережі Wi-Fi.

5. Вимоги до програмної документації

Програмною документацією щодо розроблюваного продукту вважати:

1. Справжнє Технічне завдання на розробку (Додаток Б);
2. Програму і методику випробувань (Додаток В);
3. Опис архітектури та алгоритмів функціонування (Розділ 3 пояснювальної записки);
4. Аналіз ефективності та результати тестування (Розділ 4 пояснювальної записки);
5. Текст вихідного коду програми (Додаток Г).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідні:

- зібраний апаратний прототип пристрою на базі ESP32 з підключеними датчиками;
- бездротова мережа Wi-Fi (2.4 ГГц) з доступом до мережі Інтернет;
- персональний комп'ютер або смартфон із встановленим веб-браузером;
- встановлений месенджер Telegram для взаємодії з ботом системи.

6.2 Порядок проведення випробувань

Випробування апаратно-програмного комплексу проводяться у два послідовні етапи:

-ознайомчий (1-й етап);

-випробування програмно-апаратного виробу (2-й етап).

Перелік перевірок, що проводяться на 1-му етапі (ознайомчому) випробувань, включає в себе:

1. Перевірку комплектності програмної та технічної документації (наявність Технічного завдання, Програми і методики випробувань та повного вихідного тексту програми).
2. Перевірку комплектності та готовності складу апаратних засобів (надійність з'єднання датчиків BME280, BH1750, MQ-135, реле, OLED-дисплея та RGB-індикатора з платою мікроконтролера ESP32).
3. Перевірку працездатності джерела безперебійного живлення (ДБЖ) та наявності стабільного покриття бездротової локальної мережі Wi-Fi з частотою 2.4 ГГц.

Якість програмної документації перевіряється на відповідність вимогам чинних стандартів.

Перелік перевірок, що проводяться на 2-му етапі (безпосередні випробувань) випробувань, включає в себе:

1. Перевірку відповідності технічних та експлуатаційних характеристик програми вимогам, заявленим у Технічному завданні.
2. Перевірку ступеня виконання функціональних вимог до вбудованого ПЗ (коректність збору телеметрії, обробки критичних секцій FreeRTOS, спрацювання алгоритмів індикації та логування).

Порядок безпосереднього проведення перевірок на 2-му етапі:

1. Подати живлення на пристрій, перевівши механічний тумблер MTS-102 на лінії живлення від ДБЖ у робоче положення.
2. Дочекатися завершення апаратної ініціалізації периферії та підключення до Wi-Fi. На локальному OLED-дисплеї має відобразитися призначена IP-адреса (наприклад, 192.168.0.100).
3. Запустити на клієнтському ПК або смартфоні веб-браузер та месенджер Telegram для взаємодії з верифікаційними інтерфейсами системи.

Для підтвердження повноти реалізації функціоналу пропонується виконати Тест 1, Тест 2, Тест 3 та Тест 4.

Тест 1

1. Перевірка роботи локального асинхронного веб-сервера на ядрі Core 0 та коректності динамічного виведення телеметрії;
2. введення локальної IP-адреси пристрою у веб-браузері клієнтського ПК;
3. отримання відповіді від мікроконтролера у вигляді HTML-дашборду «Офіс 1» з поточними фізичними показниками, що асинхронно оновлюються кожні 3 секунди.

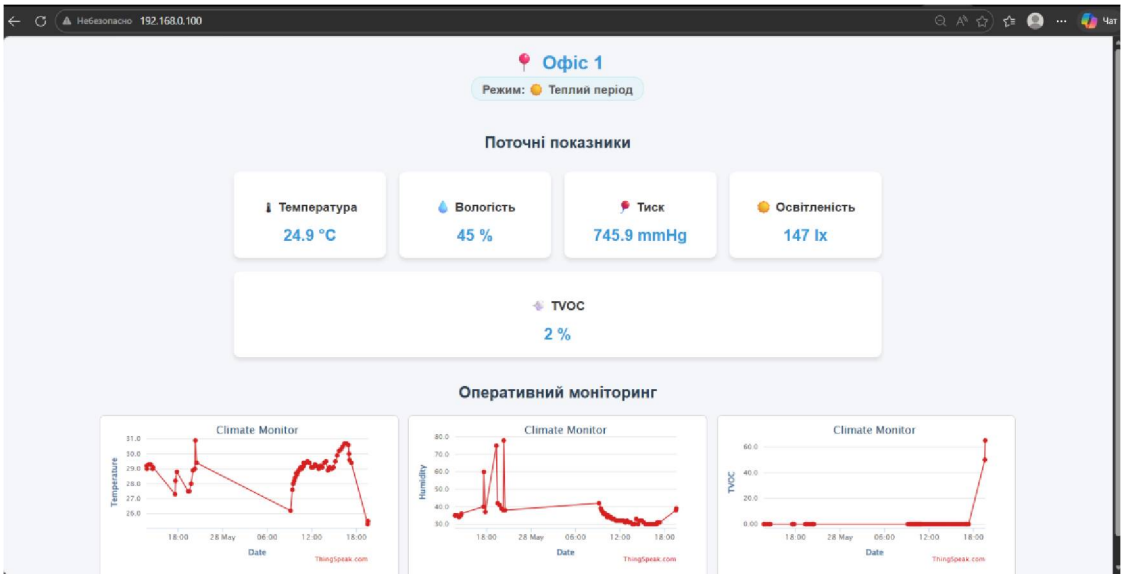


Рис. В.1 – Тест 1. Відображення локального веб-інтерфейсу з поточними показниками.

Тест 2

1. Перевірка алгоритму програмного гістерезису тривог, критичних секцій FreeRTOS та світлозвукових оповіщень;
2. штучне створення умов перевищення адаптованих порогових значень на основі ДСН 3.3.6.042-99 (штучне нагрівання датчика або імітація високої концентрації летких органічних сполук для досягнення рівня TVOC > 35 %);
3. отримання миттєвого тривожного сповіщення у Telegram-боті та автоматична зміна кольору RGB-індикатора на пристрої з активацією зуммера.



Рис. В.2 – Тест 2. Спрацювання апаратної індикації та програмних тривог.

Тест 3

1. Перевірка стабільності накопичення бази даних та коректності збору екстремумів статистичним модулем;
2. відправлення команди /report в інтерфейсі Telegram-бота після 8 годин безперервної роботи системи;
3. отримання від сервера розгорнутого аналітичного звіту з фіксацією часу роботи, екстремумів показників та автоматичними експертними рекомендаціями.

