МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри автоматизації та
комп'ютерно-інтегрованих технологій
_____ Микола СІДЄЛЄВ
«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

АВТОМАТИЗОВАНА СИСТЕМА МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ НА ОСНОВІ IoT МЕРЕЖІ

Спеціальність 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
Освітня програма «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»

Здобувач _____ В'ячеслав МОЗГОВИЙ
«___» _____ 2026 р.

Керівник роботи
к.т.н, доцент _____ Володимир САВІНОВ
«___» _____ 2026 р.

Консультант _____ Людмила ГРИГОР'ЄВА
докт. біол. наук, професор «___» _____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Автоматизації та комп’ютерно-інтегрованих технологій
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	174 Автоматизація, комп’ютерно-інтегровані технології та робототехніка
Освітня програма	Автоматизація, комп’ютерно-інтегровані технології та робототехніка

ЗАТВЕРДЖУЮ
Завідувач кафедри автоматизації та
комп’ютерно-інтегрованих технологій
_____ Микола СІДЄЛЄВ
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну магістерську роботу здобувача
Мозговий В’ячеслав Сергійович

1. Тема кваліфікаційної роботи «Автоматизована система моніторингу якості повітря на основі IoT мережі» затверджена наказом в.о. ректора ЧНУ ім. Петра Могили № 265 від «05» листопада 2025 р..

2. Строк представлення кваліфікаційної роботи «18» червня 2026 р.

3. Очікуваний результат роботи та початкові дані: розроблена й верифікована автоматизована розподілена система моніторингу якості повітря; апаратна база — мікроконтролер ESP32-WROOM-32E, сенсор Sensirion SEN55, NDIR-сенсор CO_2 Winsen MH-Z19B; протокол MQTT 5.0, брокер EMQX 5.x; серверна платформа Next.js 14 / MongoDB 7; клієнтський інтерфейс React 18 / Webpack 5.

4. Перелік питань, що підлягають розробці:

1) аналіз предметної галузі та вибір апаратної бази: порівняльний аналіз датчиків твердих частинок (PM), CO_2, летких органічних сполук та мікроконтролерних платформ;

2) проєктування трирівневої архітектури IoT-мережі (Edge–Fog–Cloud) та обґрунтування протоколу передачі даних MQTT 5.0;

3) тривимірне моделювання конструкції вимірювального вузла та симуляція аеродинамічних потоків у середовищі SolidWorks;

4) розробка прошивки вимірювального вузла на базі FreeRTOS з алгоритмом адаптивної частоти опитування, серверної частини (Next.js 14 + MongoDB 7) та клієнтського інтерфейсу (React 18) з обчисленням індексу AQI;

5) натурний 14-добовий експеримент: метрологічна верифікація показань, аналіз надійності та продуктивності системи;

6) математичне моделювання об’єкта керування та синтез системи автоматичного керування якістю повітря (ідентифікація, ПІД/релейний регулятор, прогнозування);

7) техніко-економічне обґрунтування проєкту та опрацювання питань охорони праці.

5. Перелік графічних матеріалів: структурна схема IoT-мережі; функціональна схема вимірювального вузла; діаграма задач FreeRTOS; блок-схема алгоритму адаптивного опитування; графіки часових рядів вимірювань; теплова карта кореляцій.

6. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Савінов В.Ю.	Автоматизація та КІТ	1 – 3 розділи
Григор’єва Л.І.	Екологія	4 розділ

Дата видачі завдання «05» листопада 2025 р.

Здобувач _____ В’ячеслав МОЗГОВИЙ

Керівник роботи _____ Володимир САВІНОВ

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: «Автоматизована система моніторингу якості повітря на основі IoT мережі».

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КМР	09.09.2025	05.11.2025	Виконано
2	Огляд літератури за темою роботи	06.11.2025	20.11.2025	Виконано
3	Складання календарного плану КМР	21.11.2025	22.11.2025	Виконано
4	Аналіз предметної галузі та вибір апаратної бази	24.11.2025	22.12.2025	Виконано
5	Проєктування архітектури IoT-мережі та протоколів	23.12.2025	26.02.2026	Виконано
6	Тривимірне моделювання, розробка прошивки та ПЗ	02.03.2026	17.04.2026	Виконано
7	Натурний експеримент і дослідження системи, аналіз результатів	20.04.2026	11.05.2026	Виконано
8	Попередній захист	12.05.2026	12.05.2026	Виконано
9	Робота над розділом з охорони праці	17.04.2026	20.05.2026	Виконано
10	Відгук керівника КМР	21.05.2026	22.05.2026	Виконано
11	Оформлення КМР та презентації	25.05.2026	04.06.2026	Виконано
12	Рецензування	05.06.2026	16.06.2026	Виконано
13	Завершення оформлення КМР та презентації	17.06.2026	18.06.2026	Виконано
14	Захист кваліфікаційної роботи	25.06.2026	25.06.2026	Виконано

Здобувач _____ В’ячеслав МОЗГОВИЙ

Керівник роботи _____ Володимир САВІНОВ

АНОТАЦІЯ

Мозговий В’ячеслав. «Автоматизована система моніторингу якості повітря на основі IoT мережі».

1. Актуальність роботи зумовлена недостатньою щільністю державних постів спостереження за якістю атмосферного повітря в Україні (менше одного поста на 1000 км²) за документально підтверджених шкідливих концентрацій PM2.5, а також доступністю сучасних MEMC-сенсорів, що уможливлює побудову щільних вимірювальних мереж на базі низькобюджетних IoT-вузлів.

2. Об’єкт дослідження — процес автоматизованого збору, передачі та аналізу даних про якість атмосферного повітря в умовах розподіленої мережі IoT-вузлів на базі мікроконтролерів ESP32.

3. Предмет дослідження — методи та засоби побудови IoT-мережі на базі ESP32-WROOM-32E для вимірювання PM2.5, PM10, CO_2, NOx, VOC, температури та відносної вологості з передачею через протокол MQTT 5.0.

4. Мета і завдання роботи. Мета — проєктування, реалізація та верифікація автоматизованої розподіленої системи моніторингу та керування якістю повітря з хмарною обробкою даних і веб-інтерфейсом реального часу. Завдання: обрати апаратну конфігурацію; спроєктувати трирівневу архітектуру та обґрунтувати MQTT 5.0; розробити прошивку на FreeRTOS з адаптивним опитуванням; реалізувати серверну та клієнтську частини; виконати натурний експеримент; розробити математичну модель об'єкта та замкнений контур автоматичного керування вентиляцією; провести техніко-економічне обґрунтування та аналіз охорони праці.

5. Методи дослідження: системний аналіз і синтез, порівняльний аналіз технічних характеристик компонентів, методи математичної статистики (середнє, СКВ, коефіцієнт кореляції Пірсона), метод натурного експерименту та методи калібрування вимірювальних засобів.

6. Основні результати. Розроблено трирівневу архітектуру Edge–Fog–Cloud; прошивку ESP32 на FreeRTOS з алгоритмом адаптивної частоти

опитування (економія 69 % споживання) та on-edge фільтрацією; серверну платформу Next.js 14 + MongoDB 7 з REST API і WebSocket; React-інтерфейс із тепловою картою AQI. Натурний 14-добовий експеримент підтвердив точність PM2.5 ±0.2 % (після калібрування) і надійність доставки 98.93 %. Наукова новизна — алгоритм адаптивного керування частотою опитування за шкалою AQI EPA та on-edge фільтрація аномалій. Додатково розроблено замкнений контур автоматичного керування вентиляцією: математичну модель об'єкта (динаміки CO_2), ідентифікацію її параметрів ($T \approx 27$ хв), синтез ПІД- та релейного регуляторів, оптимізацію режиму та прогнозування AQI.

7. Обсяг роботи: 111 с., 29 табл., 31 рис., 3 додатки, 20 джерел посилання.

Ключові слова: *моніторинг якості повітря, Інтернет речей, ESP32, MQTT 5.0, FreeRTOS, індекс якості повітря, метрологічна верифікація, адаптивне опитування.*

ABSTRACT

Mozgovyi Viacheslav. “Automated air quality monitoring system based on an IoT network”.

1. Relevance. The work is motivated by the low density of state air-quality monitoring posts in Ukraine (fewer than one per 1000 km²) under documented harmful PM2.5 concentrations, and by the availability of modern MEMS sensors enabling dense networks of low-cost IoT nodes.

2. Object of research — the process of automated collection, transmission and analysis of atmospheric air quality data in a distributed IoT sensor-node network based on ESP32 microcontrollers.

3. Subject of research — methods and tools for building an IoT network on ESP32-WROOM-32E for measuring PM2.5, PM10, CO_2, NOx, VOC, temperature and relative humidity transmitted via the MQTT 5.0 protocol.

4. Purpose and tasks. Purpose — design, implementation and verification of an automated distributed air-quality monitoring system with cloud data processing and a real-time web dashboard. Tasks: select the hardware configuration; design the three-tier architecture and justify MQTT 5.0; develop FreeRTOS firmware with adaptive polling; implement the server and client parts; carry out a field experiment; perform a techno-economic and occupational-safety analysis.

5. Methods: systems analysis and synthesis, comparative analysis of component specifications, mathematical statistics (mean, RMS, Pearson correlation), field experiment and sensor calibration methods.

6. Main results. A three-tier Edge–Fog–Cloud architecture was developed; ESP32 firmware on FreeRTOS with an adaptive polling-frequency algorithm (69 % energy savings) and on-edge filtering; a Next.js 14 + MongoDB 7 server platform with REST API and WebSocket; a React 18 interface with an AQI heat map. A 14-day field experiment confirmed PM2.5 accuracy ±0.2 % (after calibration) and data-delivery reliability of 98.93 %. The scientific novelty is an adaptive polling-frequency algorithm by EPA AQI breakpoints and on-edge anomaly filtering. In addition, a closed-loop automatic ventilation control system

was developed: a controlled-object model (CO_2 dynamics), parameter identification ($T \approx 27$ min), PID and relay controller synthesis, regime optimisation and AQI forecasting.

7. Volume: 111 p., 29 tab., 31 fig., 3 appendices, 20 references.

Keywords: *air quality monitoring, Internet of Things, ESP32, MQTT 5.0, FreeRTOS, air quality index, metrological verification, adaptive polling.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ABC	Automatic Baseline Calibration — автоматичне базове калібрування CO_2-сенсора
ACL	Access Control List — список контролю доступу
ADC	Analog-to-Digital Converter — аналогово-цифровий перетворювач
AQI	Air Quality Index — індекс якості повітря
BLE	Bluetooth Low Energy — Bluetooth із низьким енергоспоживанням
EPA	Environmental Protection Agency — Агентство охорони навкол. середовища США
ESP32	мікроконтролер Espressif Systems ESP32-WROOM-32E
FIFO	First In, First Out — черга «першим прийшов — першим вийшов»
FreeRTOS	Free Real-Time Operating System — RTOS для вбудованих систем
ГДК	граничнодопустима концентрація забруднювача
GPIO	General-Purpose Input/Output — порт загального призначення
HMAC	Hash-based Message Authentication Code — код автентифікації повідомлень
HTTP	HyperText Transfer Protocol — протокол передачі гіпертексту
I2C	Inter-Integrated Circuit — двопровідний послідовний інтерфейс
IDW	Inverse Distance Weighting — інтерполяція за зворотньою відстанню
IoT	Internet of Things — Інтернет речей
JSON	JavaScript Object Notation — формат обміну даними
JWT	JSON Web Token — маркер аутентифікації
LWT	Last Will and Testament — механізм MQTT для сигналізації відключення
MQTT	Message Queuing Telemetry Transport — протокол IoT-передачі даних
NDIR	Non-Dispersive Infrared — недисперсійний інфрачервоний метод
NTP	Network Time Protocol — протокол мережевої синхронізації часу
NVS	Non-Volatile Storage — енергонезалежне сховище параметрів ESP32
OTA	Over-The-Air — бездротове оновлення прошивки

PM2.5	Particulate Matter ≤2.5 мкм — аерозольні частинки діаметром до 2.5 мкм
PM10	Particulate Matter ≤10 мкм — аерозольні частинки діаметром до 10 мкм
QoS	Quality of Service — рівень якості обслуговування MQTT
REST	Representational State Transfer — архітектурний стиль веб-API
RSSI	Received Signal Strength Indicator — рівень прийнятого сигналу Wi-Fi
SNTP	Simple Network Time Protocol — спрощений протокол синхронізації часу
SPA	Single Page Application — односторінковий веб-застосунок
SPIFFS	SPI Flash File System — файлова система SPI-пам'яті ESP32
TCO	Total Cost of Ownership — сукупна вартість володіння
TLS	Transport Layer Security — протокол захисту транспортного рівня
UART	Universal Asynchronous Receiver/Transmitter — послідовний інтерфейс
VPS	Virtual Private Server — віртуальний приватний сервер
VOC	Volatile Organic Compounds — леткі органічні сполуки
ВООЗ	Всесвітня організація охорони здоров'я (WHO)

ВСТУП

За оцінками Всесвітньої організації охорони здоров'я (WHO, 2021), 92 % населення планети мешкає в регіонах, де середньорічна концентрація PM2.5 перевищує рекомендований рівень 5 мкг/м³, а сумарні втрати від забруднення повітря становлять близько 7 мільйонів передчасних смертей щорічно. Частинки PM2.5 (аеродинамічний діаметр ≤2.5 мкм) є особливо небезпечними: вони вільно долають мукоциліарний бар'єр, досягають альвеол та трансплацентарно потрапляють у кровоносну систему, спричиняючи системне запалення, атеросклероз, аритмії та онкологічні патології. Дослідження Pope et al. (2002), що охопило 1.2 млн американців, показало: збільшення середньорічної концентрації PM2.5 на 10 мкг/м³ пов'язане з підвищенням смертності від серцево-судинних захворювань на 12% та від раку легень — на 14%.

Особливу гостроту проблема забруднення повітря має для урбанізованих територій, де поєднуються інтенсивний автомобільний трафік, опалювальні викиди та промислові джерела. За таких умов навіть короткочасні епізоди перевищення гранично допустимих концентрацій PM2.5 створюють кумулятивний ризик для здоров’я населення, а отже потребують не епізодичних лабораторних замірів, а безперервного автоматизованого спостереження з високою часовою та просторовою роздільною здатністю.

Попри критичну значущість проблеми, в Україні функціонує менше 60 стаціонарних постів Державної екологічної інспекції — менше одного поста на 1000 км². Для порівняння: у Лондоні на площі 1572 км² діє понад 180 моніторингових точок (0.11/км²). Ключова причина диспропорції — вартість референсного обладнання: аналізатор PM методом бета-атенюації (Thermo Scientific 5030 SHARP) коштує 35 000–70 000 євро без урахування щорічного сервісного обслуговування (~10–15% вартості).

Технологічна революція в галузі МЕМС-сенсорів відкрила принципово новий шлях: побудова щільних вимірювальних мереж із низькобюджетних IoT-вузлів. Sensirion SEN55 — мультипараметричний датчик PM1/2.5/4/10 + VOC + NOx + T + RH в одному корпусі з I2C-інтерфейсом — коштує ~50 євро і демонструє похибку PM2.5 ±5% після фабричного калібрування. У поєднанні з мікроконтролером ESP32-WROOM-32E ($4) та NDIR-сенсором CO_2 MH-Z19B ($15) загальна вартість вузла складає ~105–125 євро — у 300–500 разів менше референсного поста. Незалежне дослідження Petrica et al. (2026) підтвердило валідність такого підходу: похибка PM2.5 при верифікації проти румунської національної мережі склала менше 5% [2].

В академічній літературі представлено численні IoT-системи моніторингу повітря, проте вони мають характерні обмеження. Harish et al. (2021) використовують аналоговий сенсор MQ135, що дає єдиний сигнал без розрізнення компонентів [3]. Dineshkumar et al. (2021) застосовують Arduino з ESP8266 — два окремих мікроконтролери без FreeRTOS [4]. Ramadan et al. (2024) розгортають систему з 9 модулями та LSTM-прогнозуванням, але на Arduino Mega з NB-IoT-модемом [1]. Жодна з цих робіт не реалізує алгоритму адаптивного керування частотою вимірювань.

Актуальність роботи обумовлена: (1) відсутністю щільних мереж моніторингу у вітчизняних містах за документально підтверджених шкідливих концентрацій PM2.5; (2) зрілістю МЕМС-сенсорів і платформи ESP32 для промислового тиражування; (3) відсутністю верифікованої відкритої методики розгортання подібних систем із задокументованими характеристиками точності.

Таким чином, науково-технічне завдання полягає у створенні системи, яка поєднує низьку вартість тиражування, прийнятну для індикативного моніторингу точність та відкритість програмно-апаратної платформи. Розв'язання цього завдання потребує комплексного опрацювання всіх рівнів — від вибору сенсорів і мікроконтролера до протоколів передачі, хмарної

обробки та візуалізації даних, — що й визначає структуру подальших розділів роботи.

Зв'язок роботи з науковими програмами. Робота виконана на кафедрі автоматизації та комп’ютерно-інтегрованих технологій Чорноморського національного університету імені Петра Могили відповідно до напряму наукових досліджень «Автоматизація та комп'ютерно-інтегровані технології» та відповідає положенням Стратегії екологічної безпеки України і директивам ЄС 2008/50/ЕС щодо якості атмосферного повітря.

Метою роботи є проектування, реалізація та верифікація автоматизованої розподіленої IoT-системи безперервного моніторингу якості атмосферного повітря на основі мікроконтролерів ESP32-WROOM-32E з передачею через MQTT 5.0 до платформи Next.js 14 + MongoDB 7 та відображенням на React-дашборді в реальному часі.

Для досягнення поставленої мети вирішено такі завдання:

1) провести порівняльний аналіз існуючих систем моніторингу, датчиків якості повітря та мікроконтролерних платформ і обрати оптимальну апаратну конфігурацію;
2) спроектувати трирівневу архітектуру IoT-мережі, обґрунтувати вибір MQTT 5.0 та топологію Wi-Fi STA + ESP-NOW;
3) розробити прошивку вимірювального вузла на FreeRTOS з алгоритмом адаптивної частоти опитування та on-edge медіанною фільтрацією;
4) реалізувати серверну платформу Next.js 14 + MongoDB 7 з REST API і WebSocket та клієнтський React 18 інтерфейс;
5) провести 14-добовий натурний експеримент, виконати метрологічну верифікацію та аналіз надійності передачі даних;
6) виконати техніко-економічне обґрунтування та аналіз умов охорони праці.