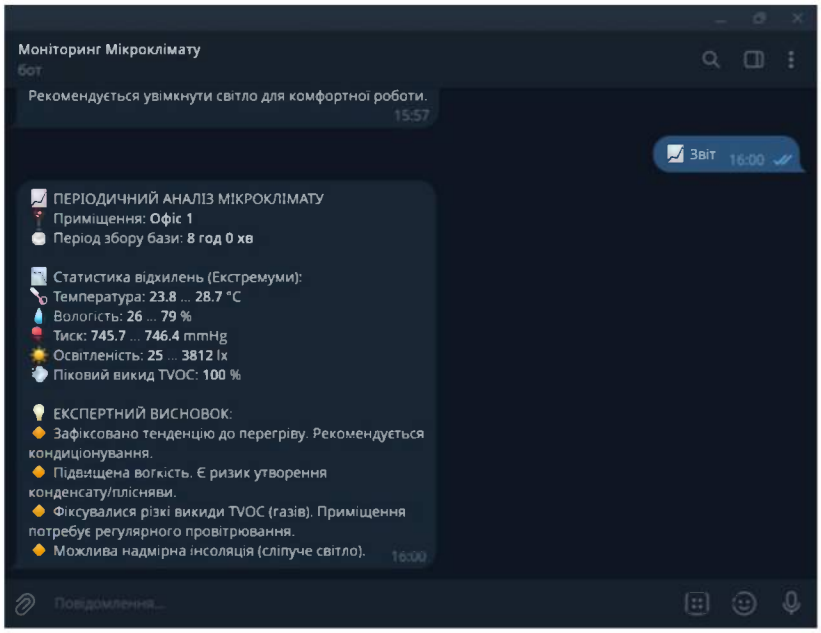


Рис. В.3 – Тест 3. Сформований аналітичний кумулятивний звіт у месенджері.

Тест 4

1. Перевірка відмовостійкості системи та неблокуючого фонового алгоритму відновлення Wi-Fi зв'язку;
2. примусове вимкнення живлення маршрутизатора під час штатної роботи комплексу з подальшим його увімкненням;
3. перехід локальної IP-адреси на OLED-дисплеї у значення 0.0.0.0 з подальшим автоматичним відновленням коректної IP-адреси без зависання основного обчислювального циклу на ядрі Core 1.



Рис. В.4 – Тест 4. Динамічна зміна мережевого статусу та IP-адреси на локальному OLED-дисплеї.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування, тест 3 успішно пройшов випробування і тест 4 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КУ-41, Чуєв Б. І. _____

Додаток Г

Лістинг вихідного коду програмного забезпечення мікроконтролера ESP32

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_BME280.h>
#include <BH1750.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>
#include <WiFiManager.h>
#include <LittleFS.h>
#include <HTTPClient.h>

// --- ЗМІННІ ДЛЯ ЗБЕРЕЖЕННЯ НАЛАШТУВАНЬ ---
char telegram_token[60] = "";
char telegram_chat_id[30] = "";
char room_name[30] = "Кімната 1";
bool shouldSaveConfig = false;

// --- БЕЗПЕЧНЕ ЗБЕРЕЖЕННЯ ---
bool pendingConfigSave = false;
unsigned long saveConfigTime = 0;

// --- СТАТИСТИКА ТА АВТОЗВІТИ (АНАЛІТИКА) ---
unsigned long bootTime = 0;
float minT = 99.0, maxT = -99.0;
float minH = 99.0, maxH = -1.0;
float minP = 2000.0, maxP = 0.0;
int minLux = 99999, maxLux = -1;
int maxTVOC = 0;

unsigned long autoReportIntervalMillis = 0; // 0 = автозвіти
вимкнено
unsigned long lastAutoReportTime = 0;

// --- НАЛАШТУВАННЯ ХМАРИ THINGSPEAK ---
String thingSpeakKey = "TOKEN";
unsigned long lastLogTime = 0;
const unsigned long LOG_INTERVAL = 900000;        // 15 хвилин
— плановий запис
unsigned long lastAlarmLogTime = 0;                // остання
тривожна відправка
// Бітові прапори: яка тривога вже зафіксована на хмарі в цьому
циклі
```

```
    uint8_t alarmLoggedFlags = 0;                    // біт0=газ,
біт1=темп, біт2=волог
    #define FLAG_GAS  0x01
    #define FLAG_TEMP 0x02
    #define FLAG_HUM  0x04

    // --- ФОНОВЕ ПЕРЕПІДКЛЮЧЕННЯ WIFI ---
    unsigned long lastWiFiCheckTime = 0;
    const unsigned long WIFI_CHECK_INTERVAL = 30000; // Перевіряти
кожні 30 секунд
    bool isOfflineMode = false;

    AsyncWebServer server(80);
    WiFiClientSecure client_tg;
    UniversalTelegramBot* bot;

    // --- НАЛАШТУВАННЯ ПІНІВ ---
    #define SCREEN_WIDTH 128
    #define SCREEN_HEIGHT 64

    const int GAS_PIN = 34;
    const int BUZZER_PIN = 25;
    const int RELAY_PIN = 18;
    const int BTN_PIN = 13;

    const int LED_R_PIN = 26;
    const int LED_G_PIN = 27;
    const int LED_B_PIN = 14;

    Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -
1);
    Adafruit_BME280 bme;
    BH1750 lightMeter;

    // --- ЗМІННІ СТАНІВ ДАТЧИКІВ ---
    int screenMode = 0;
    unsigned long lastSensorUpdate = 0;

    float temp = 0.0;
    float hum = 0.0;
    float pres = 0.0;
    float lux = 0.0;
    int pollutionPct = 0;

    // --- АДАПТОВАНІ ПОРОГОВІ ЗНАЧЕННЯ НА ОСНОВІ ДСН 3.3.6.042-99
(Категорія Іа) ---
    bool isWarmSeason = true;
    float currentMinTemp = 21.0;
    float currentMaxTemp = 28.0;
    float currentMinHum = 15.0;
    float currentMaxHum = 75.0;
```

```
// --- ЗМІННІ ТРИВОГ ТА СНАПШОТИ ---
const unsigned long TELEGRAM_COOLDOWN = 300000;

// --- М'ЮТЕКС ДЛЯ ЗАХИСТУ МІЖЯДЕРНОГО ОБМІНУ ---
portMUX_TYPE dataMux = portMUX_INITIALIZER_UNLOCKED;

// Знімок даних датчиків для безпечного читання з Telegram-таски (Core 0)
struct SensorSnapshot {
  float temp;
  float hum;
  float pres;
  float lux;
  int   pollutionPct;
} sensorSnap = {0, 0, 0, 0, 0};

// Структура флагів тривог — оновлюється тільки з Core 1, читається з Core 0
struct AlertFlags {
  bool needTelegramAlert   = false;
  bool needTelegramNormal  = false;
  int  alertPollutionLevel = 0;

  bool  needTempAlert  = false;
  bool  needTempNormal = false;
  float alertTempLevel = 0.0;

  bool  needHumAlert  = false;
  bool  needHumNormal = false;
  float alertHumLevel = 0.0;

  bool needLuxHighAlert = false;
  bool needLuxLowAlert  = false;
  int  alertLuxLevel    = 0;
} alerts;

bool isGasAlarmState = false;
unsigned long lastTelegramAlertTime = 0;
unsigned long gasAlarmStartTime = 0;

bool isTempAlarmState = false;
unsigned long tempAlarmStartTime = 0;

bool isHumAlarmState = false;
unsigned long humAlarmStartTime = 0;

bool isLuxHighAlarmState = false;
bool isLuxLowAlarmState = false;

TaskHandle_t TelegramTask;

void setRGB(int r, int g, int b) {
```

```
      ledcWrite(LED_R_PIN, r); // Core 3.x: ledcWrite по піну
      ledcWrite(LED_G_PIN, g);
      ledcWrite(LED_B_PIN, b);
    }

    void saveConfigCallback() {
      shouldSaveConfig = true;
    }

    void logToThingSpeak() {
      if (WiFi.status() != WL_CONNECTED || thingSpeakKey.length()
<= 10) return;