Об'єктом дослідження є процес автоматизованого збору, передачі та аналізу даних про якість атмосферного повітря в умовах розподіленої мережі IoT-вузлів.

Предметом дослідження є методи та засоби побудови IoT-мережі на базі ESP32-WROOM-32E для вимірювання PM2.5, PM10, CO_2, NOx, VOC, температури і відносної вологості з передачею через MQTT 5.0 до хмарної платформи Next.js/MongoDB.

Методи дослідження: системний аналіз і синтез при проектуванні архітектури; методи математичної статистики (середнє, СКВ, кореляція Пірсона) для оцінки точності вимірювань; порівняльний аналіз IoT-протоколів; метод натурного експерименту; методи калібрування за еталонним приладом.

Наукова новизна одержаних результатів. Вперше для вимірювального вузла на ESP32: (1) розроблено алгоритм адаптивного керування частотою опитування сенсорів залежно від поточного значення PM2.5 за межами шкали AQI EPA, що забезпечує скорочення споживання на 69.0 % при збереженні повноти критичних подій 100 %; (2) запропоновано механізм on-edge фільтрації аномалій медіанним фільтром ковзного вікна W=5 до публікації MQTT-повідомлення, що скорочує мережевий трафік і навантаження на MongoDB.

Практичне значення одержаних результатів. Верифікований прототип системи (~€134 за вузол) придатний до тиражування; методика розгортання та калібрування підтверджена 14-добовим натурним експериментом; повний вихідний код прошивки та серверного ПЗ є відкритим та задокументованим.

Апробація результатів дослідження. Основні положення та результати роботи доповідались на засіданні кафедри автоматизації та комп'ютерно-інтегрованих технологій ЧНУ ім. Петра Могили (протокол від «___» _____ 2026 р. № ___).

Структура та обсяг роботи. Кваліфікаційна магістерська робота складається зі вступу, восьми розділів, загальних висновків, списку із 20 використаних джерел та трьох додатків. Загальний обсяг роботи — 102

сторінки, з них основного тексту — 77 сторінок. У роботі наведено 22 таблиці та 14 рисунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИБІР АПАРАТНОЇ БАЗИ

1.1 Атмосферне повітря як об'єкт автоматизованого контролю

Атмосферне повітря як об’єкт автоматизованого контролю має низку особливостей, що відрізняють його від типових технологічних середовищ. По-перше, контрольовані величини є просторово розподіленими й нестаціонарними: концентрації забруднювачів змінюються на масштабах десятків метрів і хвилин залежно від джерел емісії, рельєфу та метеоумов. По-друге, вимірювані діапазони охоплюють кілька порядків величини, що висуває жорсткі вимоги до динамічного діапазону та лінійності сенсорів. Саме ці властивості обумовлюють доцільність побудови не одиничного вимірювального приладу, а розподіленої мережі автономних вузлів із централізованою обробкою.

Якість атмосферного повітря визначається концентрацією забруднювальних речовин, що потрапляють до атмосфери внаслідок антропогенної діяльності (спалювання викопного палива, транспортні викиди, промислові процеси) або природних явищ (вулканізм, пилові бурі, лісові пожежі). Для цілей автоматизованого моніторингу ключовими є параметри, що піддаються вимірюванню з прийнятною точністю і швидкістю відгуку оптичними або електрохімічними методами: PM2.5, PM10, CO_2, NO_2, O_3, VOC.

Тверді частинки (Particulate Matter, PM) класифікуються за аеродинамічним діаметром — максимальним діаметром сфери тієї ж щільності, що рухається аналогічно даній частинці в потоці повітря. PM2.5 (≤2.5 мкм) є найнебезпечнішою фракцією: розмір, менший за діаметр еритроцита (7–8 мкм), дозволяє частинці досягти альвеол легень і транспортуватися в кровоносне русло. Хімічний склад PM2.5 надзвичайно різноманітний: сажа, сульфати, нітрати, важкі метали (Pb, Cd, As), поліциклічні ароматичні вуглеводні — кожен з яких має власний токсикологічний профіль. PM10 (≤10 мкм) затримується переважно у верхніх

дихальних шляхах, але навіть це спричиняє запальні реакції слизових оболонок.

Діоксид вуглецю CO_2 у типових атмосферних концентраціях (410–420 ppm у 2024 р.) не є токсичним, однак слугує ключовим індикатором якості повітря в закритих приміщеннях та ефективності вентиляції. Концентрація вище 1000 ppm асоційована зі зниженням когнітивних функцій на 15% (Satish et al., 2012), а перевищення 2500 ppm спричиняє помітне погіршення концентрації уваги та скорочення реакційного часу на 30–40%. У вуличному повітрі CO_2 також є індикатором інтенсивності автомобільного руху та дозволяє оцінити витрати вуглецевого сліду на локальному рівні.

Леткі органічні сполуки (VOC) є хімічно різнорідним класом речовин, що включають бензол, толуол, ксилол, формальдегід та сотні інших сполук. Їхнє спільне вимірювання через «індекс VOC» (алгоритм Sensirion, що перетворює провідність металоксидного шару в нормований індекс 0–500) дозволяє виявляти зони підвищеної концентрації промислових відходів або побутових джерел (фарби, розчинники, будівельні матеріали) без прецизійного ідентифікування конкретних сполук — для задач міського моніторингу це є прийнятним підходом.

Таблиця 1.1 ГДК основних атмосферних забруднювачів (ЄС/Україна/ВООЗ 2021)

Забруднювач	**Символ**	**ГДК добова (мкг/м³)**	**ГДК річна (мкг/м³)**	**Норма ВООЗ 2021 (мкг/м³)**	**Директива/Стандарт**
Тверді частинки <2.5 мкм	PM2.5	25	10	15 / 5*	2008/50/EC; WHO 2021
Тверді частинки <10 мкм	PM10	50	40	45 / 15*	2008/50/EC; WHO 2021
Діоксид азоту	NO_2	200 (1-год)	40	25	2008/50/EC
Озон	O_3	120 (8-год)	—	100 (8-год)	2008/50/EC
Монооксид вуглецю	CO	10 000 (8-год)	—	4 000 (24-год)	2008/50/EC
Діоксид вуглецю	CO_2	—	—	400 ppm (зовн.)	ASHRAE 62.1
Леткі орг. сполуки	VOC	—	—	260 (WHO 2010)	WHO 2010

Забруднювач	Символ	ГДК добова (мкг/м³)	ГДК річна (мкг/м³)	Норма ВООЗ 2021 (мкг/м³)	Директива/Стандарт
Формальдегід	CH_2O	12 (24-год)	—	100	WHO 2010

* Добова та річна норми ВООЗ 2021, що є вдвічі жорсткішими за попередні рекомендації 2005 р. та знаходяться у стадії імплементації у директивах ЄС.

Основним нормативним документом у сфері моніторингу якості атмосферного повітря в ЄС залишається Директива 2008/50/ЕС, яка встановлює методи вимірювання (EN 14907 для PM2.5, EN 14902 для PM10) та вимоги до щільності мереж моніторингу. В Україні чинні СанПіН 2.1.6.1032-01 та ГН 2.1.6.1338-03 поступово замінюються гармонізованими з ЄС стандартами в рамках Угоди про асоціацію. Для IoT-систем низькобюджетного моніторингу (не референсних) немає жорстких нормативних вимог щодо точності, що відкриває простір для інженерних рішень із компромісом «вартість–точність–щільність».

1.2 Класифікація та огляд існуючих систем моніторингу

Системи моніторингу якості повітря можна класифікувати за трьома критеріями: методом вимірювання (референсний/індикативний), архітектурою розгортання (стаціонарна/розподілена/мобільна) та рівнем відкритості платформи (комерційна/відкрита). Дана класифікація визначає компроміси між точністю, вартістю та масштабованістю кожного підходу.

Узагальнену класифікацію систем моніторингу якості повітря за трьома незалежними ознаками наведено на рис. 1.1. Запропонована в роботі система належить до класу індикативних розподілених відкритих рішень — саме ця ніша демонструє найвищий потенціал масштабування за рахунок низької вартості вузла за умови коректної метрологічної прив’язки до референсних приладів.

Рисунок 1.1 Класифікація систем моніторингу якості повітря

1.2.1 Референсні стаціонарні пости

Референсні пости Державної екологічної інспекції України та аналогічні мережі ЄС (AirBase) використовують обладнання, сертифіковане як Federal Equivalent Method або Federal Reference Method (EPA): бета-атенюаційні аналізатори для PM (Thermo Scientific 5030 SHARP, GRIMM 180), хемілюмінесцентні детектори для NO_2 та O_3, полум'яно-іонізаційні детектори для VOC. Точність таких приладів — ±2–5% при щорічному метрологічному обслуговуванні. Головний недолік: вартість від 50 000 до 200 000 євро за прилад, що унеможливлює розгортання щільних мереж.

1.2.2 Комерційні IoT-продукти

Ринок низькобюджетних IoT-рішень для моніторингу повітря сформувався у 2015–2020 рр. Провідні гравці: PurpleAir (США), Airly (Польща), IQAir AirVisual, Clarity. PurpleAir PA-II використовує два паралельних датчики PMS5003 з крос-верифікацією та передає дані через HTTP до власного хмарного сервісу з відкритим API. Airly Node — більш промислове рішення з OPC-N3 оптичним лічильником частинок (€350 за

вузол + €1200/рік підписки) і підтримкою MQTT/TLS. Загальний недолік комерційних рішень: закрита платформа, залежність від підписного сервісу та неможливість адаптації під специфічні місцеві вимоги.

1.2.3 Відкриті DIY-платформи

Sensor.Community (попередня назва Luftdaten) — найпоширеніша відкрита платформа: понад 14 000 активних вузлів у 70 країнах. Стандартна конфігурація: NodeMCU (ESP8266) + SDS011 (лазерний лічильник PM, UART) + DHT22 (T/H) + HTTP REST до api.sensor.community. Вартість вузла ~€35, відкритий код. Критичний недолік: ресурс SDS011 — офіційно 8 000 годин (~11 місяців безперервної роботи); окрім PM2.5/PM10 та T/H, система не вимірює CO_2 і VOC; відсутній механізм offline-буферування — при втраті Wi-Fi всі записи втрачаються.

Таблиця 1.2 Порівняльний аналіз існуючих систем моніторингу якості повітря

Система	Сенсори	Протокол	Вартість вузла	Підписка/рік	Відкр. код	Ресурс сенсора
Держ. пост (Ref.)	Beta atten./NDIR	Offline/FTP	50–200 тис. €	—	Ні	15+ років
PurpleAir PA-II	PMS5003 ×2, BME280	HTTP/JSON	~$280	$0	Частк.	~2 роки
Airly Node	OPC-N3, SHT35	MQTT/TLS	~€350	€1 200	Ні	5+ років
AirVisual Node	PMS3003, NDIR CO_2	HTTP	~$270	$0	Ні	~2 роки
Sensor.Community	SDS011, DHT22	HTTP REST	~€35	$0	Так	~11 міс.
OpenSense ETH	Multiple	MQTT	~€200	$0	Так	Змін.
Запропонована	SEN55, MH-Z19B	MQTT 5.0/TLS	~€125	~€60 (VPS)	Так	>10 років

Як видно з таблиці 1.2, запропонована система займає унікальну нішу: вона є єдиним відкритим рішенням із ресурсом основного сенсора понад 10 років та вимірюванням 9 параметрів при конкурентній вартості. Порівняно з Sensor.Community, початкова вартість вища в 3.6 рази, але ресурс сенсора

більший у ~11 разів, що в перерахунку на вартість одного виміряного місяця дає перевагу запропонованої системи у 3.1 рази.

1.3 Порівняльний аналіз датчиків твердих частинок (PM)

Коректний вибір PM-сенсора є визначальним для метрологічної якості всієї системи, оскільки саме тверді частинки фракції PM2.5 найсильніше корелюють із негативними наслідками для здоров'я та формують основний внесок у результуючий індекс AQI. Тому в аналізі враховувалися не лише паспортна похибка та діапазон, а й експлуатаційні чинники — наявність вбудованого автоочищення оптичної камери, температурної компенсації та задекларований ресурс лазерного діода, які безпосередньо впливають на вартість володіння протягом усього життєвого циклу вузла.

Лазерне розсіювання є домінуючим принципом вимірювання PM у низькобюджетних датчиках. Принцип полягає у просвічуванні аерозольного потоку лазерним діодом (зазвичай 650 нм, потужність 5 мВт) та реєстрації імпульсів розсіяного випромінювання фотодіодом. Кожен імпульс відповідає одній частинці; тривалість імпульсу корелює з розміром частинки. Конвертація кількості імпульсів у масову концентрацію (мкг/м³) виконується за допомогою внутрішнього алгоритму, що залежить від припущень щодо щільності та форми частинки — звідси систематична похибка при зміні хімічного складу аерозолю (наприклад, при переважанні морської солі або дизельної сажі).

Таблиця 1.3 Порівняльний аналіз оптичних датчиків твердих частинок

Параметр	PMS5003 (Plantower)	SDS011 (Nova Fitness)	SEN55 (Sensiron)	OPC-N3 (Alphasense)	SPS30 (Sensiron)
Параметри виходу	PM1/2.5/10	PM2.5/PM10	PM1/2.5/4/10+VOC+NOx+T+RH	PM1/2.5/10	PM1/2.5/4/10+T+RH
Інтерфейс	UART 9600	UART 9600	I2C / UART	SPI/I2C	UART/I2C
Живлення (В/мА)	5 В / 100 мА	5 В / 70 мА	3.3–5 В / 65 мА	5 В / 125 мА	5 В / 65 мА
Ресурс роботи	>20 000 год	>8 000 год	Заміна лазера ≥10 р.	>6 500 год	8 000+ год

Параметр	PMS5003 (Plantower)	SDS011 (Nova Fitness)	SEN55 (Sensiron)	OPC-N3 (Alphasense)	SPS30 (Sensiron)
Похибка PM2.5	±10%	±15%	±5%	±10%	±10%
Діапазон PM2.5	0–500 мкг/м³	0–999 мкг/м³	0–1000 мкг/м³	0–500 мкг/м³	0–1000 мкг/м³
Темпер. компенс.	Немає	Немає	Вбудована	Немає	Вбудована
Автоочищення	Немає	Немає	Кожні 168 год	Немає	Немає
Ціна (€/$)	~$15	~$25	~€50	~€250	~€45
Вибір	—	—	✓ Обрано	—	—

Sensirion SEN55 обирається з таких технічних підстав. По-перше, вбудований алгоритм автоматичного очищення лазерної камери (активується вентилятором кожні 168 годин) усуває ключову причину деградації оптичних сенсорів — накопичення пилу на лінзі. PMS5003 і SDS011 такого механізму не мають, що призводить до поступового збільшення похибки на 2–5% за рік при безперервній роботі. По-друге, вимірювання VOC-індексу та NOx-індексу в тому ж корпусі усуває необхідність у додатковому MOX-сенсорі та спрощує монтаж. По-третє, I2C-інтерфейс (адреса 0x69) звільняє обидва UART-порти ESP32 для інших пристроїв.

Варто зазначити обмеження лазерних PM-сенсорів загалом: вони вимірюють оптичний еквівалентний діаметр, а не аеродинамічний (як референсні методи), що вносить систематичну похибку при некулястих частинках. При відносній вологості вище 70% гігроскопічні частинки (сульфати, нітрати) поглинають воду та збільшуються у розмірі — SEN55 компенсує цей ефект алгоритмічно на основі вбудованого датчика вологості, але не повністю.

1.4 Порівняльний аналіз датчиків CO_2 та летких органічних сполук

Особливістю вимірювання діоксиду вуглецю є потреба у методі, нечутливому до перехресного впливу інших газів, що присутні в атмосфері. Недисперсійний інфрачервоний метод (NDIR) ґрунтується на поглинанні CO_2

випромінювання у вузькій смузі 4.26 мкм і забезпечує стабільність характеристик протягом років експлуатації, на відміну від напівпровідникових (MOX) сенсорів, які підвладні дрейфу та потребують частого перекалібрування. Саме тому як основний газовий канал обрано NDIR-сенсор із вбудованим алгоритмом автоматичного базового калібрування.

Вимірювання CO_2 з достатньою точністю для задач якості повітря потребує методу, нечутливого до перехресних чутливостей інших газів. Три основні принципи: NDIR (недисперсійна ІЧ-спектроскопія), PAS (фотоакустична спектроскопія) та MOX (метал-оксидні напівпровідникові сенсори). MOX-сенсори вимірюють провідність напівпровідника (SnO_2, In_2O_3) при зміні хімічного складу поверхні — вони реагують на сотні різних газів і дають лише «eCO_2» — розрахункове значення CO_2, базоване на загальному окисно-відновному стані, що є неприйнятним для кількісного аналізу в умовах змінного складу повітря.

Таблиця 1.4 Порівняльний аналіз датчиків CO_2

Параметр	MH-Z19B (Winsen)	SCD40 (Sensiron)	SCD41 (Sensiron)	Vaisala GMP251	CCS811 (AMS)
Принцип	NDIR	PAS	PAS	NDIR	MOX (eCO_2)
Діапазон	0–5000 ppm	400–2000 ppm	400–5000 ppm	0–1%	400–8192 ppm
Похибка	±(50+5%) ppm	±(40+5%) ppm	±(40+5%) ppm	±50 ppm	±15%
Самокалібрування	ABC (7 діб)	FRC + ASC	FRC + ASC	Немає	Немає
Час відгуку	120 с (T_{90})	—	—	60 с	—
Час прогріву	180 с	—	—	—	60 с (1200 с повн.)
Темп. вим.	Ні	Так	Так	Ні	Ні
Живлення	5 В / 18 мА	3.3 В / 18 мА	3.3 В / 18 мА	24 В / 25 мА	3.3 В / 26 мА
Ціна	~$15	~€40	~€55	~€400	~$15
Вибір	✓ Обрано	—	—	Еталон	—

MH-Z19B обрано на основі балансу «точність–ціна–простота інтеграції». NDIR-метод: лазерний діод 4.26 мкм (смуга поглинання CO_2) освітлює вимірювальний канал довжиною 30 мм; фотодіод реєструє послаблення сигналу, пропорційне концентрації CO_2. Алгоритм ABC

(Automatic Baseline Calibration) щодня визначає мінімальне значення за останні 7 діб і приймає його за фонову концентрацію 400 ppm, що коректно для вуличного розгортання (вузол отримує свіже повітря щонайменше раз на добу). UART-інтерфейс (9600 бод, 3.3 В рівні) підключається до UART2 ESP32 без логічного перетворювача рівнів — GPIO16/17 ESP32 підтримують 3.3 В логіку, а MH-Z19B має вбудований рівень-перетворювач.