      // Знімаємо локальні копії під мютексом, щоб не тримати лок
під час HTTP
      float t, h; int p;
      portENTER_CRITICAL(&dataMux);
        t = sensorSnap.temp;
        h = sensorSnap.hum;
        p = sensorSnap.pollutionPct;
      portEXIT_CRITICAL(&dataMux);

      HTTPClient http;
       http.setConnectTimeout(5000); // 5 сек — максимальний час
з'єднання
      http.setTimeout(8000);          // 8 сек — максимальний час
відповіді
      char url[200];
      snprintf(url, sizeof(url),
         "http://api.thingspeak.com/update?api_key=%s&field1=%.1f&
field2=%.0f&field3=%d",
        thingSpeakKey.c_str(), t, h, p);
      http.begin(url);
      int httpCode = http.GET();
      http.end();
      Serial.printf("ThingSpeak: %d\n", httpCode);
    }

    // --- ФУНКЦІЯ ГЕНЕРАЦІЇ АНАЛІТИЧНОГО ЗВІТУ ---
    // Викликається з Telegram-таски (Core 0) — читає тільки через
знімок
    String generateReport() {
      // Захищений знімок поточних даних і статистики
       float rMinT, rMaxT, rMinH, rMaxH, rMinP, rMaxP, rCurTemp,
rCurHum, rCurPres;
      int rMinLux, rMaxLux, rMaxTVOC, rCurPollution;
      unsigned long rBootTime;

      portENTER_CRITICAL(&dataMux);
        rMinT = minT; rMaxT = maxT;
        rMinH = minH; rMaxH = maxH;
        rMinP = minP; rMaxP = maxP;
```

```
          rMinLux = minLux; rMaxLux = maxLux;
          rMaxTVOC = maxTVOC;
          rCurTemp = sensorSnap.temp;
          rCurHum  = sensorSnap.hum;
          rCurPres = sensorSnap.pres;
          rCurPollution = sensorSnap.pollutionPct;
          rBootTime = bootTime;
          // Скидання лічильників під тим самим мютексом
          minT = rCurTemp; maxT = rCurTemp;
          minH = rCurHum;  maxH = rCurHum;
          minP = rCurPres; maxP = rCurPres;
          minLux = (int)sensorSnap.lux; maxLux = (int)sensorSnap.lux;
          maxTVOC = rCurPollution;
          bootTime = millis();
        portEXIT_CRITICAL(&dataMux);

        unsigned long uptime = (millis() - rBootTime) / 60000;
        int hours = uptime / 60;
        int mins = uptime % 60;

        String msg = "📈 **ПЕРІОДИЧНИЙ АНАЛІЗ МІКРОКЛІМАТУ**\n";
        msg += "📍 Приміщення: *" + String(room_name) + "*\n";
        msg += "⏱️ Період збору бази: *" + String(hours) + " год " +
String(mins) + " хв*\n\n";

        msg += "📉 **Статистика відхилень (Екстремуми):**\n";
        msg += "🌡️ Температура: *" + String(rMinT, 1) + "* ... *" +
String(rMaxT, 1) + "* °C\n";
        msg += "💧 Вологість: *" + String(rMinH, 0) + "* ... *" +
String(rMaxH, 0) + "* %\n";
        msg += "🎈 Тиск: *" + String(rMinP, 1) + "* ... *" +
String(rMaxP, 1) + "* mmHg\n";
        msg += "🌕 Освітленість: *" + String(rMinLux) + "* ... *" +
String(rMaxLux) + "* lx\n";
        msg += "💨 Піковий викид TVOC: *" + String(rMaxTVOC) + "*
%\n\n";

        msg += "💡 **ЕКСПЕРТНИЙ ВИСНОВОК:**\n";
        bool hasIssues = false;

        if (rMaxT > currentMaxTemp) { msg += "🔸 Зафіксовано тенденцію
до **перегріву**. Рекомендується кондиціонування.\n"; hasIssues =
true; }
        if (rMinT < currentMinTemp) { msg += "🔸 Ризик
**переохолодження** приміщення. Перевірте опалення.\n"; hasIssues =
true; }
        if (rMaxH > currentMaxHum) { msg += "🔸 Підвищена
**вогкість**. Є ризик утворення конденсату/плісняви.\n"; hasIssues
= true; }
```

```
        if (rMinH < currentMinHum) { msg += "🔸 Повітря **пересушене**. Рекомендується зволожувач.\n"; hasIssues = true; }
       if (rMaxTVOC > 35) { msg += "🔸 Фіксувалися різкі викиди **TVOC** (газів). Приміщення потребує регулярного провітрювання.\n"; hasIssues = true; }
      if (rMaxLux > 800) { msg += "🔸 Можлива **надмірна інсоляція** (сліпуче світло).\n"; hasIssues = true; }
       if (rMaxLux < 100 && hours > 0) { msg += "🔸 Спостерігається хронічна **нестача освітлення**.\n"; hasIssues = true; }

       if (!hasIssues) {
         msg += "✅ Клімат ідеально стабільний. Відхилень від норм ДСН за звітний період не виявлено.";
       }

       return msg;
     }

     const char index_html[] PROGMEM = R"rawliteral(
     <!DOCTYPE html>
     <html>
     <head>
         <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
         <title>Моніторинг Мікроклімату</title>
         <style>
              body { font-family: Arial, sans-serif; text-align: center; background: #f4f6f9; color: #333; margin: 0; padding: 20px; }
             h2 { color: #2c3e50; margin-top: 30px; }
              .room-title { color: #3498db; font-size: 28px; font-weight: bold; margin-bottom: 5px; }
              .season-badge { font-size: 18px; font-weight: bold; margin-bottom: 20px; color: #555; background: #e8f4f8; display: inline-block; padding: 5px 15px; border-radius: 15px; border: 1px solid #bce8f1;}
             .container { display: flex; flex-wrap: wrap; justify-content: center; max-width: 1000px; margin: 0 auto; }
            .card { background: white; padding: 20px; margin: 10px; border-radius: 10px; box-shadow: 0 4px 6px rgba(0,0,0,0.1); min-width: 180px; flex: 1; }
              .value { font-size: 24px; font-weight: bold; color: #3498db; margin-top: 10px; }
             .card.alarm { border: 2px solid #e74c3c; box-shadow: 0 0 10px #e74c3c; }
               #charts-wrapper { display: flex; flex-wrap: wrap; justify-content: center; gap: 15px; margin-top: 20px; }
            iframe { border: 1px solid #cccccc; border-radius: 8px; background: white; }
         </style>
```

```
    </head>
    <body>
                    <div       class="room-title"       id="val-
room">Завантаження...</div>
            <div   class="season-badge"   id="val-season">Оновлення
статусу...</div>

        <h2>Поточні показники</h2>
        <div class="container">
                  <div   class="card"><h3>🌡️   Температура</h3><div
class="value" id="val-temp">0.0 °C</div></div>
                   <div   class="card"><h3>💧   Вологість</h3><div
class="value" id="val-hum">0 %</div></div>
              <div class="card"><h3>🎈 Тиск</h3><div class="value"
id="val-pres">0 mmHg</div></div>
                  <div   class="card"><h3>🌞   Освітленість</h3><div
class="value" id="val-lux">0 lx</div></div>
             <div id="card-tvoc" class="card"><h3>🍽️ TVOC</h3><div
class="value" id="val-tvoc">0 %</div></div>
        </div>

        <h2>Оперативний моніторинг</h2>
        <div id="charts-wrapper">
                        <iframe    width="450"    height="260"
src="https://thingspeak.com/channels/3390717/charts/1?bgcolor=%23f
fffff&color=%23d62020&dynamic=true&results=60&type=line&update=15"
></iframe>
                        <iframe    width="450"    height="260"
src="https://thingspeak.com/channels/3390717/charts/2?bgcolor=%23f
fffff&color=%23d62020&dynamic=true&results=60&type=line&update=15"
></iframe>
                        <iframe    width="450"    height="260"
src="https://thingspeak.com/channels/3390717/charts/3?bgcolor=%23f
fffff&color=%23d62020&dynamic=true&results=60&type=line&update=15"
></iframe>
        </div>

        <script>
            setInterval(function() {
                                fetch('/data').then(response    =>
response.json()).then(data => {
                    document.getElementById('val-room').innerText
= "📍 " + data.room;
                    document.getElementById('val-temp').innerText
= data.temp + ' °C';
                   document.getElementById('val-hum').innerText =
data.hum + ' %';
                    document.getElementById('val-pres').innerText
= data.pres + ' mmHg';
                   document.getElementById('val-lux').innerText =
data.lux + ' lx';
```

```
                        document.getElementById('val-tvoc').innerText
= data.tvoc + ' %';

                    if(data.season) {
                                document.getElementById('val-
season').innerHTML = "Режим: 🌞 Теплий період";
                    } else {
                                document.getElementById('val-
season').innerHTML = "Режим: ❄️ Холодний період";
                    }

                    if(data.tvoc > 35) {
                                document.getElementById('card-
tvoc').classList.add('alarm');
                    } else {
                                document.getElementById('card-
tvoc').classList.remove('alarm');
                    }
                });
            }, 3000);
        </script>
    </body>
    </html>
    )rawliteral";

    // --- БРОНЬОВАНИЙ СЕЙФ ВІД ДУБЛІВ ---
    RTC_DATA_ATTR int last_processed_msg_id = 0;

    void handleNewMessages(int numNewMessages) {
      String mainKeyboard = "[[\"📊 Статус\", \"🚨 Тривоги\"],
[\"📈 Звіт\", \"📅 Сезон\"], [\"🌐 IP-адреса\", \"⏱️ Автозвіти\"],
[\"💡 Індикація\"]]";

      for (int i = 0; i < numNewMessages; i++) {
        String chat_id = String(bot->messages[i].chat_id);
        if (chat_id != String(telegram_chat_id)) continue;

        int current_msg_id = bot->messages[i].message_id;
        if (current_msg_id <= last_processed_msg_id) {
          continue;
        }
        last_processed_msg_id = current_msg_id;

        String text = bot->messages[i].text;

        if (text == "/start" || text == "🔙 Головне меню") {
              String welcome = "🤖 **Система Моніторингу
Мікроклімату**\n\n";
          welcome += "Пристрій: *" + String(room_name) + "*\n\n";
          welcome += "Використовуйте кнопки нижче для швидкого
керування 👇";
```

```
            bot->sendMessageWithReplyKeyboard(chat_id,  welcome,
"Markdown", mainKeyboard, true);
        }

        else if (text == "/season" || text == "🗓️ Сезон") {
          // Захист: isWarmSeason і норми читаються з Core 1 в
sensor loop
          portENTER_CRITICAL(&dataMux);
            isWarmSeason = !isWarmSeason;
            if (isWarmSeason) {
              currentMinTemp = 21.0; currentMaxTemp = 28.0;
            } else {
              currentMinTemp = 20.0; currentMaxTemp = 25.0;
            }
            isTempAlarmState = false; isHumAlarmState = false;
             alerts.needTempNormal = false; alerts.needTempAlert =
false;
             alerts.needHumNormal = false;  alerts.needHumAlert =
false;
          portEXIT_CRITICAL(&dataMux);

            // pendingConfigSave/saveConfigTime — прості булеви,
атомарні на 32-біт архітектурі
          pendingConfigSave = true;
          saveConfigTime = millis() + 2000;

           String msg = "🗓️ **Перемикання профілю ДСН 3.3.6.042-
99**\n\n";
          if (isWarmSeason) {
            msg += "Встановлено: 🌞 **Теплий період року**\nНорма
температури: **21.0 - 28.0 °C**\nНорма вологості: **15 - 75%**";
          } else {
            msg += "Встановлено: ❄️ **Холодний період року**\nНорма
температури: **20.0 - 25.0 °C**\nНорма вологості: **15 - 75%**";
          }
                bot->sendMessageWithReplyKeyboard(chat_id,   msg,
"Markdown", mainKeyboard, true);
        }

        else if (text == "/ip" || text == "🌐 IP-адреса") {
             String  ipMsg  =  "🌐  Адреса  веб-дашборду  ["  +
String(room_name) + "]: http://" + WiFi.localIP().toString();
          bot->sendMessage(chat_id, ipMsg, "");
        }

        else if (text == "/status" || text == "📊 Статус") {
          // Захищений знімок — читаємо з Core 0 безпечно
          float st, sh, sp, sl; int spol;
          portENTER_CRITICAL(&dataMux);
            st = sensorSnap.temp; sh = sensorSnap.hum;
```

```
            sp = sensorSnap.pres; sl = sensorSnap.lux;
            spol = sensorSnap.pollutionPct;
          portEXIT_CRITICAL(&dataMux);
          String msg = "📊 **СТАН ПРИМІЩЕННЯ: " + String(room_name)
+ "**\n";
           msg += "📅 Сезон (ДСН): " + String(isWarmSeason ? "🌞
Теплий" : "❄️ Холодний") + "\n\n";
          msg += "🌡️ Температура: " + String(st, 1) + " °C\n";
          msg += "💧 Вологість: " + String(sh, 0) + " %\n";
          msg += "🎈 Тиск: " + String(sp, 1) + " mmHg\n";
          msg += "🌞 Освітленість: " + String((int)sl) + " lx\n";
          msg += "🌫️ Забруднення (TVOC): " + String(spol) + " %\n";
          bot->sendMessage(chat_id, msg, "Markdown");
        }

        else if (text == "/alerts" || text == "🚨 Тривоги") {
          float at, ah, al; int apol;
          bool gas, tmp, hm, lhi, llo;
          portENTER_CRITICAL(&dataMux);
            at = sensorSnap.temp; ah = sensorSnap.hum;
            al = sensorSnap.lux; apol = sensorSnap.pollutionPct;
            gas = isGasAlarmState; tmp = isTempAlarmState;
            hm = isHumAlarmState; lhi = isLuxHighAlarmState; llo =
isLuxLowAlarmState;
          portEXIT_CRITICAL(&dataMux);
          String msg = "🚨 **АКТИВНІ ПОПЕРЕДЖЕННЯ ТА ТРИВОГИ**\n📍
Приміщення: *" + String(room_name) + "*\n\n";
          bool hasAlerts = false;