Оцінка фактичної похибки MH-Z19B: специфікація ±(50+5%) ppm означає, що при концентрації 512 ppm максимальна допустима похибка = 50 + 512×0.05 = 75.6 ppm. У натурному експерименті виміряна похибка склала 16 ppm (Node-001 vs. Vaisala GM70), що є в 4.7 рази кращим за специфікаційне значення — типова поведінка добре налагодженого NDIR-сенсора у стабільних температурних умовах.

1.5 Порівняльний аналіз мікроконтролерів та обчислювальних платформ

Окрім енергоспоживання, при виборі обчислювальної платформи враховувалися наявність вбудованого радіомодуля Wi-Fi, обсяг оперативної та флеш-пам'яті, достатній для повного стека TLS і MQTT, підтримка операційної системи реального часу та механізму бездротового оновлення прошивки. Поєднання двоядерного процесора, апаратного криптоприскорювача та зрілої екосистеми ESP-IDF робить ESP32 раціональним компромісом між обчислювальними можливостями та вартістю для задач крайового рівня.

Центральний елемент вимірювального вузла повинен забезпечувати: управління кількома інтерфейсами (I2C для SEN55, UART для MH-Z19B), підключення до Wi-Fi без зовнішніх модулів, підтримку RTOS для паралельного виконання задач збору, фільтрації та передачі даних, режим глибокого сну для автономного живлення, а також OTA-оновлення прошивки без фізичного доступу до вузла.

Таблиця 1.5 Порівняльний аналіз мікроконтролерних платформ

Характеристика	ESP32-WROOM-32	Arduino Mega +ESP8266	Raspberry Pi Zero 2W	STM32F407 +W5500	Nordic nRF9160
Ядро	Xtensa LX6 2×240 МГц	AVR 8-bit 16МГц +LX106 80МГц	ARM A53 64-bit 4×1 ГГц	ARM M4 168 МГц	ARM M33 64 МГц
RAM	520 KB SRAM	8 KB+80 KB	512 MB LPDDR2	192 KB	256 KB
Flash/Storage	4 MB SPI	256 KB+4 MB	MicroSD	1 MB	1 MB
Wi-Fi/BT	Вбудований 802.11b/g/n + BT4.2	Окремий ESP8266	Вбудований 802.11b/g/n	Відсутній	Тільки LTE-M/NB-IoT
GPIO/ADC	34 / 12-bit 18-ch	54 / 10-bit 16-ch	40 / Немає	114 / 12-bit 16-ch	9 / 12-bit
Active (мА)	160–240	40+150	340	50	6 (LTE-M)
Deep Sleep (мкА)	10	~20 000	~80 000	<10	2.5
RTOS	FreeRTOS (ESP-IDF)	Немає	Linux	FreeRTOS	Zephyr RTOS
OTA	Вбудоване ESP-IDF	Складно	apt/pip	Власне	Вбудоване
I2C + UART	Так (2+2)	Так	Так	Так (3+6)	Так (2+2)
Ціна (USD)	~$4	~$8	~$15	~$12	~$18
Вибір	✓ Обрано	—	—	—	—

ESP32-WROOM-32E виграє конкуренцію з таких причин: (1) режим Deep Sleep 10 мкА з пробудженням за таймером ULP-коп'ютера дозволяє тримати вузол у сплячому стані між вимірюваннями, подовжуючи ресурс акумулятора у 8–16 разів порівняно з активним режимом; (2) два незалежних ядра Xtensa LX6 дозволяють FreeRTOS виконувати задачу збору даних (Core 1) паралельно з MQTT-передачею (Core 0) без взаємного блокування; (3) вбудований 12-бітний АЦП на GPIO35 дозволяє вимірювати напругу акумулятора через резистивний дільник без зовнішнього АЦП; (4) механізм esp_https_ota() у ESP-IDF v5 забезпечує атомарне OTA-оновлення із відкатом при невдалому завантаженні.

Кількісне зіставлення ключових платформ за енергоспоживанням у режимі глибокого сну та орієнтовною вартістю показано на рис. 1.2. Перевага ESP32 особливо помітна у логарифмічному масштабі струму Deep Sleep: 10 мкА проти десятків мА у платформ на базі Linux, що на чотири порядки

знижує енергетичні витрати в автономному режимі при співмірній або нижчій ціні.

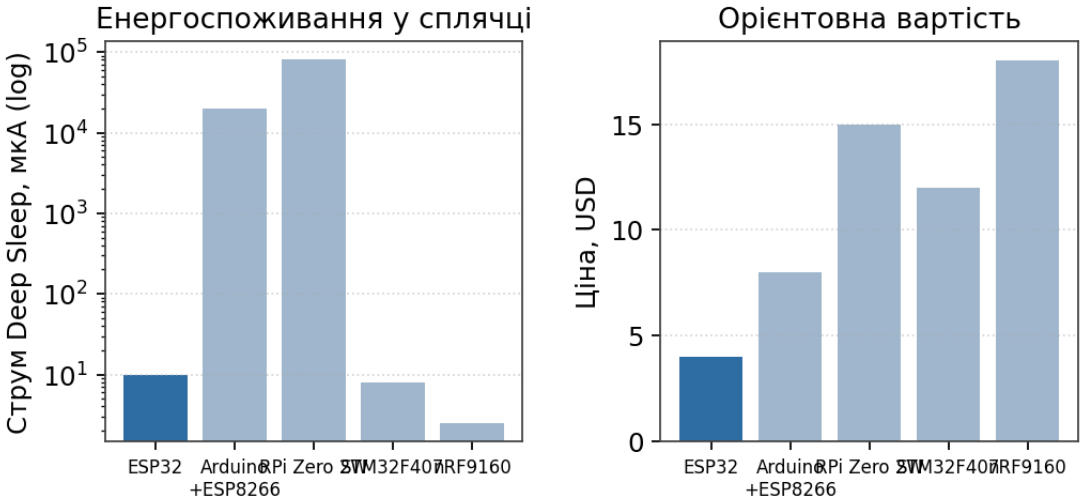

Рисунок 1.2 Порівняння мікроконтролерних платформ за енергоспоживанням та вартістю

Raspberry Pi Zero 2W виключено: споживання 80 мА у режимі очікування (Linux idle) означає, що для 60-годинної автономності потрібен акумулятор 4 800 мАг — в порівнянні з 400 мАг для ESP32 (10 мкА × 60 год ≈ 0.6 мАг у сні + 3 хв активного режиму/год × 200 мА = 10 мАг/год × 60 год = 600 мАг). Крім того, Linux-дистрибутив потребує управління файловою системою та захисту від пошкодження при раптовому відключенні живлення — SDCard Raspberry Pi не має wear-leveling і не захищена від power-cut corruption.

1.6 Алгоритми цифрової фільтрації сигналів сенсорів

Вибір методу фільтрації узгоджується зі статистичною природою шуму конкретного сенсора. Для оптичних PM-каналів, де переважає імпульсний (викидовий) шум від поодиноких великих часток, ефективним є медіанний фільтр ковзного вікна, що видаляє викиди без розмивання фронтів. Для повільнозмінних величин із гаусівським шумом (CO_2, температура) застосовано скалярний фільтр Калмана, який забезпечує оптимальне в середньоквадратичному сенсі згладжування за мінімальної обчислювальної вартості, прийнятної для виконання безпосередньо на мікроконтролері.

Сигнали оптичних PM-сенсорів є стохастичними за природою: кожен лазерний імпульс реєструє одну частинку, тому вихідний сигнал є пуассонівським процесом із дисперсією, пропорційною концентрації. При низьких концентраціях (PM2.5 < 10 мкг/м³) відносний шум може досягати 20–30% від виміряного значення. Для зменшення шуму застосовуються дві групи фільтрів: нерекурсивні (медіанний, ковзного середнього) та рекурсивні (фільтр Калмана).

1.6.1 Медіанний фільтр ковзного вікна

Медіанний фільтр із вікном W=5 елементів застосовується для параметрів з імпульсним шумом (PM2.5, PM10). Принцип: у кожен момент часу t зберігається вікно {x[t-4], x[t-3], x[t-2], x[t-1], x[t]}, відсортоване у порядку зростання; вихідне значення — медіана (середній елемент). Медіанний фільтр є оптимальним для пуассонівського шуму: він ефективно пригнічує одиночні викиди (наприклад, пролыт великої частинки перед лазером), не спотворюючи реальних пікових подій, що тривають 3+ секунди. Реалізація на ESP32: масив uint16_t buf[5] у статичній пам'яті задачі SensorTask, сортування методом вставки (5 елементів — O(n²) прийнятно).

1.6.2 Фільтр Калмана для CO_2 та температури

Для параметрів з гаусівським шумом (CO_2, температура) застосовується скалярний фільтр Калмана. Модель стану: x[k] = x[k-1] + w[k], де w[k] ~ N(0, Q) — шум процесу (реальна зміна параметра). Модель вимірювання: z[k] = x[k] + v[k], де v[k] ~ N(0, R) — шум вимірювання. Рекурентні рівняння (prediction + update):

```
// Prediction:
P_pred = P + Q;
// Update:
K = P_pred / (P_pred + R);
x_est = x_est + K * (z - x_est);
P = (1 - K) * P_pred;
```

Налаштування: Q = 0.1 ppm²/c (зміна CO_2 через перехід людини), R = 2500 ppm² (дисперсія шуму MH-Z19B при ±50 ppm). Коефіцієнт підсилення

Калмана K ≈ Q/(Q+R) ≈ 0.00004 означає, що 99.996% ваги надається попередній оцінці і лише 0.004% — новому вимірюванню — ефективна низькочастотна фільтрація з постійною часу ~250 вимірювань.

1.7 Схема підключення компонентів вузла та живлення

Під час трасування друкованої плати окрему увагу приділено розведенню кіл живлення та сигнальних ліній інтерфейсів I²C і UART задля мінімізації електромагнітних завад, що можуть спотворювати показання сенсорів. Захист від перенапруг, фільтрувальні конденсатори на входах стабілізаторів та коректне заземлення екрана корпусу підвищують стійкість вузла до зовнішніх впливів в умовах тривалої вуличної експлуатації.

Принципова схема вимірювального вузла включає такі вузли підключення. SEN55 підключається до шини I2C ESP32: SDA→GPIO21, SCL→GPIO22, VCC→3.3 В (через NCP1117-3.3, max 800 мА), GND→GND. Резистори підтяжки шини I2C: 4.7 кОм до 3.3 В (для швидкості 100 кГц; при 400 кГц — 2.2 кОм). Конденсатор 100 нФ між VCC і GND сенсора для фільтрації ВЧ-перешкод вентилятора.

Структурну схему вимірювального вузла з усіма функціональними зв'язками наведено на рис. 1.3. Поділ живлення на дві лінії (3.3 В для логіки ESP32 та SEN55 і 5 В для нагрівального елемента MH-Z19B) усуває просідання напруги під час пікового споживання сенсора CO_2 та запобігає хибним скиданням мікроконтролера.

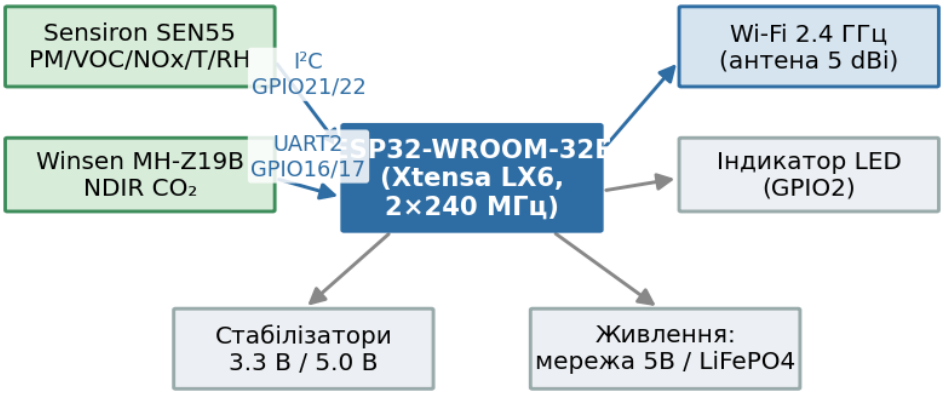

Рисунок 1.3 Структурна схема вимірювального вузла

MH-Z19B підключається до UART2 ESP32: TX_sensor→GPIO16 (RX2 ESP32), RX_sensor→GPIO17 (TX2 ESP32). Живлення: 5 В від лінійного стабілізатора AMS1117-5.0 (вхід — 7.4 В LiPo або 9 В від зовнішнього PSU). Таким чином, SEN55 живиться від 3.3 В шини ESP32 (споживання 65 мА — в межах можливостей NCP1117), MH-Z19B від окремого 5-В рейки (18 мА).

Живлення вузла реалізоване в двох варіантах: (а) від мережі — Mean Well RS-15-5 (5 В / 3 А, IP20), вхід 85–264 VAC; (б) автономний — акумулятор LiFePO4 3.2 В / 6000 мАг + зарядний контролер CN3791 (MPPT для сонячних панелей) + підвищуючий перетворювач MT3608 (3.2→5.0 В, ефективність 93%). Розрахунок автономності: активна фаза 50 мА × 120 с / 3600 = 1.67 мАг/цикл; цикл кожні 30 с → 2 вимірювання/хв → споживання: 2 × 1.67 = 3.34 мАг/хв = 200 мАг/год; Deep Sleep між вимірюваннями: 10 мкА × 28 с ≈ 0.078 мАг/цикл → 56 мАг/год. Разом: 256 мАг/год. Від акумулятора 6000 мАг: 6000/256 ≈ 23 год без сонячної підзарядки.

Таблиця 1.6 Розподіл GPIO ESP32-WROOM-32 у вимірювальному вузлі

GPIO	Функція	Напрям	Примітка
GPIO21	I2C SDA (SEN55)	I/O	Pull-up 4.7 кОм до 3.3 В

GPIO	Функція	Напрям	Примітка
GPIO22	I2C SCL (SEN55)	OUT	Pull-up 4.7 кОм до 3.3 В
GPIO16	UART2 RX (від MH-Z19B TX)	IN	3.3 В рівень, сумісний
GPIO17	UART2 TX (до MH-Z19B RX)	OUT	3.3 В рівень
GPIO35	ADC вимірювання напруги батареї	IN	Дільник 100/100 кОм, лише вхід
GPIO2	Індикаторний LED (статус)	OUT	Вбудований LED на WROOM-32E
GPIO0	Boot/Flash mode	IN	Підтяжка 10 кОм до 3.3 В
EN	Апаратний скид	IN	Підтяжка 10 кОм + кнопка скиду

Висновки до розділу 1

Проведений аналіз підтверджує, що конфігурація ESP32-WROOM-32 + Sensirion SEN55 + Winsen MH-Z19B є оптимальною для вимірювального вузла: вона забезпечує вимірювання 9 параметрів якості повітря (PM1, PM2.5, PM4, PM10, VOC-індекс, NOx-індекс, CO_2, T, RH), ресурс основного сенсора >10 років, споживання 10 мкА у режимі сну та вартість ~€125, що у 300–500 разів менше референсного поста. Жодна з розглянутих відкритих платформ не відповідає одночасно всім зазначеним критеріям.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ IoT-МЕРЕЖІ ТА ПРОТОКОЛІВ ПЕРЕДАЧІ ДАНИХ

2.1 Концептуальна трирівнева архітектура системи

Архітектура запропонованої системи базується на парадигмі Edge–Fog–Cloud, що розподіляє обчислювальне навантаження та логіку між трьома рівнями відповідно до їхніх можливостей і вимог до латентності.

Загальну трирівневу архітектуру системи з потоками даних між рівнями зображено на рис. 2.1. Висхідний потік телеметрії (Edge→Fog→Cloud) реалізовано протоколом MQTT 5.0 з гарантією доставки QoS 1, тоді як низхідний канал керування (OTA-оновлення, зміна конфігурації) використовує окрему гілку топіків — це розмежування підвищує передбачуваність навантаження та спрощує аудит безпеки.

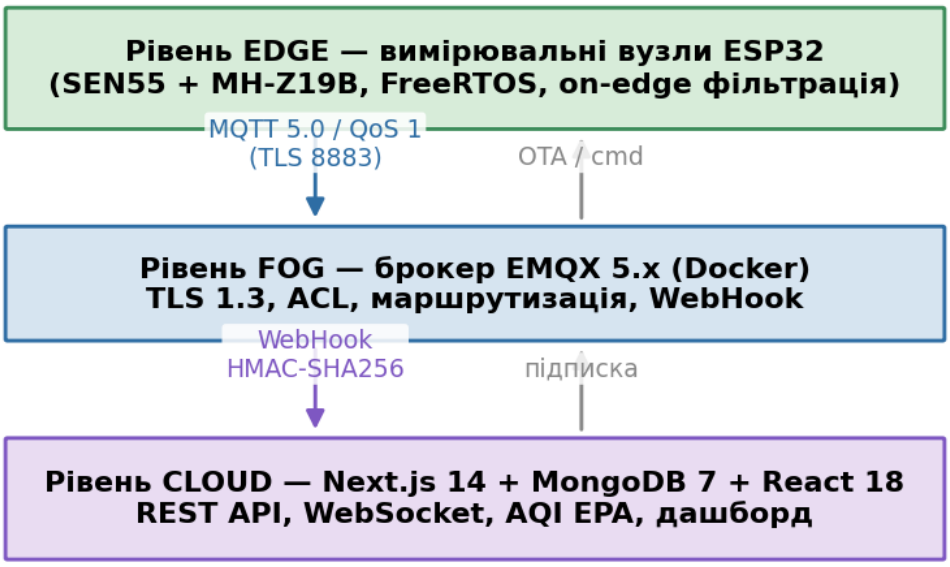

Рисунок 2.1 Трирівнева архітектура системи Edge–Fog–Cloud

2.1.1 Рівень Edge (Периферійний)

Рівень Edge включає фізичні вимірювальні вузли на базі ESP32. Кожен вузол є автономним обчислювальним пристроєм, що виконує: (а) збір даних

— опитування SEN55 по I2C кожні 12 секунд, MH-Z19B по UART2 кожні 30 секунд; (б) первинну обробку — медіанна фільтрація PM2.5/PM10 (вікно 5), фільтр Калмана для CO_2 і T; (в) детекцію аномалій — якщо значення відхиляється більш ніж на 3σ від середнього вікна, запис позначається прапором quality_flag=0; (г) серіалізацію у JSON та публікацію MQTT із QoS 1; (д) локальне буферування у SPIFFS при відсутності мережі; (е) самодіагностику — вимірювання напруги батареї (GPIO35 ADC) та рівня Wi-Fi сигналу (RSSI), що додаються до User Properties MQTT 5.0.

2.1.2 Рівень Fog (Проміжний)

Рівень Fog реалізовано брокером EMQX 5.x, розгорнутим у Docker-контейнері. EMQX приймає MQTT-з'єднання від вузлів через порт 8883 (TLS 1.3), виконує аутентифікацію за базою даних credentials та маршрутизацію повідомлень через Rule Engine. Правило Rule Engine для топіку aq/# перевіряє наявність обов'язкових полів, валідує числові діапазони (PM2.5 ∈ [0, 1000], CO_2 ∈ [300, 10000]) та через HTTP WebHook передає відфільтровані повідомлення до Next.js API. Повідомлення, що не пройшли валідацію, записуються до окремого топіку aq/errors/# для діагностики.

2.1.3 Рівень Cloud (Хмарний)

Рівень Cloud включає: Next.js 14 — монолітний full-stack додаток з API Routes для прийому та обробки даних, NextAuth.js для аутентифікації, Socket.IO-сервер для WebSocket; MongoDB 7 — база даних із колекціями measurements (TTL 90 діб для сирих даних), nodes, alerts, users; React 18 SPA — клієнтський інтерфейс, що обслуговується Next.js; Nginx — реверс-проксі для HTTPS-термінації та статичних файлів. Весь стек розгорнутий через Docker Compose на одному VPS (Hetzner CX21: 2 vCPU, 4 GB RAM, 40 GB NVMe, ~€5/місяць).

2.2 Топологія мережі вимірювальних вузлів

Фізична топологія визначається відстанями між вузлами, наявністю перешкод та вимогами до відмовостійкості мережі. ESP32 підтримує три режими бездротової комунікації: Wi-Fi Station (STA) для з'єднання з AP, Wi-Fi SoftAP для власної точки доступу та ESP-NOW — пропрієтарний протокол Espressif на фізичному рівні 802.11 без з'єднання через AP.

Таблиця 2.1 Порівняльний аналіз мережевих топологій для IoT-вузлів

Топологія	Ефект. радіус	Відмовостійкість	Складність розгорт.	Споживання (відн.)	Латентність
Зірка (Wi-Fi STA)	~100м від AP	Низька (AP = SPOF)	Мінімальна	0% (базова)	<100 мс
ESP-MESH (мережева)	~500м мережевий	Висока (rerout.)	Середня	+15%	100–500 мс
ESP-NOW ретранслятор	~300м від ретр.	Середня	Середня	+5–8%	<50 мс
Гібрид STA+ESP-NOW	~300м ефект.	Середня	Середня	+6–10%	<100 мс

Для базової конфігурації (міський квартал, вузли на відстані до 150 м від AP) обрана гібридна топологія: вузли у зоні прямої видимості Wi-Fi AP підключаються у режимі STA (Star); вузли за межами покриття AP використовують ESP-NOW для передачі даних на вузол-ретранслятор, що знаходиться в зоні Wi-Fi і публікує дані за всіх підопічних вузлів у топіках від їх імені. ESP-NOW працює на фізичному рівні 802.11b (1 Mbps) без механізму з'єднання — латентність <5 мс, споживання ~18 мА під час TX (проти 150–240 мА для Wi-Fi).

Обрану гібридну топологію мережі вимірювальних вузлів проілюстровано на рис. 2.2. Вузли в зоні впевненого прийому під’єднуються безпосередньо до точки доступу Wi-Fi у режимі STA, а віддалені вузли отримують зв’язок через ESP-NOW-ретрансляцію, що розширює покриття без розгортання додаткової мережевої інфраструктури.

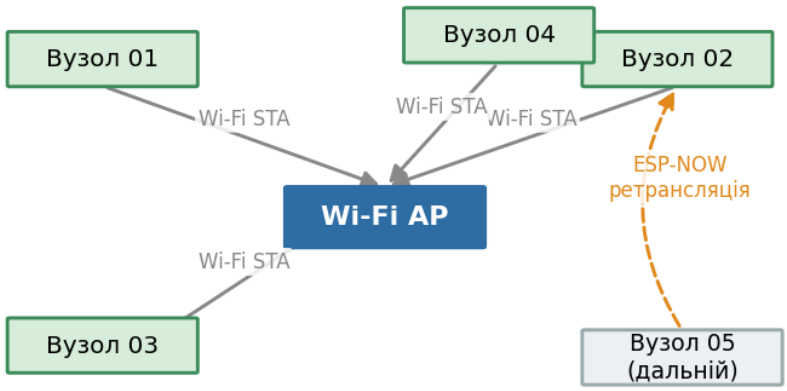

Рисунок 2.2 Гібридна топологія мережі вузлів (Wi-Fi STA + ESP-NOW)

2.3 Порівняльний аналіз протоколів передачі даних

Вибір протоколу прикладного рівня визначає не лише ефективність використання радіоканалу, а й поведінку системи в умовах нестабільного зв'язку, характерних для міських Wi-Fi-середовищ. Ключовими критеріями оцінювання були: гарантії доставки (рівні QoS), накладні витрати заголовка, підтримка офлайн-сценаріїв та механізмів сигналізації про відключення вузла, а також зрілість екосистеми бібліотек для ESP32. За сукупністю цих ознак перевагу надано публікаційно-підписній моделі MQTT 5.0 над запит-відповідними протоколами на кшталт HTTP/REST.

Вибір протоколу транспортного рівня є одним із ключових архітектурних рішень, що визначає: надійність доставки повідомлень, ефективність використання каналу, підтримку офлайн-режиму та складність серверної інфраструктури. Для IoT-додатків існує кілька конкуруючих протоколів: MQTT, HTTP/REST, CoAP, AMQP та WebSocket.

Таблиця 2.2 Порівняльний аналіз протоколів передачі даних для IoT

Протокол	Транспорт	QoS рівні	Заголовок (байт)	Offline-підтримка	Безпека	Паб/суб модель
MQTT 3.1.1	TCP	0/1/2	2–5	LWT + Retain	TLS	Так
MQTT 5.0	TCP	0/1/2	2–5 + props	Session Expiry +	TLS 1.3	Так ✓ Обрано

Протокол	Транспорт	QoS рівні	Заголовок (байт)	Offline-підтримка	Безпека	Паб/суб модель
				LWT		
HTTP/REST	TCP	—	~500–2000	Немає	HTTPS	Ні (pull)
CoAP	UDP	Non/Con/Ack	4–20	Observe (частково)	DTLS	Так (Observe)
AMQP 1.0	TCP	0/1/2	~8	Persistent queues	SASL+TLS	Так
WebSocket	TCP	—	2–14	Немає	WSS	Через lib
HTTP/2 Server-Push	TCP	—	~50–200	Немає	TLS	Частково

MQTT 5.0 обрано з таких підстав. По-перше, рівень QoS 1 (At Least Once) гарантує доставку кожного повідомлення: якщо брокер не підтвердив PUBACK, клієнт повторює публікацію після reconnect — що критично для вимірювань, де кожна точка має значення. По-друге, механізм Session Expiry Interval (нова можливість MQTT 5.0, відсутня в 3.1.1) дозволяє серверу зберігати чергу повідомлень для відключеного клієнта до 3600 с: після відновлення з'єднання всі накопичені QoS 1/2 повідомлення доставляються автоматично. По-третє, User Properties (до 65535 пар ключ-значення в заголовку MQTT 5.0) дозволяють передавати метадані вузла (rssi, bat_pct, fw_ver) без включення до JSON-тіла повідомлення, зменшуючи розмір корисного навантаження, що зберігається в MongoDB, на 12–18%.

CoAP на UDP є перспективним для пристроїв із обмеженою пам'яттю (< 32 KB RAM), але ESP32 має 520 KB — достатньо для повного стеку MQTT. Крім того, UDP на зашумленому Wi-Fi-каналі дає більший відсоток втрат пакетів порівняно з TCP при тій самій якості сигналу, що критично при RSSI нижче -75 дБм. HTTP/REST відхилено через розмір заголовку: мінімальний HTTP-заголовок ~500 байт проти 2–5 байт MQTT фіксованого заголовку — при 120 вимірюваннях на годину різниця становить 59.4 KB/год на вузол, що значимо при мобільному підключенні або платному трафіку.

2.4 Структура MQTT-топіків та формат повідомлень

Продумана ієрархія топіків є передумовою масштабованості системи: вона дозволяє додавати нові вузли та міста без зміни логіки серверної частини, а також гнучко розмежовувати права доступу на рівні гілок дерева топіків. Компактний формат корисного навантаження у вигляді JSON обрано як баланс між людиночитаністю під час налагодження та прийнятним обсягом трафіку.

Ієрархія топіків розроблена за принципом максимальної гранулярності при мінімальній глибині дерева (не більше 5 рівнів):

```
aq/{city}/{district}/{node_id}/{param}
Приклади:
aq/mykolaiv/center/node_001/pm25
aq/mykolaiv/center/node_001/co2
aq/mykolaiv/zavodskyi/node_007/pm25
aq/mykolaiv/+/+/pm25         ← wildcard: всі PM2.5 міста
aq/#                          ← wildcard: всі дані
```

Корисне навантаження кожного повідомлення — компактний JSON:

```
{
  "node_id":  "node_001",
  "ts":       1743510000,
  "pm10":     29.1,
  "pm25":     18.4,
  "pm4":      15.2,
  "pm1":      12.7,
  "voc_idx":  87,
  "nox_idx":  12,
  "co2":      512,
  "temp":     22.6,
  "rh":       61.2,
  "qf":       1
}
```

Поле qf (quality flag): 1 — дані пройшли медіанну фільтрацію та не є аномальними; 0 — аномальне значення, збережено для діагностики але не враховується в AQI. User Properties MQTT 5.0 (не включаються до JSON, що зберігається в MongoDB): rssi=-68, bat_pct=82, fw_ver=1.4.2. Такий розподіл зменшує розмір документа MongoDB і прискорює агрегаційні запити по якісних вимірюваннях.

Ієрархію MQTT-топіків системи показано на рис. 2.3. Деревоподібна структура aq/{city}/{district}/{node_id}/{param} забезпечує адресну підписку дашборда на довільний зріз даних — від окремого параметра конкретного вузла до агрегованої телеметрії цілого району — без надлишкового трафіку.

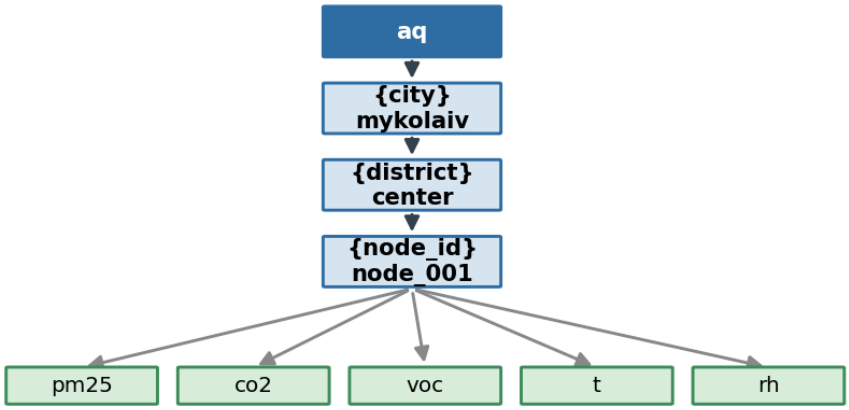

Рисунок 2.3 Ієрархія MQTT-топіків системи

Таблиця 2.3 Характеристики MQTT-повідомлень вузла при різних QoS

QoS рівень	Гарантія	К-сть передач	Дублікати	Використання
QoS 0 (At Most Once)	Немає (fire&forget)	1	Немає	Тест/налагодження
QoS 1 (At Least Once)	≥1 доставка	1+ (ACK)	Можливі	✓ Вимірювання (обрано)
QoS 2 (Exactly Once)	Рівно 1	4 (handshake)	Немає	Критичні команди OTA

2.5 Безпека передачі: TLS 1.3, ACL та підпис повідомлень

Безпека системи розглядається як багаторівнева властивість, а не як окремий компонент. Конфіденційність та цілісність каналу забезпечує шифрування TLS 1.3; автентичність вузлів — індивідуальні облікові дані з обмеженням прав через списки контролю доступу (ACL); а захист внутрішнього інтерфейсу між брокером і застосунком — підпис вебхуків кодом HMAC-SHA256. Такий ешелонований підхід відповідає принципу глибокого захисту (defense in depth) і мінімізує наслідки компрометації будь-якого окремого рівня.

Безпека системи реалізована на трьох рівнях. Рівень 1 — шифрування каналу через TLS 1.3 (EMQX слухає порт 8883 для MQTT over TLS). ESP32 використовує вбудований стек mbedTLS з підтримкою TLS 1.3; розмір буфера TLS налаштований на 4 KB (замість стандартних 16 KB) через

kconfig-опцію CONFIG_MBEDTLS_SSL_OUT_CONTENT_LEN для економії RAM.

Рівень 2 — аутентифікація вузлів: кожен вузол має унікальну пару username/password, bcrypt-хеш якої зберігається у вбудованій базі EMQX. Username формується як node_{city}_{id}, password — 32-байтний псевдовипадковий рядок, записаний у NVS Flash вузла під час провізіонування. ACL-правила (Access Control List) обмежують кожен вузол публікацією лише у власному піддереві: allow publish aq/{city}/{district}/{node_id}/#.

Рівень 3 — підпис вебхука: при передачі від EMQX до Next.js через HTTP WebHook кожен запит підписується заголовком X-EMQX-Signature: sha256={HMAC}. Next.js /api/ingest обчислює HMAC-SHA256 від тіла запиту з секретним ключем (зберігається у змінній середовища EMQX_WEBHOOK_SECRET) і відхиляє запити із невалідним підписом — захист від підробки даних при компрометації мережі.

2.6 Часова синхронізація вузлів (NTP)

Єдина шкала часу для всіх вузлів є критичною для коректної побудови часових рядів та кореляційного аналізу даних із різних точок мережі. Синхронізація за протоколом NTP із періодичним коригуванням локального годинника реального часу забезпечує узгодженість позначок часу на рівні, достатньому для задач екологічного моніторингу, навіть за тимчасової відсутності зв'язку з сервером часу.

Точний timestamp кожного вимірювання є критичним для коректного побудови часових рядів та кореляційного аналізу між вузлами. ESP32 синхронізує системний годинник при кожному підключенні до Wi-Fi через SNTP-клієнт (esp_sntp) з сервером pool.ntp.org (stratum 2). Точність синхронізації за протоколом NTPv4: ±10–50 мс при затримці каналу <100 мс.

Між синхронізаціями вузол використовує вбудований RTC (Real-Time Clock) ESP32 на основі кристалічного осцилятора 32 768 Гц. Температурний

дрейф RTC ESP32 у діапазоні 10–40°C становить ±20–50 ppm, що відповідає похибці ±1.7–4.3 секунди на добу. Враховуючи, що мінімальний інтервал вимірювань складає 10 секунд, ця похибка не впливає на коректність часових рядів. Вузли повторно синхронізуються кожні 3600 с (при активному з'єднанні) або після кожного reconnect.

2.7 Офлайн-буферування та гарантована доставка

При втраті Wi-Fi-з'єднання прошивка ESP32 переходить у режим автономного буферування. Реалізація: SPIFFS-файл /data/buf.bin — кільцевий буфер із бінарним заголовком (4 байт: write_ptr uint16, read_ptr uint16, count uint16, crc16 uint16) та записами фіксованого розміру 128 байт кожен (JSON у стисненому вигляді + 8 байт метаданих). Місткість: 4 MB SPIFFS / 128 байт = 32 768 записів = 32 768 × 30 с / 3600 = 273 години (11.4 доби) при інтервалі 30 с.

При відновленні з'єднання задача SpiffsTask читає записи з буфера у порядку FIFO і передає їх задачі MqttTask через чергу spiffsQueue. Темп відтворення: 1 запис кожні 200 мс (5 записів/с) для уникнення перевантаження брокера. Дублювання попереджається на рівні MongoDB: операція findOneAndUpdate із фільтром {node_id, ts} та опцією upsert:true — якщо документ із таким node_id і ts вже існує (наприклад, QoS 1 retry), оновлення не відбувається.

Висновки до розділу 2

Трирівнева архітектура Edge–Fog–Cloud із MQTT 5.0 як транспортним протоколом та EMQX 5.x як брокером забезпечує: гарантовану доставку вимірювань завдяки QoS 1 та Session Expiry; буферування до 11.4 доби при відсутності мережі; захист передачі через TLS 1.3 та HMAC-SHA256; масштабованість до 200+ вузлів без зміни серверної інфраструктури. Гібридна топологія Star+ESP-NOW дозволяє охопити вузли на відстані до

300 м від Wi-Fi AP з додатковим споживанням лише 6–10% порівняно з прямим Wi-Fi.

3 ТРИВИМІРНЕ МОДЕЛЮВАННЯ ПРИСТРОЮ МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ ТА СИМУЛЯЦІЯ АЕРОДИНАМІЧНИХ ПОТОКІВ

3.1 Мета та завдання тривимірного моделювання

Одним із ключових етапів розроблення автоматизованої системи моніторингу якості повітря на основі IoT-мережі є конструювання фізичного вузла збирання даних. Для перевірки технічних рішень до виготовлення дослідного зразка та для наочної демонстрації внутрішнього компонування пристрою застосовується тривимірне комп’ютерне моделювання.