           if (gas) { msg += "🔴 **КРИТИЧНО:** Високий рівень TVOC
(" + String(apol) + "%). Потрібно провітрити!\n"; hasAlerts = true;
}
           if (tmp) { msg += "🟡 **ТРИВОГА:** Температурний режим
порушено (" + String(at, 1) + " °C). Норма ДСН: " +
String(currentMinTemp, 0) + "-" + String(currentMaxTemp, 0) + "
°C.\n"; hasAlerts = true; }
           if (hm) { msg += "🟣 **ТРИВОГА:** Рівень вологості
порушено (" + String(ah, 0) + " %). Норма ДСН: " +
String(currentMinHum, 0) + "-" + String(currentMaxHum, 0) + " %.\n";
hasAlerts = true; }
           if (lhi) { msg += "🌞 **УВАГА:** Надмірна освітленість
(" + String((int)al) + " lx). Можливо, сліпить.\n"; hasAlerts = true;
}
          else if (llo) { msg += "🌑 **УВАГА:** Низька освітленість
(" + String((int)al) + " lx). Рекомендується увімкнути світло.\n";
hasAlerts = true; }

          if (!hasAlerts) {
```

```
            msg = "✅ **ВСЕ В НОРМІ**\n📍 Приміщення: *" +
String(room_name) + "*\n\nПоточні показники знаходяться в межах
норми ДСН та санітарних вимог.";
          }
          bot->sendMessage(chat_id, msg, "Markdown");
        }

        else if (text == "/report" || text == "📈 Звіт") {
          bot->sendMessage(chat_id, generateReport(), "Markdown");
        }

        // --- ПІДМЕНЮ АВТОЗВІТІВ ---
        else if (text == "/auto" || text == "⏱️ Автозвіти") {
          String msg = "⏱️ **Налаштування Автозвітів**\n\n";
           msg += "Оберіть бажаний інтервал отримання аналітичних
звітів:\n\n";
                msg += "Поточний інтервал: *" +
String(autoReportIntervalMillis > 0 ?
String(autoReportIntervalMillis / 3600000UL) + " год" : "Вимкнено")
+ "*";

           String autoKeyboard = "[[\"/auto 1\", \"/auto 12\"],
[\"/auto 24\", \"/auto 0\"], [\"🔙 Головне меню\"]]";
                bot->sendMessageWithReplyKeyboard(chat_id, msg,
"Markdown", autoKeyboard, true);
        }
        else if (text.startsWith("/auto ")) {
          int hours = text.substring(6).toInt();
          if (hours == 0) {
            autoReportIntervalMillis = 0;
                 bot->sendMessageWithReplyKeyboard(chat_id, "🔴
Автоматичні експертні звіти **вимкнено**.", "Markdown",
mainKeyboard, true);
          } else {
            autoReportIntervalMillis = hours * 3600000UL;
                 bot->sendMessageWithReplyKeyboard(chat_id, "⏱️
Автоматичні експертні звіти налаштовано: раз на **" + String(hours)
+ " год.**", "Markdown", mainKeyboard, true);
            lastAutoReportTime = millis();
          }
           pendingConfigSave = true; saveConfigTime = millis() +
1000;
        }

        else if (text == "/led" || text == "💡 Індикація") {
          String msg = "💡 **Індикація пристрою:**\n\n";
          msg += "🟢 **Зелений:** Ідеально (все в нормі)\n";
          msg += "🔵 **Синій:** Робочий режим\n";
          msg += "🟡 **Жовтий:** Температурна тривога\n";
```

```
            msg += "🟣 **Фіолетовий:** Тривога вологості\n";
            msg += "🔴 **Червоний:** КРИТИЧНО (ГАЗ!)\n\n";
             msg += "*(Блимання = тривога щойно спрацювала. Суцільне світло = тривога активна, але не нова)*";
            bot->sendMessage(chat_id, msg, "Markdown");
          }
        }
      }

      void telegramTaskCode( void * pvParameters ){
        for(;;){
          // Моніторинг heap — попереджає про виснаження пам'яті
          if (ESP.getFreeHeap() < 20000) {
               Serial.printf("[WARN] Low heap: %u bytes free\n", ESP.getFreeHeap());
          }
          if (WiFi.status() == WL_CONNECTED && strlen(telegram_token) > 0 && strlen(telegram_chat_id) > 0) {

              if (autoReportIntervalMillis > 0 && (millis() - lastAutoReportTime >= autoReportIntervalMillis)) {
              lastAutoReportTime = millis();
               bot->sendMessage(telegram_chat_id, generateReport(), "Markdown");
              }

            // Забираємо знімок флагів під мютексом, потім відпускаємо — не тримаємо лок під час HTTP
            AlertFlags localAlerts;
            portENTER_CRITICAL(&dataMux);
              localAlerts = alerts;
              alerts = AlertFlags(); // скидаємо всі флаги одразу
            portEXIT_CRITICAL(&dataMux);

            if (localAlerts.needTelegramAlert) {
                 String alertMsg = "⚠️ **КРИТИЧНА ТРИВОГА!**\n📍 Приміщення: *" + String(room_name) + "*\n\nЗафіксовано високий рівень забруднення (TVOC)!\nПоточний рівень: *" + String(localAlerts.alertPollutionLevel) + "%*";
                  bot->sendMessage(telegram_chat_id, alertMsg, "Markdown");
            }
            if (localAlerts.needTelegramNormal) {
                String normalMsg = "✅ **Повітря очистилось**\n📍 Приміщення: *" + String(room_name) + "*\n\nРівень забруднення (TVOC) в нормі: *" + String(localAlerts.alertPollutionLevel) + "%*";
                  bot->sendMessage(telegram_chat_id, normalMsg, "Markdown");
            }

            if (localAlerts.needTempAlert) {
```

```
            String alertMsg = "🌡️ **⚠️ ТЕМПЕРАТУРНА ТРИВОГА!**\n📍
Приміщення: *" + String(room_name) + "*\n\nВихід за межі санітарних
норм ДСН (" + String(currentMinTemp, 0) + "-" +
String(currentMaxTemp, 0) + "°C).\nПоточна температура: *" +
String(localAlerts.alertTempLevel, 1) + " °C*";
                    bot->sendMessage(telegram_chat_id, alertMsg,
"Markdown");
            }
            if (localAlerts.needTempNormal) {
                    String normalMsg = "🌡️ **✅ Температура
нормалізувалась**\n📍 Приміщення: *" + String(room_name) +
"*\n\nТемпература повернулась в межі норми ДСН: *" +
String(localAlerts.alertTempLevel, 1) + " °C*";
                    bot->sendMessage(telegram_chat_id, normalMsg,
"Markdown");
            }

            if (localAlerts.needHumAlert) {
               String alertMsg = "💧 **⚠️ ТРИВОГА ВОЛОГОСТІ!**\n📍
Приміщення: *" + String(room_name) + "*\n\nВихід за межі санітарних
норм ДСН (" + String(currentMinHum, 0) + "-" + String(currentMaxHum,
0) + "%).\nПоточна вологість: *" + String(localAlerts.alertHumLevel,
0) + " %*";
                    bot->sendMessage(telegram_chat_id, alertMsg,
"Markdown");
            }
            if (localAlerts.needHumNormal) {
               String normalMsg = "💧 **✅ Вологість в нормі**\n📍
Приміщення: *" + String(room_name) + "*\n\nВологість
стабілізувалась: *" + String(localAlerts.alertHumLevel, 0) + " %*";
                    bot->sendMessage(telegram_chat_id, normalMsg,
"Markdown");
            }

            if (localAlerts.needLuxHighAlert) {
               String alertMsg = "🌞 **Надмірна освітленість**\n📍
Приміщення: *" + String(room_name) + "*\n\nОсвітленість: *" +
String(localAlerts.alertLuxLevel) + " lx*\nМожливо, забули вимкнути
світло?";
                    bot->sendMessage(telegram_chat_id, alertMsg,
"Markdown");
            }
            if (localAlerts.needLuxLowAlert) {
                String alertMsg = "🌑 **Низька освітленість**\n📍
Приміщення: *" + String(room_name) + "*\n\nОсвітленість: *" +
String(localAlerts.alertLuxLevel) + " lx*\nРекомендується увімкнути
світло для комфортної роботи.";
                    bot->sendMessage(telegram_chat_id, alertMsg,
"Markdown");
            }
```