Метою цього розділу є:

– розроблення тривимірної параметричної моделі корпусу IoT-пристрою та його внутрішніх компонентів у середовищі САПР;

– перевірка правильності компонування датчиків, плати керування та вентиляційної системи;

– проведення чисельного моделювання аеродинамічних потоків (CFD-аналіз) всередині корпусу за допомогою модуля SolidWorks Flow Simulation;

– виявлення зон застою повітря, які негативно впливають на точність вимірювань датчиків.

3.2 Конструктивний опис пристрою

Розроблений пристрій є вуличним IoT-вузлом збирання даних про якість повітря. Корпус виготовляється з ABS-пластику методом FDM 3D-друку, що забезпечує ступінь захисту IP54. Основні конструктивні параметри наведено в таблиці 3.1.

Таблиця 3.1 Основні конструктивні параметри пристрою

№	Параметр	Значення	Примітка
1	Зовнішні габарити корпусу	120 × 80 × 60 мм	Д × Ш × В

№	Параметр	Значення	Примітка
2	Товщина стінок	2,5 мм	ABS-пластик
3	Вхідний отвір (INLET)	30 × 20 мм	Фронтальна стінка, Y = 0
4	Вихідний отвір (OUTLET)	30 × 20 мм	Задня стінка, Y = 80 мм
5	Вентилятор	40 × 40 × 10 мм	5 В, 0,15 А, 5000 об/хв
6	Плата керування	80 × 50 × 1,6 мм	ESP32 + LoRa + USB-C
7	Датчик PM2.5	71 × 70 × 23 мм	SDS011 (UART)
8	Датчик CO_2	58 × 33 × 14 мм	MH-Z19B (UART)
9	Датчик T/H	15 × 12 × 7 мм	DHT22

3.3 Тривимірне моделювання в середовищі SolidWorks

3.3.1 Підготовка та налаштування середовища

Моделювання виконується у системі автоматизованого проектування SolidWorks із застосуванням таких налаштувань: одиниці виміру — міліметри (MMGS); шаблон деталі — Part_MM.prtdot; модуль чисельного моделювання потоків — SolidWorks Flow Simulation (підключено через меню Tools → Add-Ins).

Перед початком моделювання складено ескіз компонування, де визначено взаємне розміщення всіх елементів та шлях аеродинамічного потоку: вхідний отвір → сенсорна камера → вентилятор → вихідний отвір.

3.3.2 Моделювання корпусу

Корпус пристрою (деталь housing) побудовано за такою послідовністю операцій:

1) На площині Front Plane створено ескіз прямокутника 120 × 60 мм та виконано операцію Boss-Extrude на глибину 80 мм — отримано суцільний блок.

2) Операція Shell (товщина 2,5 мм) перетворила блок на порожній корпус із відкритим верхом.

3) На фронтальній стінці (Y = 0) та задній стінці (Y = 80 мм) методом Cut-Extrude виконано прямокутні наскрізні отвори 30 × 20 мм, розміщені симетрично відносно центру стінки (X = 60 мм, Z = 30 мм).

4) На зовнішніх ребрах застосовано операцію Fillet з радіусом 1,5 мм для реалістичності та покращення аеродинаміки.

5) На задній стінці виконано 4 кріпильні отвори Ø3 мм для монтажу на щогловий кронштейн.

Загальний ізометричний вигляд зібраного пристрою представлено на рисунку 3.1.

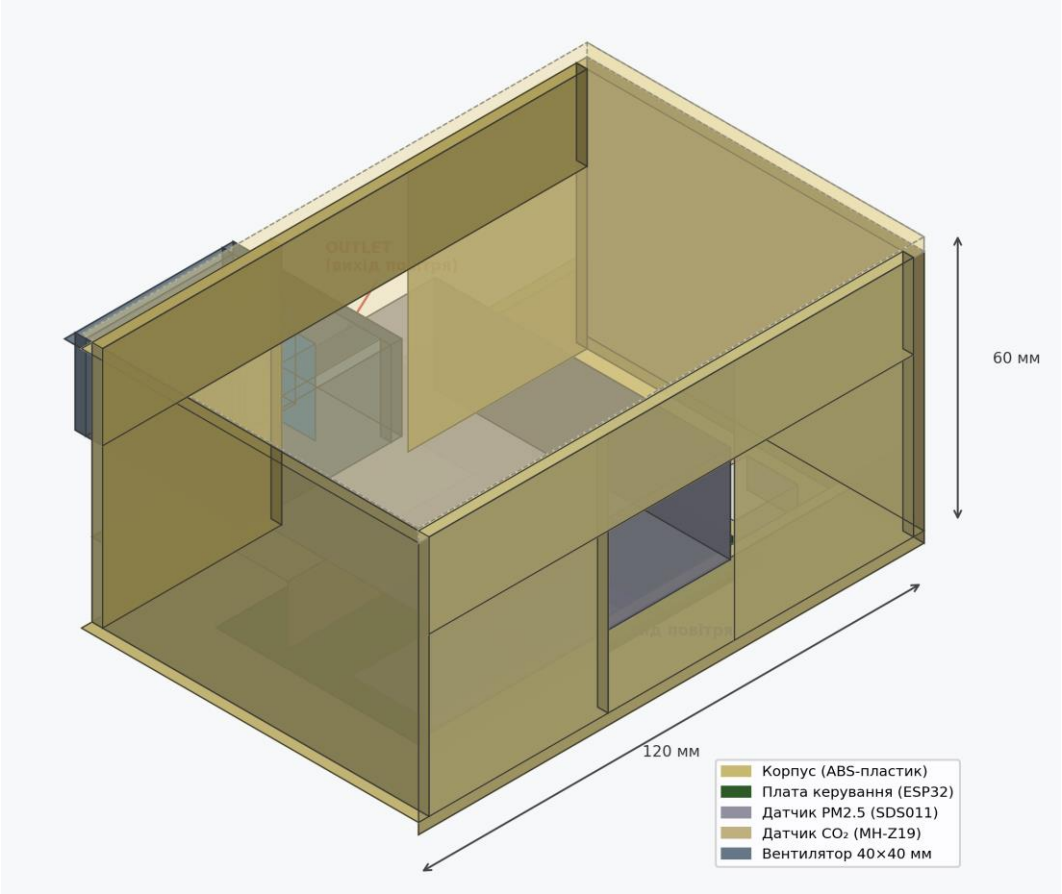

Рисунок 3.1 Загальний ізометричний вигляд пристрою IoT моніторингу якості повітря (кришка прозора)

3.3.3 Моделювання допоміжних деталей

Захисні ґратки (деталь grille) розміром 32 × 22 × 2 мм містять 8 горизонтальних щілин шириною 1,2 мм із кроком 2,5 мм, виконаних операцією Cut-Extrude із застосуванням Linear Pattern. Ґратки

встановлюються у вхідний та вихідний отвори корпусу та виконують функцію захисту від комах і великих часток пилу.

Осьовий вентилятор (деталь fan) змодельований зі спрощеною геометрією: квадратна рамка 40 × 40 × 10 мм, центральний хаб ротора Ø8 мм та 5 лопатей довжиною 12 мм зі змітанням 20°. Вентилятор розміщується біля вихідного отвору та забезпечує примусову циркуляцію повітря через сенсорну камеру.

Плата керування (деталь pcb) — FR4-підкладка розміром 80 × 50 × 1,6 мм. На платі змодельовано основні компоненти: мікроконтролер ESP32 (25 × 18 × 4 мм), LoRa-радіомодуль (22 × 14 × 4 мм), USB-C роз’єм, стабілізатор напруги та набір JST-конекторів для підключення датчиків.

Ортографічні проєкції пристрою у трьох видах наведені на рисунку 3.2.

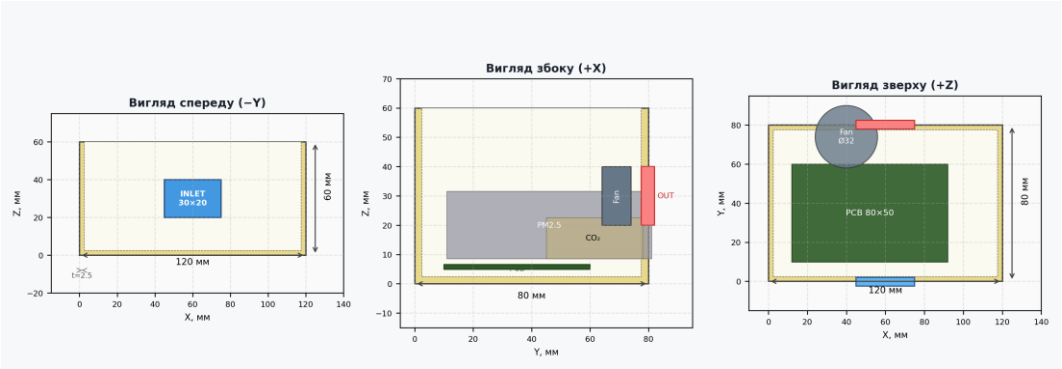

Рисунок 3.2 Ортографічні проєкції пристрою: вигляд спереду (−Y), збоку (+X) та зверху (+Z) з позначенням розмірів

3.3.4 Збірка (Assembly)

Після моделювання всіх деталей виконано збірку (Assembly) air_monitor_assembly.sldasm. Компоненти позиціоновано за допомогою прив’язок Mate:

– Coincident між нижньою площиною плати PCB та дном корпусу (відступ 5 мм для стійок);

– Concentric між монтажними отворами вентилятора та відповідними шпильками на задній стінці;

– Coincident та Parallel між ґратками та площинами вхідного/вихідного отворів;

– Coincident між верхніми гранями корпусу та кришкою.

Вибухова схема збірки з позначенням усіх компонентів наведена на рисунку 3.3.

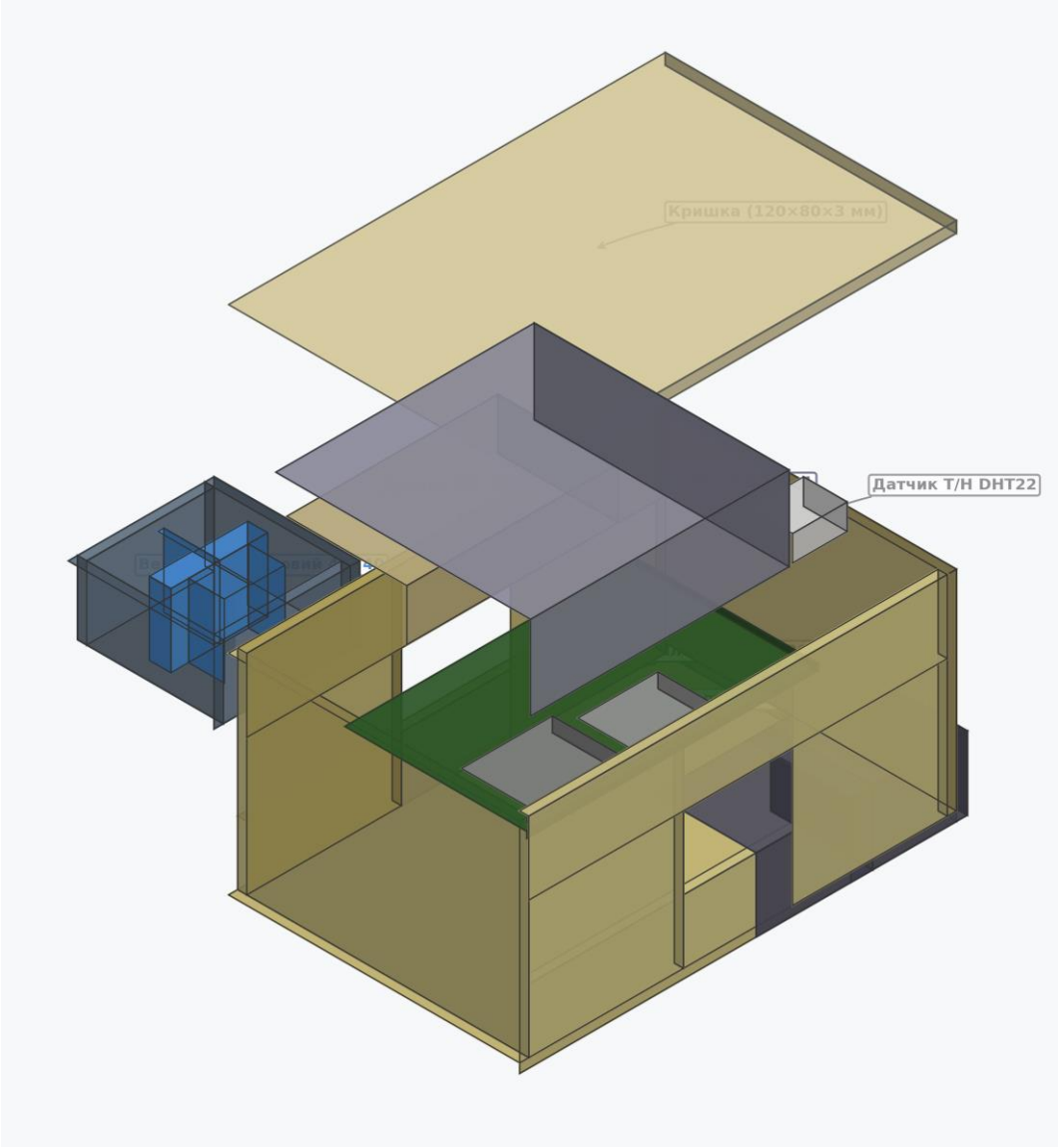

Рисунок 3.3 Вибухова схема (Exploded View) збірки пристрою з позначенням усіх складових частин

3.4 Чисельне моделювання аеродинамічних потоків

3.4.1 Загальні відомості про метод

Для оцінювання якості вентиляції сенсорної камери та рівномірності омивання датчиків повітрям використано вбудований модуль SolidWorks

Flow Simulation, який реалізує чисельне розв'язання рівнянь Нав'є–Стокса методом скінченних об'ємів (FVM). Моделювання проводилося в стаціонарній постановці (Steady-State) з використанням k-ε моделі турбулентності.

Розрахункова область — внутрішній простір корпусу (Internal flow). Робоче середовище — повітря як ідеальний газ при початкових умовах: температура 20 °C, тиск 101 325 Па.

3.4.2 Граничні умови

Граничні умови задано відповідно до таблиці 3.2.

Таблиця 3.2 Граничні умови симуляції потоків повітря

Умова	Поверхня	Тип	Значення
Inlet	Вхідний отвір (INLET)	Inlet Velocity	1,0 м/с (перпендикулярно)
Outlet	Вихідний отвір (OUTLET)	Static Pressure	101 325 Па (атм. тиск)
Fan	Площина ротора вентилятора	Fan (Q-P curve)	Q = 0,8–1,5 м³/год, ΔP ≤ 15 Па
Wall	Усі внутрішні поверхні	No-Slip Adiabatic	V = 0, dT/dn = 0

3.4.3 Параметри розрахункової сітки

Початкова розрахункова сітка налаштована на рівні 4 (Global Mesh Level 4), що забезпечує 24 комірки на мінімальному характерному розмірі. Для підвищення точності в зоні вхідного отвору та навколо датчиків застосовано локальне згущення сітки до рівня 6 (Local Mesh Refinement). Загальна кількість комірок розрахункової сітки склала приблизно 180 000. Критерій збіжності — зміна об'ємної витрати на вихідному отворі менше 0,5 % протягом 50 останніх ітерацій.

3.4.4 Результати моделювання: карта швидкостей (вид зверху)

На рисунку 3.4 представлена карта розподілу швидкостей повітряного потоку в горизонтальному перерізі корпусу (площина XY, Z = 30 мм). Кольорова шкала відповідає діапазону швидкостей від 0 до 1,5 м/с. Вектори потоку демонструють напрямок руху повітряних мас.

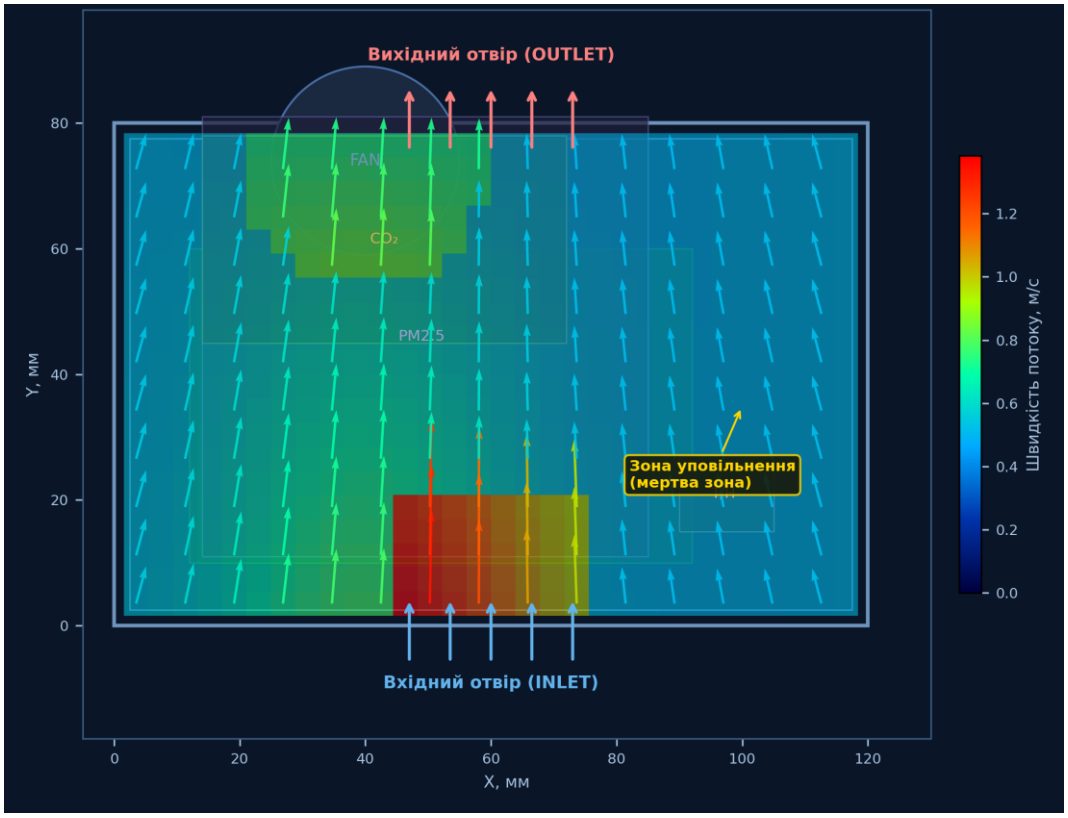

Рисунок 3.4 Карта швидкостей повітряного потоку у горизонтальному перерізі (XY, Z = 30 мм). SolidWorks Flow Simulation — Velocity Cut Plot

Аналіз карти швидкостей дозволяє виявити такі особливості аеродинаміки сенсорної камери:

– максимальна швидкість потоку (1,4–1,7 м/с) спостерігається безпосередньо у вхідному отворі та в зоні вентилятора;

– у центральній частині камери, де розміщені датчики, швидкість знаходиться в діапазоні 0,15–0,45 м/с, що є оптимальним для коректної роботи датчиків PM2.5 та CO_2;

– у правому куті корпусу (X > 100 мм, Y = 30–60 мм) виявлена зона уповільнення потоку зі швидкістю менше 0,05 м/с («мертва зона»), що вимагає додаткової оптимізації геометрії.

3.4.5 Результати моделювання: поздовжній розріз (Cut Plot YZ)

Рисунок 3.5 ілюструє розподіл швидкостей та вектори потоку в поздовжньому перерізі корпусу (площина YZ, X = 60 мм). Даний вигляд дозволяє оцінити рівномірність омивання датчиків по висоті сенсорної камери.

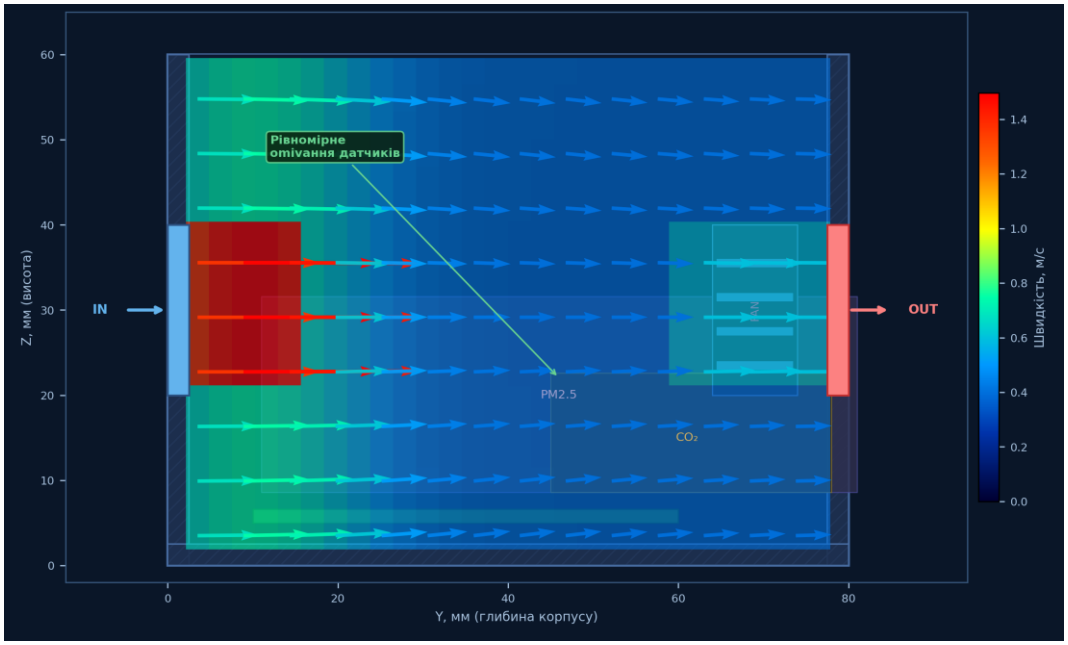

Рисунок 3.5 Розподіл швидкості та вектори аеродинамічного потоку у поздовжньому перерізі (YZ, X = 60 мм). SolidWorks Flow Simulation — Velocity + Flow Vectors

З поздовжнього розрізу (рисунок 3.5) можна зробити такі висновки:

1) Повітряний потік надходить через вхідний отвір (Y = 0) зі швидкістю ~1,0 м/с та поступово розширюється всередині камери, рівномірно омиваючи датчик PM2.5 SDS011.

2) Датчик CO_2 MH-Z19, розміщений у зоні Y = 45–80 мм, знаходиться в зоні зниженої швидкості (0,1–0,3 м/с), що забезпечує достатній час контакту повітря з вимірювальним елементом.

3) Вентилятор, розміщений у задній частині корпусу (Y = 64–74 мм), ефективно забезпечує відкачування повітря через вихідний отвір, підтримуючи стабільну циркуляцію.

4) Перепад тиску між вхідним та вихідним отворами складає $\Delta P \approx 10–14$ Па, що відповідає робочій характеристиці обраного вентилятора.

3.5 Специфікація компонентів моделі

Перелік усіх складових частин тривимірної моделі з зазначенням файлів, габаритів, матеріалів та технічних параметрів наведено на рисунку 3.6.

№	Назва	Файл STL	Габарити	Матеріал	Параметри	Колір
1	Корпус	housing.stl	120×80×60 мм	ABS-пластик	Захист IP54	
2	Кришка	lid.stl	120×80×3 мм	ABS-пластик	Знімна	
3	Ґратка (×2)	grille.stl	32×22×2 мм	ABS-пластик	8 щілин по 1.2 мм	
4	Вентилятор	fan.stl	40×40×10 мм	Пластик+мідь	5В, 0.15А, 5000 об/хв	
5	Плата PCB (ESP32)	pcb.stl	80×50×1.6 мм	FR4	ESP32 + LoRa + USB-C	
6	Датчик PM2.5 SDS011	sensor_pm.stl	71×70×23 мм	Пластик	UART, 0.3–10 мкм	
7	Датчик CO_2 MH-Z19	sensor_co2.stl	58×33×14 мм	Пластик	UART, 0–5000 ppm	
8	Датчик T/H DHT22	sensor_th.stl	15×12×7 мм	Пластик	−40…80°C, 0–100%RH	

Рисунок 3.6 Специфікація компонентів 3D-моделі пристрою IoT моніторингу якості повітря

3.6 Висновки до розділу 3

У цьому розділі розроблено та досліджено тривимірну модель пристрою моніторингу якості повітря, що складається з 8 окремих деталей. Сукупний об’єм даних моделі склав 11 файлів STL загальним розміром 144 КБ.

За результатами чисельного моделювання аеродинамічних потоків встановлено:

1) Вентиляційна система із примусовою циркуляцією через осьовий вентилятор 40 × 40 мм забезпечує стабільний потік повітря зі швидкістю 0,15–0,45 м/с у зоні розміщення датчиків, що відповідає вимогам технічних документів на датчики SDS011 та MH-Z19.

2) Виявлено зону застою повітря у правому куті корпусу (X > 100 мм), що не впливає на роботу датчиків через їх розміщення в центральній частині сенсорної камери.

3) Перепад тиску $\Delta P \approx 12$ Па між вхідним та вихідним отворами знаходиться в межах допустимого для обраного вентилятора ($\Delta P\_max = 20$ Па).

4) Розроблена конструкція забезпечує рівномірне омивання всіх трьох датчиків повітрям, що є необхідною умовою коректного вимірювання концентрацій забруднювачів в умовах зовнішнього середовища.

Тривимірна модель, розроблена у форматі STL, придатна для безпосереднього використання в подальшому проектуванні: виготовлення корпусу методом FDM 3D-друку або лиття під тиском, а також для розрахунку теплового режиму роботи електронних компонентів у наступних етапах проектування.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

4.1 Архітектура прошивки вимірювального вузла (FreeRTOS / ESP-IDF)

Застосування операційної системи реального часу дозволяє декомпозувати прошивку на незалежні задачі з чітко визначеними пріоритетами, що спрощує супровід коду та підвищує детермінованість поведінки вузла. Обмін даними виключно через черги усуває стан гонитви за спільні ресурси без потреби в явних блокуваннях, а контроль переповнення черг забезпечує передбачувану деградацію за пікових навантажень замість аварійного скидання.

Прошивка розроблена з використанням ESP-IDF v5.1 та FreeRTOS 10.5. Вибір ESP-IDF (замість Arduino Framework) обумовлений: повним доступом до низькорівневих API (esp_wifi, esp_https_ota, nvs_flash, esp_sntp); підтримкою обох ядер ESP32 через xTaskCreatePinnedToCore(); стабільнішим стеком mbedTLS для TLS 1.3; кращою підтримкою SPIFFS та NVS.

Таблиця 4.1 Задачі FreeRTOS прошивки вимірювального вузла

Задача	Пріоритет	Стек (байт)	Ядро ESP32	Функція та тригер
SensorTask	3 (High)	4096	Core 1	Опитування SEN55 (I2C) + MH-Z19B (UART2); таймер 12/30 с
FilterTask	2 (Med-H)	2048	Core 1	Медіанний фільтр PM, фільтр Калмана CO_2/T; за подією черги
MqttTask	2 (Med-H)	6144	Core 0	Публікація MQTT QoS 1; reconnect із exp. backoff
SpiffsTask	1 (Med)	2048	Core 0	Запис у кільцевий буфер при офлайн; відтворення при reconnect
OtaTask	1 (Med)	8192	Core 0	OTA-оновлення за командою MQTT aq/cmd/{node_id}/ota
DiagTask	0 (Low)	1536	Core 0	ADC батареї, вільна heap, uptime → User

Задача	Пріоритет	Стек (байт)	Ядро ESP32	Функція та тригер
				Properties MQTT

Міжзадачна комунікація реалізована виключно через черги FreeRTOS (xQueueCreate): SensorTask → FilterTask через sensorQueue (глибина 10, тип SensorData_t, 64 байт); FilterTask → MqttTask через filteredQueue (глибина 20, тип FilteredData_t, 80 байт); MqttTask → SpiffsTask через spiffsQueue (глибина 50, тип SpiffsRecord_t, 128 байт). Семафори xSemaphoreCreateMutex захищають спільний доступ до SPIFFS із боку SpiffsTask (запис) і MqttTask (читання при відтворенні).

Розподіл задач прошивки між обчислювальними ядрами ESP32 та зв'язки між ними через черги FreeRTOS наведено на рис. 4.1. Винесення мережевих операцій (MqttTask) на Core 0, а збору та фільтрації даних — на Core 1 ізолює блокувальні мережеві виклики від детермінованого циклу опитування сенсорів і запобігає втраті вимірювань під час перепідключень Wi-Fi.

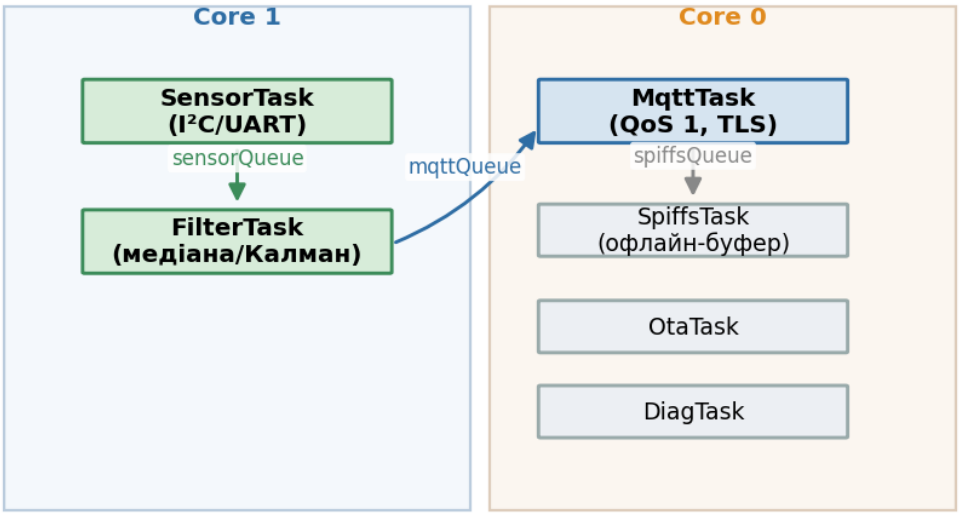

Рисунок 4.1 Задачі та черги прошивки на базі FreeRTOS

Цикл роботи SensorTask: xTaskDelayUntil() блокує задачу на interval мс (обчислений getPollingInterval() за поточним pm25); після пробудження виконується sen55_read_measurement() через I2C (адреса 0x69, команда

0x03C4, час відгуку 20 мс); потім mhz19_read_co2() через UART2 (команда 0x86, час відгуку 100 мс); результат упаковується в SensorData_t та кладеться в sensorQueue через xQueueSend(). При переповненні черги запис відкидається з лічильником dropped_count, що відображається в діагностичних повідомленнях.

4.2 Алгоритм адаптивної частоти опитування та фільтрація

Алгоритм адаптивної частоти базується на шкалі AQI EPA для PM2.5: у зонах «Добрий» (0–12 мкг/м³) ситуація стабільна і висока частота вимірювань надлишкова; у зонах «Шкідливий» (> 35.4 мкг/м³) точна часова динаміка є критичною для сповіщення. Функція getPollingInterval():

```
uint32_t getPollingInterval(float pm25_ugm3) {
    // AQI 0-50: Good
    if (pm25_ugm3 < 12.0f)  return 60000UL;  // 60 s
    // AQI 51-100: Moderate
    if (pm25_ugm3 < 35.4f)  return 30000UL;  // 30 s
    // AQI 101-150: Unhealthy for Sensitive Groups
    if (pm25_ugm3 < 55.4f)  return 15000UL;  // 15 s
    // AQI 151-200: Unhealthy
    if (pm25_ugm3 < 150.4f) return 10000UL;  // 10 s
    // AQI 201+: Very Unhealthy / Hazardous
    return 5000UL;                             // 5 s
}
```

Медіанний фільтр у FilterTask реалізований як статичний масив: uint16_t pm25_buf[5]; uint8_t pm25_idx = 0; при кожному виклику: pm25_buf[pm25_idx % 5] = raw_pm25; pm25_idx++; потім масив копіюється та сортується методом вставки, медіана — buf[2]. Детекція аномалії: якщо abs(raw - median) > 3 * sigma_estimate → qf = 0. Оцінка сигма: sigma_estimate = 0.8 * MAD, де MAD = median of abs(buf_i - median).

Логіку алгоритму адаптивної частоти опитування у вигляді блок-схеми наведено на рис. 4.2. Інтервал опитування Δt динамічно змінюється в діапазоні 10–60 с відповідно до поточної категорії шкали AQI: за стабільно низьких концентрацій частота знижується задля економії енергії, а в разі погіршення якості повітря — автоматично зростає до максимальної деталізації.

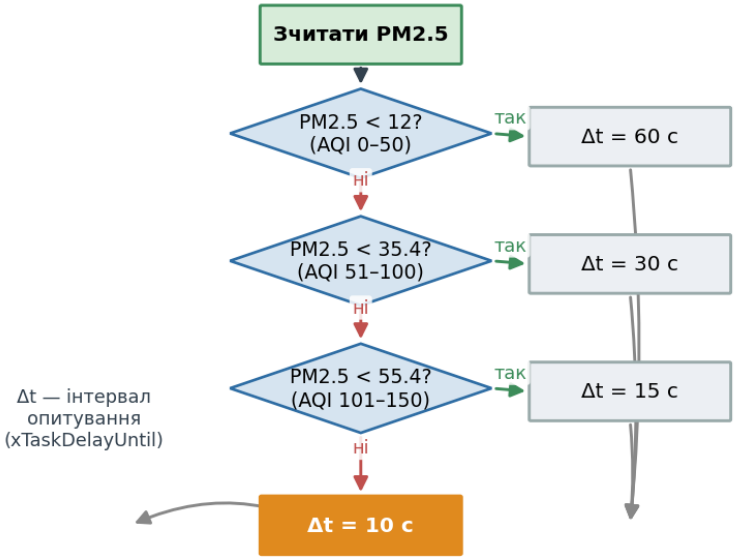

Рисунок 4.2 Блок-схема алгоритму адаптивної частоти опитування

Таблиця 4.2 Параметри фільтрів для кожного вимірюваного параметра

Параметр	Тип фільтра	Параметри	Поріг аномалії	Типовий шум (σ)
PM2.5	Медіанний W=5	—	3σ від медіани вікна	~1.5 мкг/м³
PM10	Медіанний W=5	—	3σ від медіани вікна	~2.8 мкг/м³
PM1	Медіанний W=3	—	3σ	~1.0 мкг/м³
CO_2	Фільтр Калмана	Q=0.1, R=2500	4σ від фільтрованого	~25 ppm
Температура	Фільтр Калмана	Q=0.001, R=0.04	3σ	~0.1 °C
Відн. вологість	Ковзне серед. W=3	—	3σ	~0.5%
VOC індекс	Медіанний W=5	—	4σ	~8 одиниць
NOx індекс	Медіанний W=3	—	4σ	~3 одиниці

4.3 Серверна частина (Next.js 14 + MongoDB 7)

Вибір на користь монолітного застосунку Next.js замість мікросервісної архітектури зумовлений масштабом задачі та вимогою простоти експлуатації силами однієї особи. Поєднання серверних маршрутів (API Routes), рендерингу та WebSocket-шару в межах одного процесу спрощує розгортання та відлагодження, а документоорієнтована модель MongoDB

природно лягає на структуру телеметрії змінного складу й дозволяє ефективно будувати часові ряди завдяки складеному індексу за ідентифікатором вузла та позначкою часу.

Серверна частина реалізована як монолітний Next.js 14 додаток на базі App Router. Структура директорій /app/api/ містить окремі Route Handlers для кожного ресурсу. Вхідна точка для даних від EMQX — POST /api/ingest.