```
                int    numNewMessages    =    bot->getUpdates(bot-
>last_message_received + 1);
          int safetyCounter = 0;
          while (numNewMessages && safetyCounter < 5) {
            handleNewMessages(numNewMessages);
                        numNewMessages    =    bot->getUpdates(bot-
>last_message_received + 1);
            safetyCounter++;
          }
        }
        vTaskDelay(5000 / portTICK_PERIOD_MS); // yield — дозволяє
IDLE-таскам годувати WDT
      }
    }

    void setup() {
      Serial.begin(115200);
      Wire.begin();

      pinMode(BUZZER_PIN, OUTPUT); digitalWrite(BUZZER_PIN, HIGH);
      pinMode(RELAY_PIN, OUTPUT); digitalWrite(RELAY_PIN, LOW);
       pinMode(BTN_PIN,  INPUT_PULLDOWN);  //  Запобігає  "плаванню"
піну
      // LEDC PWM для RGB LED — Core 3.x API
      ledcAttach(LED_R_PIN, 5000, 8); // пін, частота, розрядність
      ledcAttach(LED_G_PIN, 5000, 8);
      ledcAttach(LED_B_PIN, 5000, 8);

      // Ініціалізація OLED один раз — до перевірки кнопки
      if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("OLED init failed — reboot in 5s"));
        delay(5000);
        ESP.restart();
      }
      display.setTextColor(SSD1306_WHITE);

      // Утримання кнопки при старті — показуємо зворотний відлік
без delay()
      if (digitalRead(BTN_PIN) == HIGH) {
        unsigned long btnHoldStart = millis();
        bool resetConfirmed = false;
        while (digitalRead(BTN_PIN) == HIGH) {
          unsigned long held = millis() - btnHoldStart;
          if (held >= 3000) { resetConfirmed = true; break; }
          display.clearDisplay(); display.setTextSize(1);
              display.setCursor(5,  15);  display.print("Hold  to
RESET:");
          display.setTextSize(3); display.setCursor(52, 32);
          display.print((int)(3 - held / 1000) + 1);
          display.display();
          delay(50); // маленький yield, щоб watchdog не спрацював
        }
```

```
        if (resetConfirmed) {
          display.clearDisplay(); display.setTextSize(1);
               display.setCursor(10,  20);  display.print("WIPING
DATA..."); display.display();
          digitalWrite(BUZZER_PIN, LOW); setRGB(80, 0, 0);
          LittleFS.format();
          WiFi.disconnect(true, true);
          delay(1500);
          ESP.restart();
        }
      }

      if (!bme.begin(0x76)) {
        Serial.println(F("BME280 init failed — reboot in 5s"));
        display.clearDisplay(); display.setTextSize(1);
         display.setCursor(0, 20); display.print("BME280 ERROR!");
display.display();
        delay(5000);
        ESP.restart();
      }
      lightMeter.begin();

      display.clearDisplay(); display.setTextSize(1);
       display.setCursor(0,  0);  display.print("Mounting  FS...");
display.display();

      if (LittleFS.begin(true)) {
        if (LittleFS.exists("/config.json")) {
          File configFile = LittleFS.open("/config.json", "r");
          if (configFile) {
            StaticJsonDocument<512> json;
            if (!deserializeJson(json, configFile)) {
              strcpy(telegram_token, json["token"] | "");
              strcpy(telegram_chat_id, json["chat_id"] | "");
              strcpy(room_name, json["room_name"] | "Кімната 1");

              if (json.containsKey("season")) {
                isWarmSeason = json["season"].as<bool>();
              } else {
                isWarmSeason = true;
              }

              if (json.containsKey("auto_report")) {
                                  autoReportIntervalMillis    =
json["auto_report"].as<unsigned long>();
              } else {
                autoReportIntervalMillis = 0;
              }

              if (isWarmSeason) {
                currentMinTemp = 21.0; currentMaxTemp = 28.0;
              } else {
```

```
                currentMinTemp = 20.0; currentMaxTemp = 25.0;
              }
            }
            configFile.close(); // звільняємо файловий дескриптор
          }
        }
      }

          display.clearDisplay();     display.setCursor(0,     0);
display.print("Connecting WiFi...");
      display.setCursor(0, 20); display.print("Or connect to AP:");
       display.setCursor(0, 30); display.print("Climate_Monitor");
display.display();

       WiFiManagerParameter custom_token("token", "Telegram Bot
Token", telegram_token, 60);
       WiFiManagerParameter custom_chat_id("chat_id", "Telegram
Chat ID", telegram_chat_id, 30);
       WiFiManagerParameter custom_room("room", "Room Name",
room_name, 30);

      WiFiManager wm;
      wm.setSaveConfigCallback(saveConfigCallback);
                                  wm.addParameter(&custom_token);
wm.addParameter(&custom_chat_id); wm.addParameter(&custom_room);

      wm.setConnectTimeout(15);
      wm.setConfigPortalTimeout(60);

      if (!wm.autoConnect("Climate_Monitor", "Admin12345")) {
        isOfflineMode = true;
        display.clearDisplay(); display.setTextSize(1);
        display.setCursor(15, 20); display.print("WIFI FAILED!");
        display.setCursor(15, 35); display.print("OFFLINE MODE");
        display.display();
        delay(3000);
      } else {
        isOfflineMode = false;
      }

      WiFi.mode(WIFI_STA);
      WiFi.setSleep(false);

      if (shouldSaveConfig) {
        strcpy(telegram_token, custom_token.getValue());
        strcpy(telegram_chat_id, custom_chat_id.getValue());
        strcpy(room_name, custom_room.getValue());

        StaticJsonDocument<512> json;
           json["token"]   =   telegram_token;   json["chat_id"]   =
telegram_chat_id; json["room_name"] = room_name;
        json["season"] = isWarmSeason;
```

```
        json["auto_report"] = autoReportIntervalMillis;

        File configFile = LittleFS.open("/config.json", "w");
          if (configFile) { serializeJson(json, configFile);
configFile.close(); }

        display.clearDisplay();
        display.setCursor(10, 20);
        display.print("Saved! Restarting...");
        display.display();
        delay(2000);
        ESP.restart();
      }

      client_tg.setInsecure();
      // Статичне виділення — без витоку heap при перезапуску задачі
       static UniversalTelegramBot tgBotInstance(telegram_token,
client_tg);
      bot = &tgBotInstance;