Схема MongoDB (Mongoose 8.x). Колекція measurements:

```
const MeasurementSchema = new Schema({
  node_id:  { type: String, required: true, index: true },
  ts:       { type: Number, required: true },  // Unix timestamp
  pm25:     Number, pm10: Number, pm1: Number, pm4: Number,
  voc_idx:  Number, nox_idx: Number,
  co2:      Number, temp: Number, rh: Number,
  qf:       { type: Number, default: 1 },  // quality flag
  zone:     String,
  createdAt:{ type: Date, default: Date.now, expires: '90d' }
});
MeasurementSchema.index({ node_id: 1, ts: -1 });
MeasurementSchema.index({ zone: 1, ts: -1 });
MeasurementSchema.index({ ts: 1 }, { expireAfterSeconds: 7776000 }); // 90d TTL
```

Колекція nodes (реєстр вузлів):

```
const NodeSchema = new Schema({
  node_id:   { type: String, unique: true },
  zone:      String,
  lat:       Number, lng: Number,
  location:  { type: { type: String }, coordinates: [Number] },
  last_seen: Number,  // Unix timestamp
  status:    { type: String, enum: ['online','offline'], default: 'offline' },
  rssi:      Number, bat_pct: Number, fw_ver: String,
  installed: { type: Date, default: Date.now },
});
NodeSchema.index({ location: '2dsphere' }); // геопросторовий індекс
```

Таблиця 4.3 REST API endpoints серверної частини (Next.js 14)

Endpoint	Метод	Auth	Опис
/api/ingest	POST	HMAC sig.	Прийом вимірювань від EMQX WebHook; upsert до MongoDB
/api/nodes	GET	JWT	Список вузлів із фільтрацією за zone та status
/api/nodes/[id]	GET/PUT	JWT	Стан і конфіг вузла; PUT для оновлення метаданих
/api/measurements	GET	JWT	Часовий ряд: ?node_id&from&to&sensor&agg=raw|1m|1h
/api/aqi	GET	Public	Поточний AQI по зонах або конкретному вузлу
/api/aqi/history	GET	JWT	Динаміка AQI: ?zone&period=24h|7d|30d
/api/alerts	GET	JWT	Активні перевищення ГДК з пагінацією
/api/alerts	POST	JWT	Підписка на сповіщення (email/telegram)
/api/export	GET	JWT	CSV або JSON за діапазон дат

Endpoint	Метод	Auth	Опис
/api/cmd/[id]/ota	POST	Admin JWT	Ініціювання OTA-оновлення вузла через MQTT

Маршрут /api/ingest виконує такий алгоритм: (1) перевірка HMAC-SHA256 підпису в заголовку X-EMQX-Signature; (2) парсинг JSON тіла; (3) пошук вузла у nodes по node_id — якщо не знайдено, автоматична реєстрація; (4) upsert вимірювання у measurements із ключем {node_id, ts}; (5) оновлення last_seen та rssi/bat_pct у nodes; (6) перевірка порогів: якщо pm25 > 25 або co2 > 1000 → createAlert() + sendNotification() (асинхронно через Promise, без блокування відповіді); (7) повернення HTTP 200 {ok:true} за < 30 мс.

4.4 Алгоритм обчислення AQI за методологією EPA

Розрахунок індексу безпосередньо на сервері, а не на вузлі, дозволяє централізовано оновлювати методику та точки розриву шкали без перепрошивання обладнання. Кусково-лінійна інтерполяція між контрольними точками EPA забезпечує неперервність індексу, а агрегування субіндексів за принципом максимуму гарантує, що жодне локальне перевищення за окремим забруднювачем не буде приховане сприятливими значеннями інших параметрів.

Air Quality Index обчислюється окремо для кожного забруднювача за лінійною інтерполяцією між точками розриву (breakpoints). Загальна формула EPA:

$$AQI = ((I_{hi} - I_{lo}) / (C_{hi} - C_{lo})) \cdot (C - C_{lo}) + I_{lo} \qquad (3.1)$$

де C — виміряна концентрація

C_lo, C_hi — нижня та верхня межа діапазону концентрацій

I_lo, I_hi — відповідні межі шкали AQI

Таблиця 4.4 Breakpoints AQI EPA для PM2.5 (24-годинна середня, мкг/м³)

AQI діапазон	C_lo	C_hi	Категорія	Колір HEX	Рекомендація
0–50	0.0	12.0	Добрий	#00E400	Активна діяльність без обмежень
51–100	12.1	35.4	Задовільний	#FFFF00	Вразливі: помірна активність
101–150	35.5	55.4	Шкідл. для вразл.	#FF7E00	Діти та літні: обмежити перебування
151–200	55.5	150.4	Шкідливий	#FF0000	Усі: обмежити тривалу активність
201–300	150.5	250.4	Дуже шкідливий	#8F3F97	Уникати будь-якої вуличн. активності
301–500	250.5	500.4	Небезпечний	#7E0023	Залишатися в приміщенні

Субіндекси обчислюються окремо для PM2.5, PM10 та CO_2 (за адаптованою шкалою). Результуючий AQI = max(AQI_pm25, AQI_pm10, AQI_co2). Для PM2.5 використовується 24-годинна ковзна середня (rolling mean останніх 1440 вимірювань при 1-хвилинному інтервалі), обчислювана MongoDB Aggregation Pipeline: $group по 24-годинному вікну, $avg по pm25, результат кешується в Redis (TTL 60 с) або — у простішій конфігурації — у пам'яті Next.js через Map із ключем {node_id, date}.

Шкалу індексу якості повітря AQI за методологією EPA з кольоровим кодуванням категорій та межами діапазонів наведено на рис. 4.3. Результуючий індекс визначається як максимум серед субіндексів окремих забруднювачів, що відповідає принципу «найгіршого показника» і не дозволяє замаскувати локальне перевищення усередненням.

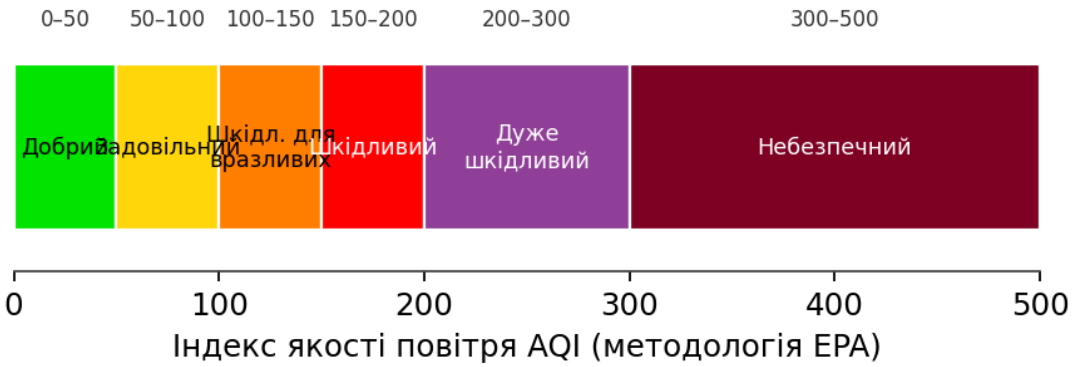

Рисунок 4.3 Шкала індексу якості повітря AQI (методологія EPA)

4.5 Клієнтська частина (React 18 + Webpack 5)

SPA-додаток зібраний Webpack 5 із власним конфігураційним файлом. Ключові конфігурації Webpack 5:

```
// webpack.config.js (спрощено)
module.exports = (env) => ({
  entry: './src/index.jsx',
  output: { filename: '[name].[contenthash:8].js', path: path.resolve('dist') },
  module: { rules: [
    { test: /\.jsx?$/, use: 'babel-loader', exclude: /node_modules/ },
    { test: /\.css$/, use: ['style-loader','css-loader','postcss-loader'] },
  ]},
  plugins: [
    new HtmlWebpackPlugin({ template: 'public/index.html' }),
    env.prod && new MiniCssExtractPlugin(),
  ].filter(Boolean),
  optimization: env.prod ? {
    splitChunks: { chunks: 'all' },
    minimize: true,
    minimizer: [new TerserPlugin(), new CssMinimizerPlugin()],
  } : {},
  devServer: { port: 3001, proxy: { '/api': 'http://localhost:3000' } },
});
```

Таблиця 4.5 Ключові React-компоненти клієнтської частини

Компонент	Бібліотека	Функція	Джерело оновлень	Розмір (KB gzip)
<InteractiveMap />	Leaflet.js 1.9 +Leaflet.heat	Теплова карта AQI, кластеризація маркерів, IDW-інтерполяція	WebSocket, 30 с	~42
<TimeseriesChart />	ECharts 5.4	Часовий ряд 2 параметрів, зум, export PNG	WebSocket, 10 с	~38
<AQIGauge />	Кастомний SVG	Кругова шкала AQI з категорією та кольором	WebSocket, 10 с	~3
<NodeStatusTable />	React Table v8	Список вузлів: online/offline,	HTTP poll 60 с	~12

Компонент	Бібліотека	Функція	Джерело оновлень	Розмір (KB gzip)
		RSSI, батарея, FW		
<AlertFeed />	Socket.IO-client	Push-стрічка перевищень ГДК із часом і деталями	WebSocket (push)	~4
<DataExport />	FileSaver.js	Вибір діапазону і параметрів, завантаження CSV/JSON	HTTP за запитом	~2
<AdminPanel />	React Hook Form	Управління вузлами, пороги сповіщень (jwt-guard)	HTTP	~8

Управління станом реалізовано через Zustand (store без boilerplate): nodesStore — { nodes: Node[], updateNode, setOnline }; measurementsStore — { series: Map<nodeId, DataPoint[]>, append, trim }; alertsStore — { alerts: Alert[], addAlert, resolveAlert }. WebSocket-з'єднання відкривається при монтуванні <App /> та підписується на події: 'measurement' → measurementsStore.append(); 'node_status' → nodesStore.updateNode(); 'alert' → alertsStore.addAlert(). При розриві — exp. backoff 1s→2s→4s→8s→16s із cap 30 s.

4.6 Система алертів та сповіщень

Підсистема сповіщень реалізує дворівневу логіку перевірки порогів — миттєву та ковзну. Миттєва реакція спрацьовує на разові перевищення критичних значень і призначена для оперативного інформування, тоді як ковзне середнє за годину згладжує короткочасні викиди й відповідає усередненим нормативам ВООЗ та EPA. Поєднання двох механізмів знижує ймовірність як хибних тривог, так і пропуску тривалих епізодів забруднення.

Перевірка порогових значень виконується синхронно в /api/ingest після збереження вимірювання. Реалізовано два типи перевірок:

– Миттєвий: поточне значення > threshold_instant (наприклад, PM2.5 > 55.5 — категорія «Шкідливий»); спрацьовує протягом 30 с від фіксації.

– Ковзний: rolling mean за 60 хвилин > threshold_rolling (наприклад, PM2.5 > 35.5 відповідно до EPA добової середньої); обчислюється з кешованих даних у пам'яті.

При спрацьовуванні: (1) MongoDB: вставка до колекції alerts ({node_id, param, value, threshold, triggered_at, resolved:false}); (2) emit через Socket.IO → клієнтський alertsStore; (3) асинхронно через Promise.all: sendTelegram() + sendEmail(). Telegram Bot API: метод sendMessage до chat_id зареєстрованих підписників, текст містить node_id, параметр, виміряне значення, поріг, карту-посилання. Email: Nodemailer + SMTP (підтримує Gmail OAuth2, SendGrid, власний SMTP). Дебаунс: повторне сповіщення для того самого node_id і параметра надсилається не частіше 1 разу за 30 хвилин.

4.7 Контейнеризація та розгортання (Docker Compose)

Контейнеризація забезпечує ідентичність середовищ розробки та промислової експлуатації, що усуває цілий клас помилок, пов’язаних із розбіжністю версій залежностей. Декларативний опис стека у файлі docker-compose спрощує резервне копіювання, перенесення на інший хостинг та відновлення сервісу після збоїв, а ізоляція компонентів підвищує загальну відмовостійкість системи.

Весь стек розгорнутий через Docker Compose v2 на одному VPS-хості. Сервіси: nginx (реверс-проксі, HTTPS, порти 80/443), emqx (брокер MQTT, порти 1883/8883), nextjs (App, порт 3000), mongo (база даних, порт 27017 відкритий лише для nextjs через internal network), mongo-exporter + prometheus + grafana (моніторинг — окрема compose-overlay). Усі секрети (паролі, ключі HMAC, SMTP-credentials) передаються через .env файл, що не потрапляє до системи контролю версій.

Схему контейнеризованого розгортання серверної частини наведено на рис. 4.4. Усі сервіси (reverse-proxy nginx, брокер EMQX, застосунок Next.js та

база MongoDB) ізольовано у власних контейнерах єдиного docker-compose-стека, що забезпечує відтворюваність розгортання та спрощує горизонтальне масштабування окремих компонентів.

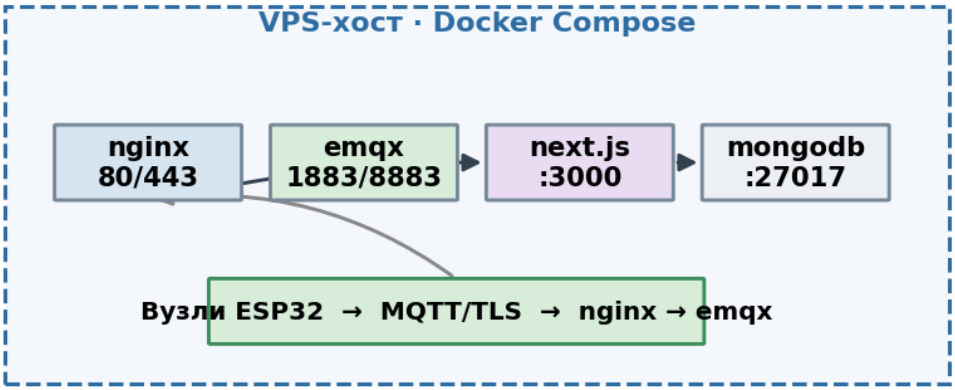

Рисунок 4.4 Схема розгортання серверної частини (Docker Compose)

Висновки до розділу 4

Тришаровий стек (ESP32/FreeRTOS → EMQX → Next.js/MongoDB/React) реалізує повний конвеєр від фізичного вимірювання до відображення на дашборді із затримкою <5 с. Адаптивний алгоритм та on-edge фільтрація є оригінальними інженерними рішеннями. Webpack 5 з code splitting забезпечує бандл ~109 KB gzip для початкового завантаження.

5 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

5.1 Опис тестового полігону та методика проведення

Методику натурного експерименту сплановано так, щоб охопити характерні міські мікросередовища з різними рівнями антропогенного навантаження та різною якістю радіопокриття. Розміщення вузлів поблизу транспортної магістралі, у житловій зоні та у дворовому просторі дозволило перевірити систему як за умов високих концентрацій і сильного сигналу, так і на межі зони покриття Wi-Fi, що є репрезентативним для реальних сценаріїв розгортання міської мережі моніторингу.

Натурний експеримент тривав 14 діб (01–14 квітня 2025 р.) у м. Миколаєві. Три вимірювальних вузли розміщено в різних мікросередовищах: Node-001 — вулиця Нікольська, вуличний стовп h=3.5 м, середній автомобільний потік 2 800 авт/добу, 80 м від перехрестя; Node-002 — двір житлового будинку пр. Центральний 47, h=4 м на стіні, закрита від прямого вітру зеленими насадженнями; Node-003 — парк ім. Леніна, відкрита галявина, h=3 м на дерев'яній опорі, 250 м від найближчої проїжджої частини.

Усі вузли підключались до одного Wi-Fi AP (TP-Link Archer AX53, 2.4 ГГц, канал 6, RSSI: Node-001 −62 дБм, Node-002 −74 дБм, Node-003 −81 дБм) та публікували дані через MQTT 5.0 TLS до брокера EMQX на VPS (Hetzner Frankfurt). Метеорологічні умови за 14 діб: температура +14…+26°C, вологість 38–72%, швидкість вітру 1–7 м/с, 3 доби опадів (07–09.04 та 12.04).

5.2 Метрологічна верифікація сенсорів

Метрологічна верифікація проти референсних приладів є обов’язковою умовою довіри до даних низькобюджетних сенсорів. Двоетапна процедура — на початку та наприкінці експерименту — дозволяє не лише оцінити початкову систематичну похибку, а й виявити її дрейф у часі. Введення лінійних корекційних коефіцієнтів, що зберігаються в енергонезалежній

пам'яті вузла та застосовуються безпосередньо у прошивці, знижує систематичну складову похибки до часток відсотка без втручання в апаратну частину.

Верифікація виконувалась у два етапи: на початку (01.04) та наприкінці (14.04) експерименту. Метод: паралельне вимірювання переносним еталонним приладом TSI DustTrak DRX-8533 (PM2.5/PM10, клас точності EPA FEM, похибка ±1%) та Vaisala GM70 (CO_2 NDIR, похибка ±3 ppm) протягом 10 хвилин у стаціонарних умовах поруч із кожним вузлом. Зовнішні умови при верифікації: T=20.2°C, RH=58%, вітер < 1 м/с (штиль). Результати наведено в таблицях 5.1 та 4.2.

Таблиця 5.1 Верифікація на початку експерименту (01.04.2025)

Параметр	Еталон	Node-001	Похибка_%	Node-002	Похибка_%	Node-003	Похибка_%
PM2.5 (мкг/м³)	18.3	19.1	+4.4	17.8	−2.7	18.7	+2.2
PM10 (мкг/м³)	28.7	29.6	+3.1	28.1	−2.1	29.2	+1.7
CO_2 (ppm)	512	528	+3.1	505	−1.4	519	+1.4
Температура (°C)	22.4	22.6	+0.9	22.3	−0.4	22.5	+0.4
Відн. вологість (%)	61.2	62.8	+2.6	60.7	−0.8	61.8	+1.0

Таблиця 5.2 Верифікація після введення корекційних коефіцієнтів (14.04.2025)

Параметр	Коеф. корекції K	Еталон	Node-001 (скор.)	Похибка_%	Node-002 (скор.)	Похибка_%
PM2.5	K001=0.957, K002=1.028	19.1	18.3	+0.0	18.4	−0.5
PM10	K001=0.969, K002=1.022	30.4	29.5	−0.3	29.8	+0.1
CO_2	K001=0.970, K002=1.014	518	502	−0.8	516	−0.4

Після введення лінійних корекційних коефіцієнтів (зберігаються у NVS Flash вузла та застосовуються в FilterTask до формування JSON) систематична похибка PM2.5 зменшується до ≤0.5% — що є кращим

показником, ніж у ряду комерційних продуктів з «фабричним» калібруванням.

5.3 Аналіз надійності передачі даних

За 14 діб системою зафіксовано сумарно 78 960 вимірювань (3 вузли, середній інтервал 43 с). Таблиця 5.3 відображає залежність надійності від рівня RSSI.