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

             server.on("/data",        WebRequestMethod::HTTP_GET,
[](AsyncWebServerRequest *request){
        // Захищений знімок для веб-відповіді (читається з будь-
якого ядра)
        float t, h, pr, lx; int pol; bool season;
        portENTER_CRITICAL(&dataMux);
          t = sensorSnap.temp; h = sensorSnap.hum;
          pr = sensorSnap.pres; lx = sensorSnap.lux;
          pol = sensorSnap.pollutionPct; season = isWarmSeason;
        portEXIT_CRITICAL(&dataMux);

        char buf[160];
        snprintf(buf, sizeof(buf),
           "{\"room\":\"%s\",\"temp\":%.1f,\"hum\":%.0f,\"pres\":%
.1f,\"lux\":%d,\"tvoc\":%d,\"season\":%s}",
           room_name, t, h, pr, (int)lx, pol, season ? "true" :
"false");
        request->send(200, "application/json", buf);
      });
      server.begin();

      xTaskCreatePinnedToCore(telegramTaskCode, "Telegram", 20480,
NULL, 1, &TelegramTask, 0); // збільшено для TLS+String

      bootTime = millis();
      lastAutoReportTime = millis();
    }
```

```
     void loop() {
       unsigned long currentMillis = millis();
       bool forceDisplayUpdate = false;

       // --- ФОНОВЕ ВІДНОВЛЕННЯ WIFI ---
       if (WiFi.status() != WL_CONNECTED) {
             if    (!isOfflineMode)    {    isOfflineMode    =    true;
forceDisplayUpdate = true; }
               if    (currentMillis    -    lastWiFiCheckTime    >=
WIFI_CHECK_INTERVAL) {
          lastWiFiCheckTime = currentMillis;
          Serial.println(F("WiFi: reconnecting..."));
          WiFi.disconnect(false);
          WiFi.reconnect();
        }
       } else if (isOfflineMode) {
         isOfflineMode = false;
         forceDisplayUpdate = true;
         Serial.println(F("WiFi: restored"));
       }

       // --- БЕЗПЕЧНЕ ЗБЕРЕЖЕННЯ ---
       if (pendingConfigSave && currentMillis >= saveConfigTime) {
         pendingConfigSave = false;
         StaticJsonDocument<512> json;
         json["token"] = telegram_token;
         json["chat_id"] = telegram_chat_id;
         json["room_name"] = room_name;
         json["season"] = isWarmSeason;
         json["auto_report"] = autoReportIntervalMillis;
         File configFile = LittleFS.open("/config.json", "w");
           if  (configFile)  {  serializeJson(json,  configFile);
configFile.close(); }
       }

       int reading = digitalRead(BTN_PIN);
       static unsigned long buttonPressedTime = 0;
       static bool isPressing = false;
       static bool warningShown = false;

       if (reading == HIGH) {
         if (!isPressing) { isPressing = true; buttonPressedTime =
currentMillis; warningShown = false; }
        unsigned long holdTime = currentMillis - buttonPressedTime;
         if (holdTime >= 1000) {
           int timeLeft = 4 - (holdTime / 1000);
           if (timeLeft > 0) {
                 display.clearDisplay();  display.setTextSize(1);
display.setCursor(10, 10); display.print("HOLD TO RESET:");
               display.setTextSize(3);  display.setCursor(55,  30);
display.print(timeLeft); display.display();
```

```
            warningShown = true;
          } else {
                display.clearDisplay();  display.setTextSize(1);
display.setCursor(10,   20);   display.print("WIPING   DATA...");
display.display();
            digitalWrite(BUZZER_PIN, LOW); setRGB(80, 0, 0);
                LittleFS.format();  WiFi.disconnect(true,  true);
delay(2000); ESP.restart();
          }
        }
      }
      else {
        if (isPressing) {
          isPressing = false;
               unsigned   long   holdTime   =   currentMillis   -
buttonPressedTime;
          if (warningShown) {
                display.clearDisplay();  display.setTextSize(2);
display.setCursor(15,       25);       display.print("CANCELED!");
display.display();
            delay(1000); warningShown = false; forceDisplayUpdate
= true;
          } else if (holdTime > 50 && holdTime < 1000) {
              screenMode++; if (screenMode > 3) screenMode = 0;
forceDisplayUpdate = true;
          }
        }
      }

      if (currentMillis - lastSensorUpdate >= 3000) {
        lastSensorUpdate = currentMillis;

        // Читаємо датчики поза мютексом (I2C/ADC — тільки Core 1)
        float newTemp = bme.readTemperature();
        float newHum  = bme.readHumidity();
        float newPres = bme.readPressure() * 0.00750062f;
        float newLux  = lightMeter.readLightLevel();
        int gasRaw = analogRead(GAS_PIN);
        int currentGasPct = map(gasRaw, 0, 4095, 0, 100);

        // Захист від помилок датчиків: NaN (BME280 відключений)
або -1 (BH1750 помилка)
        if (isnan(newTemp) || isnan(newHum) || isnan(newPres)) {
          Serial.println(F("[WARN] BME280 read error — skipping
cycle"));
          return; // пропускаємо цикл, не псуємо статистику
        }
        if (newLux < 0) {
          newLux = lux; // залишаємо попереднє значення
          Serial.println(F("[WARN] BH1750 read error — using last
value"));
        }
```

```
          // Захист глобальних змінних і знімку — мінімальний
критичний регіон
         portENTER_CRITICAL(&dataMux);
            temp = newTemp; hum = newHum; pres = newPres; lux =
newLux;
          pollutionPct = (currentGasPct > pollutionPct)
                        ? currentGasPct
                        : (pollutionPct * 3 + currentGasPct) / 4;

          sensorSnap.temp = temp; sensorSnap.hum = hum;
          sensorSnap.pres = pres; sensorSnap.lux = lux;
          sensorSnap.pollutionPct = pollutionPct;

          // Оновлення статистики (рекорди)
          if (temp < minT) minT = temp;
          if (temp > maxT) maxT = temp;
          if (hum  < minH) minH = hum;
          if (hum  > maxH) maxH = hum;
          if (pres < minP) minP = pres;
          if (pres > maxP) maxP = pres;
          if ((int)lux < minLux) minLux = (int)lux;
          if ((int)lux > maxLux) maxLux = (int)lux;
          if (pollutionPct > maxTVOC) maxTVOC = pollutionPct;

          // АНАЛІЗ ЯКОСТІ ПОВІТРЯ (ГАЗ / TVOC)
          if (pollutionPct > 35) {
                  if   (!isGasAlarmState   ||   (currentMillis   -
lastTelegramAlertTime >= TELEGRAM_COOLDOWN)) {
              alerts.alertPollutionLevel = pollutionPct;
              isGasAlarmState = true;
              lastTelegramAlertTime = currentMillis;
              gasAlarmStartTime = currentMillis;
              alerts.needTelegramAlert = true;
            }
          } else if (pollutionPct < 30) {
            if (isGasAlarmState) {
              alerts.alertPollutionLevel = pollutionPct;
              alerts.needTelegramNormal = true;
            }
            isGasAlarmState = false;
              alarmLoggedFlags &= ~FLAG_GAS; // тривога знята —
наступне спрацювання знову запишеться одразу
          }

          // ТЕМПЕРАТУРА
          if (temp > currentMaxTemp || temp < currentMinTemp) {
            if (!isTempAlarmState) {
              alerts.alertTempLevel = temp;
              isTempAlarmState = true;
              tempAlarmStartTime = currentMillis;
              alerts.needTempAlert = true;
```

```
            }
          } else if (temp >= currentMinTemp + 0.5f && temp <=
currentMaxTemp - 0.5f) {
            if (isTempAlarmState) {
              alerts.alertTempLevel = temp;
              alerts.needTempNormal = true;
            }
            isTempAlarmState = false;
            alarmLoggedFlags &= ~FLAG_TEMP;
          }

          // ВОЛОГІСТЬ
          if (hum > currentMaxHum || hum < currentMinHum) {
            if (!isHumAlarmState) {
              alerts.alertHumLevel = hum;
              isHumAlarmState = true;
              humAlarmStartTime = currentMillis;
              alerts.needHumAlert = true;
            }
            } else if (hum >= currentMinHum + 2.0f && hum <=
currentMaxHum - 2.0f) {
            if (isHumAlarmState) {
              alerts.alertHumLevel = hum;
              alerts.needHumNormal = true;
            }
            isHumAlarmState = false;
            alarmLoggedFlags &= ~FLAG_HUM;
          }

          // СВІТЛО
          if (lux > 800) {
            if (!isLuxHighAlarmState) {
              alerts.alertLuxLevel = (int)lux;
              alerts.needLuxHighAlert = true;
              isLuxHighAlarmState = true;
            }
            isLuxLowAlarmState = false;
          } else if (lux < 100) {
            if (!isLuxLowAlarmState) {
              alerts.alertLuxLevel = (int)lux;
              alerts.needLuxLowAlert = true;
              isLuxLowAlarmState = true;
            }
            isLuxHighAlarmState = false;
          } else if (lux >= 150 && lux <= 700) {
            isLuxHighAlarmState = false;
            isLuxLowAlarmState = false;
          }
        portEXIT_CRITICAL(&dataMux);

         // ThingSpeak — тривожний запис: 1 раз при появі НОВОЇ
тривоги,
```

```
          // далі не частіше ніж раз на 15 хв (але лише якщо є активна
тривога)
          {
            uint8_t activeFlags = 0;
            if (isGasAlarmState)  activeFlags |= FLAG_GAS;
            if (isTempAlarmState) activeFlags |= FLAG_TEMP;
            if (isHumAlarmState)  activeFlags |= FLAG_HUM;

            // нові тривоги = з'явились зараз, ще не зафіксовані
            uint8_t newFlags = activeFlags & ~alarmLoggedFlags;
            // скидаємо флаги для тривог що вже пройшли
            alarmLoggedFlags &= activeFlags;

            bool shouldLog = false;
            if (newFlags) {
              // нова тривога — фіксуємо одразу
              alarmLoggedFlags |= newFlags;
              shouldLog = true;
                 } else if (activeFlags && (currentMillis -
lastAlarmLogTime >= LOG_INTERVAL)) {
              // стара тривога досі активна — повторно раз на 15 хв
              shouldLog = true;
            }

            if (shouldLog) {
              lastAlarmLogTime = currentMillis;
              logToThingSpeak();
            }
          }

           if (!warningShown && !isPressing) forceDisplayUpdate =
true;
        }

        if (currentMillis - lastLogTime >= LOG_INTERVAL) {
          lastLogTime = currentMillis;
          logToThingSpeak();
        }

        if (forceDisplayUpdate && !warningShown) {
          display.clearDisplay();

          if (screenMode == 0) {
                 display.setTextSize(1);  display.setCursor(0,  0);
display.print("IP: "); display.print(WiFi.localIP());
                display.setTextSize(2);  display.setCursor(0,  20);
display.print(F("T:"));          display.print(temp,          1);
display.print(F("C"));
                display.setCursor(0,  45);  display.print(F("H:"));
display.print(hum, 0); display.print(F(" %"));
          }
          else if (screenMode == 1) {
```

```
              display.setTextSize(1);  display.setCursor(30,  0);
display.print(F("ATMOSPHERE"));
           display.setCursor(0, 15); display.print(F("Pressure:"));
              display.setTextSize(2);  display.setCursor(0,  27);
display.print(pres, 1); display.print(F(" mmHg"));
              display.setTextSize(1);  display.setCursor(0,  50);
display.print(F("Lux: "));
               display.setTextSize(2);   display.print((int)lux);
display.print(F(" lx"));
         }
         else if (screenMode == 2) {
              display.setTextSize(1);  display.setCursor(30,  0);
display.print(F("AIR QUALITY"));
               display.setCursor(0,  20);  display.print(F("TVOC
Level:"));
             display.setTextSize(3);  display.setCursor(20,  35);
display.print(pollutionPct); display.print(F("%"));
            if  (pollutionPct  >  35)  {  display.setTextSize(1);
display.setCursor(0, 56); display.print(F("!!! VENTILATE !!!")); }
         }
         else if (screenMode == 3) {
              display.setTextSize(1);  display.setCursor(20,  0);
display.print(F("LED INDICATORS"));
          display.setCursor(0, 15); display.print(F("GRN: OK   BLU:
Norm"));
            display.setCursor(0,  27);  display.print(F("YEL:  Temp
Alert"));
             display.setCursor(0,  39);  display.print(F("PUR:  Hum
Alert"));
             display.setCursor(0,  51);  display.print(F("RED:  Gas
Alarm!"));
         }

         display.display();
       }

       if (pollutionPct > 35) {
         if (currentMillis - gasAlarmStartTime < 7000) {
           digitalWrite(BUZZER_PIN, ((currentMillis / 500) % 2 == 0)
? LOW : HIGH);
            if ((currentMillis / 500) % 2 == 0) setRGB(80, 0, 0);
else setRGB(0, 0, 0);
         } else {
           digitalWrite(BUZZER_PIN, HIGH);
           setRGB(80, 0, 0);
         }
       }
       else if (isTempAlarmState) {
         digitalWrite(BUZZER_PIN, HIGH);
         if (currentMillis - tempAlarmStartTime < 10000) {
           if ((currentMillis / 500) % 2 == 0) setRGB(80, 40, 0);
else setRGB(0, 0, 0);
```

```
        } else {
          setRGB(80, 40, 0);
        }
      }
      else if (isHumAlarmState) {
        digitalWrite(BUZZER_PIN, HIGH);
        if (currentMillis - humAlarmStartTime < 5000) {
          if ((currentMillis / 500) % 2 == 0) setRGB(80, 0, 80);
else setRGB(0, 0, 0);
        } else {
          setRGB(80, 0, 80);
        }
      }
      else if (pollutionPct < 20 && temp >= currentMinTemp + 1.0 &&
temp <= currentMaxTemp - 1.0 && hum >= currentMinHum + 5 && hum <=
currentMaxHum - 5) {
        digitalWrite(BUZZER_PIN, HIGH); setRGB(0, 80, 0);
      }
      else {
        digitalWrite(BUZZER_PIN, HIGH); setRGB(0, 0, 80);
      }
    }
```