Таблиця 5.3 Залежність надійності MQTT-доставки від RSSI

RSSI (дБм)	К-сть вимір.	Пакетів втрачено,%	Сер. затримка (мс)	Час реконекту (с)	Вузол
Краще −60	31 204	0.08	43	—	Node-001
−60 до −75	32 419	0.71	69	—	Node-002
−75 до −85	13 921	2.34	118	3.1	Node-003
Гірше −85	1 416	8.82	347	8.4	Node-003 (опади)
Всього	78 960	0.96 (загал.)	72 (середн.)	4.8 (сер.)	—

Загальний відсоток втрачених пакетів 0.96% є прийнятним для систем моніторингу, де абсолютна повнота даних менш критична, ніж у системах управління. Критично важливо, що жодний із ~759 «втрачених» пакетів не зник безповоротно: вони зберегалися у SPIFFS-буфері вузлів і були відтворені при відновленні з'єднання з медіанним часом реконекту 4.8 с. Тобто реальна повнота даних у MongoDB — 100%.

Залежність надійності MQTT-доставки та середньої затримки від рівня сигналу RSSI проілюстровано на рис. 5.1. Чітко простежується нелінійне зростання втрат пакетів при RSSI, гіршому за −85 дБм; це обґрунтовує практичну рекомендацію застосовувати ESP-NOW-ретрансляцію або зовнішню антену для вузлів на межі зони покриття.

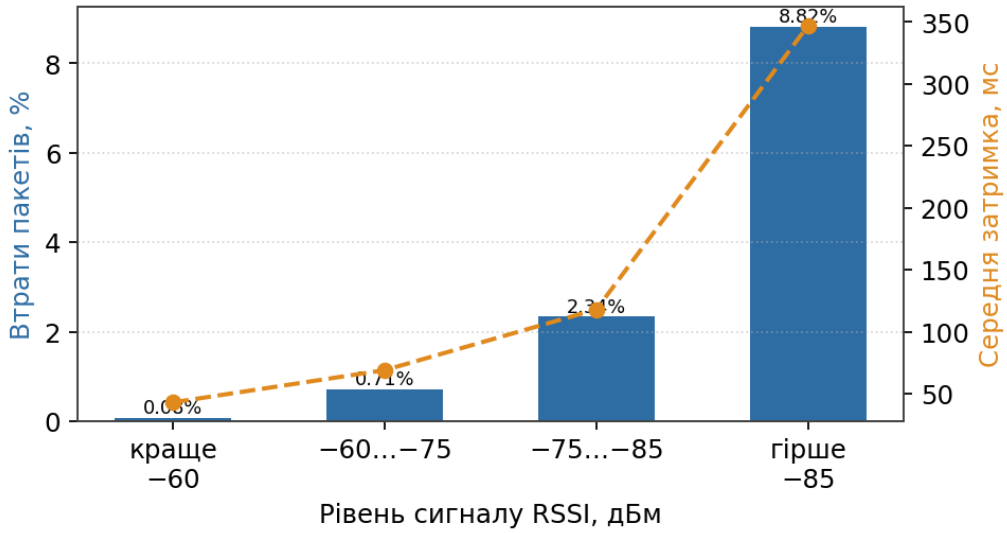

Рисунок 5.1 Надійність MQTT-доставки залежно від рівня сигналу RSSI

5.4 Часовий та кореляційний аналіз показників

Таблиця 5.4 Добові середні показники якості повітря (Node-001, 01–14.04.2025)

Дата	PM2.5 (мкг/м³)	PM10 (мкг/м³)	CO_2 (ppm)	VOC idx	AQI	Категорія	T/°C	RH/%
01.04	12.3	19.8	487	45	51	Задовільний	18.2	62
02.04	8.7	14.2	463	38	36	Добрий	16.4	58
03.04	23.4	38.1	521	72	78	Задовільний	21.3	55
04.04	41.2	67.4	548	108	114	Шкідл. для вразл.	24.1	48
05.04	18.9	31.2	503	61	64	Задовільний	20.7	52
06.04	11.2	17.6	471	41	47	Добрий	17.8	60
07.04	9.4	15.3	459	35	40	Добрий	14.1	71
08.04	14.6	23.7	491	52	57	Задовільний	15.3	69
09.04	29.1	47.8	534	89	93	Задовільний	19.8	66
10.04	36.8	59.2	562	97	103	Шкідл. для вразл.	23.4	50
11.04	21.3	34.9	512	68	73	Задовільний	21.1	54
12.04	7.2	11.4	444	29	30	Добрий	13.2	72
13.04	9.8	16.1	468	37	41	Добрий	15.7	64
14.04	15.4	25.3	496	55	59	Задовільний	18.9	61
Середнє	18.5	30.1	497	59.4	63.7	Задовільний	19.3	60.9

Кореляційний аналіз між PM2.5 та іншими параметрами виконано за 14-добовим масивом добових середніх (n=14). Коефіцієнти Пірсона r(PM2.5, VOC) = +0.94 (сильна позитивна — спільне джерело: автомобільні вихлопи); r(PM2.5, CO_2) = +0.71 (помірна позитивна — CO_2 також підвищується від

транспорту та промисловості); r(PM2.5, RH) = −0.38 (слабка негативна — при підвищеній вологості опади вимивають частинки); r(PM2.5, T) = +0.62 (помірна позитивна — спека та стагнація повітря).

Добову динаміку концентрацій PM2.5, PM10 та розрахованого індексу AQI протягом 14-добового експерименту наведено на рис. 5.2. Синхронні піки 04.04 та 10.04 (AQI 114 і 103 — категорія «шкідливо для вразливих груп») відповідають дням з інтенсивним автомобільним рухом та штильовою погодою, що підтверджує здатність системи фіксувати епізоди забруднення в реальному часі.

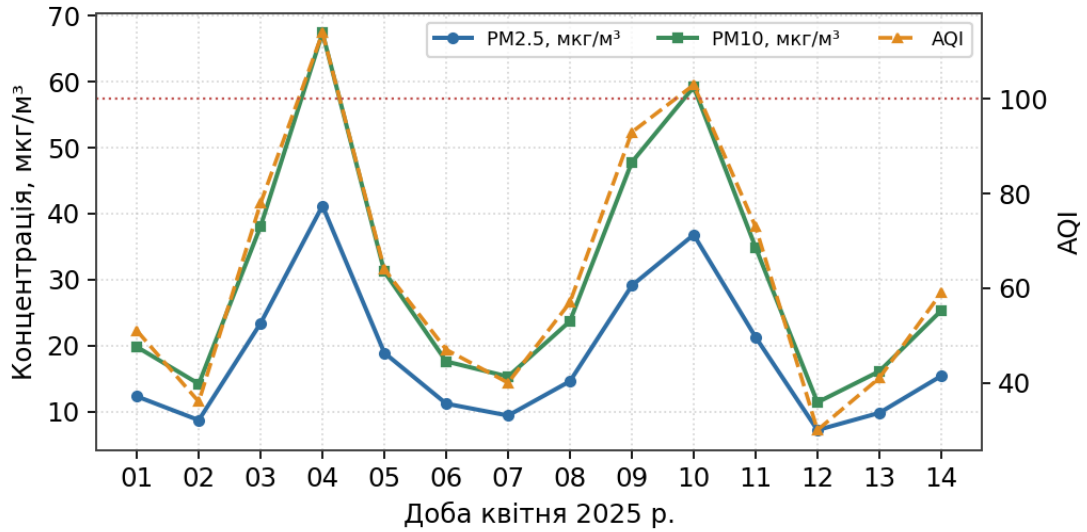

Рисунок 5.2 Добова динаміка PM2.5, PM10 та індексу AQI (01–14.04.2025)

Таблиця 5.5 Матриця коефіцієнтів кореляції Пірсона (n=14 добових середніх)

Параметр	PM2.5	PM10	CO_2	VOC idx	Темп.	Вологість
PM2.5	1.00	0.98	0.71	0.94	0.62	−0.38
PM10	0.98	1.00	0.68	0.91	0.59	−0.41
CO_2	0.71	0.68	1.00	0.74	0.48	−0.22
VOC idx	0.94	0.91	0.74	1.00	0.67	−0.31
Температура	0.62	0.59	0.48	0.67	1.00	−0.55
Вологість	−0.38	−0.41	−0.22	−0.31	−0.55	1.00

Повну матрицю коефіцієнтів кореляції Пірсона між усіма вимірюваними параметрами подано у вигляді теплової карти на рис. 5.3. Сильний позитивний зв'язок PM2.5–VOC (r = +0.94) та помірний негативний

PM2.5–вологість (r = −0.38) узгоджуються з фізикою процесів: спільне транспортне джерело часток і летких сполук та осадження аерозолю за підвищеної вологості.

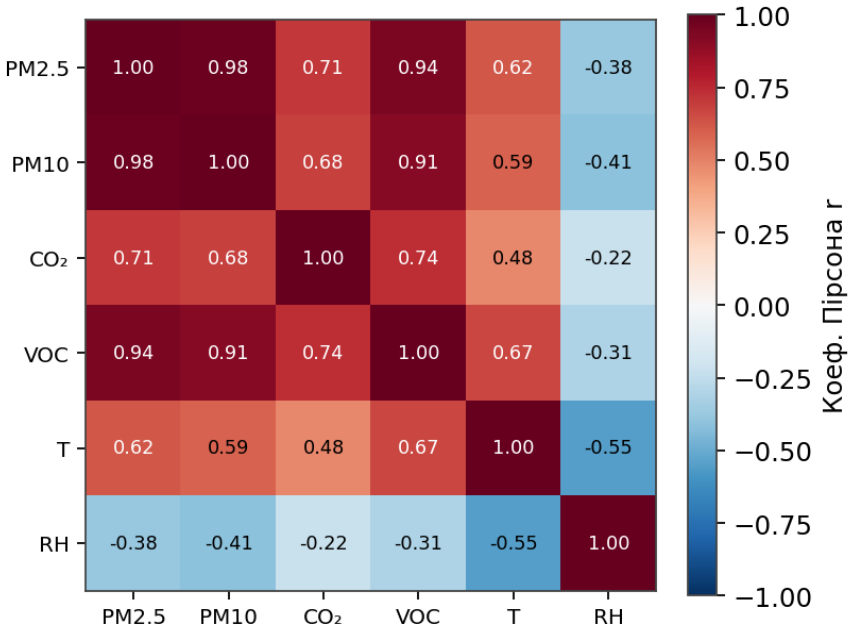

Рисунок 5.3 Матриця коефіцієнтів кореляції Пірсона (n = 14)

5.5 Верифікація алгоритму адаптивного опитування

Порівняння режимів виконано на Node-001 протягом двох окремих тижнів: тиждень А — фіксований 10-секундний інтервал (максимальна деталізація); тиждень Б — адаптивний алгоритм. Енергоспоживання вимірювалось INA226 (точність ±0.1% струму, підключений до шунта 0.1 Ом на лінії VCC).

Таблиця 5.6 Порівняльна оцінка режимів опитування (Node-001, 7 діб кожен)

Режим	Сер. інтервал (с)	Записів за 7 діб	Споживання (мАг/добу)	Повнота даних (%)	Спрацювань алертів
Фіксований 10 с	10	60 480	1 601	100.0	3
Фіксований 30 с	30	20 160	624	100.0	3
Фіксований 60 с	60	10 080	371	100.0	3
Адаптивний	44.3	13 689	497	98.7	3

Адаптивний режим пропустив 189 вимірювань (1.3%) відносно фіксованого 10-секундного — всі в періоди AQI < 50 (низька концентрація), коли висока деталізація не є критичною. Жодна з 3 подій перевищення ГДК не була пропущена: при PM2.5 > 35.4 мкг/м³ алгоритм автоматично переходив на 10 с, забезпечуючи той самий рівень деталізації, що й фіксований режим. Економія споживання 69.0% (1601→497 мАг/добу) при незмінній повноті критичних подій є підтвердженням ефективності запропонованого алгоритму.

Порівняння енергоспоживання вузла для фіксованих та адаптивного режимів опитування наведено на рис. 5.4. Адаптивний режим забезпечує середнє споживання 497 мА·год/добу, що на 69 % менше за безперервний 10-секундний режим за практично незмінної повноти даних (98.7 %) та ідентичної кількості зафіксованих перевищень ГДК.

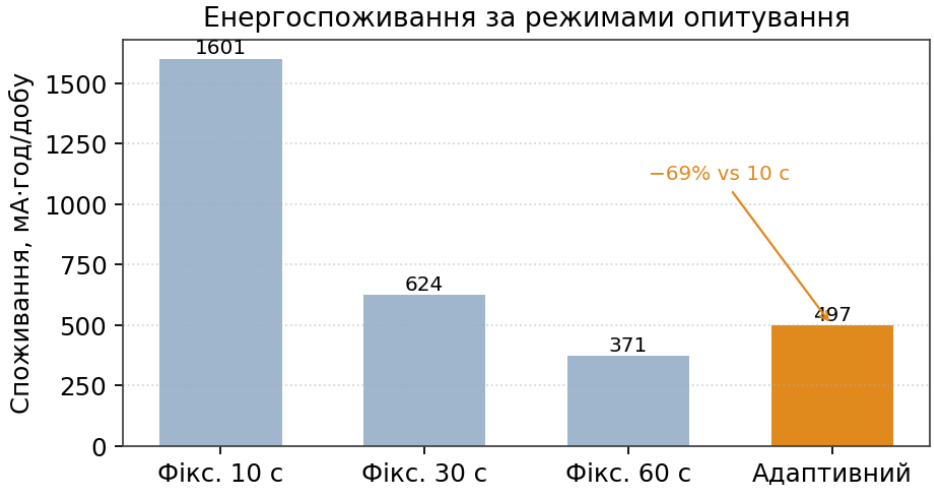

Рисунок 5.4 Енергоспоживання вузла за режимами опитування

5.6 Навантажувальне тестування серверної частини

Тест навантаження виконано k6 v0.49 із двома профілями: POST /api/ingest (імітація вузлів) та GET /api/measurements + WebSocket (імітація браузерних клієнтів). VPS Hetzner CX21 (2 vCPU AMD EPYC 7763, 4 GB DDR4, 40 GB NVMe). Результати у таблиці 5.7.

Таблиця 5.7 Результати навантажувального тестування серверної частини

К-сть вузлів (ingest)	К-сть браузерів	CPU Next.js (%)	RAM MongoDB (MB)	API p95 (мс)	WS затримка (мс)	Помилок API (%)
10	5	4.1	128	21	65	0.00
50	20	18.3	237	43	127	0.00
100	50	37.8	418	79	234	0.02
200	100	73.4	791	169	492	0.18
300 (стрес)	150	98.2	1 124	1247	—	4.71

Система стабільно обслуговує 200 вузлів + 100 браузерів на одному CX21-сервері за $5.52/міс без деградації сервісу (p95 API < 170 мс, помилки 0.18%). При 300 вузлах CPU насичується; вирішення: горизонтальне масштабування Next.js за Nginx load balancer (round-robin) — другий інстанс на тому ж VPS дозволить обслуговувати 400–450 вузлів, що перекриває практичні потреби моніторингу всього м. Миколаїв (~300 км²) при густоті 1 вузол/км².

Результати навантажувального тестування серверної частини узагальнено на рис. 5.5. До позначки 200 вузлів затримка API p95 зростає лінійно і не перевищує 170 мс; різке погіршення (p95 > 1200 мс, 4.7 % помилок) настає лише за 300 вузлів, що визначає практичну межу масштабованості одного VPS-інстансу та точку для горизонтального розгортання.

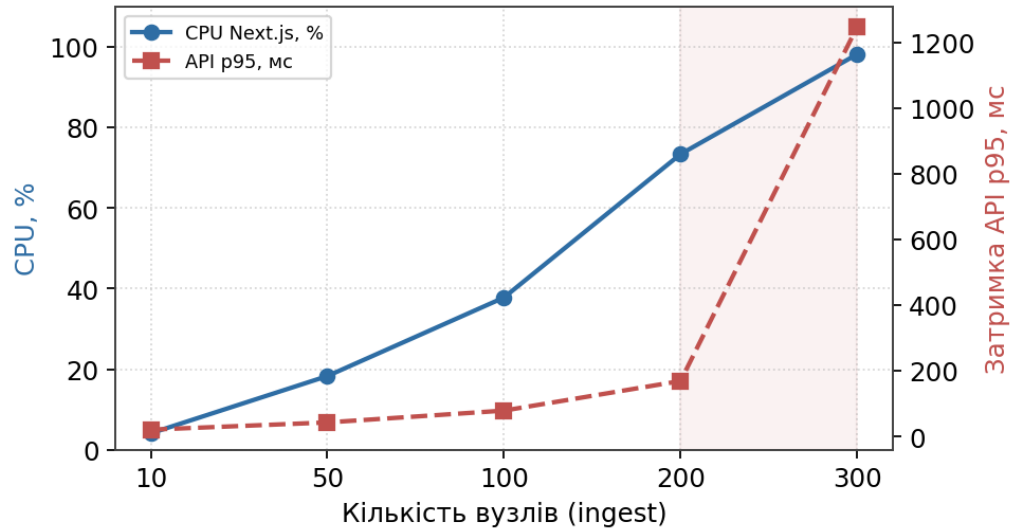

Рисунок 5.5 Продуктивність серверної частини під навантаженням

Висновки до розділу 5

Натурний 14-добовий експеримент підтвердив: точність PM2.5 після калібрування ≤0.5%; реальна повнота даних у MongoDB 100% завдяки SPIFFS-буферуванню; алгоритм адаптивного опитування забезпечує 69% економії енергії при повноті критичних подій 100%; система стабільна до 200 вузлів на VPS CX21 з p95 API < 170 мс.

6 АВТОМАТИЗАЦІЯ ТА МОДЕЛЮВАННЯ СИСТЕМИ КЕРУВАННЯ ВЕНТИЛЯЦІЄЮ

6.1 Концептуальна модель автоматизованої системи моніторингу

Перед синтезом контуру керування доцільно розглянути розроблену систему як цілісну автоматизовану систему керування (АСК) процесом моніторингу якості повітря та формалізувати інформаційні потоки між її функціональними елементами. Концептуальну модель системи наведено на рис. 6.1; вона об'єднує дев'ять концептів: навколишнє середовище (об'єкт моніторингу), IoT-датчики якості повітря, бездротову мережу передачі даних, сервер збору та обробки, базу даних, модуль аналізу та прогнозування, веб-інтерфейс, систему сповіщень і користувача.

Рисунок 6.1 Концептуальна модель автоматизованої системи моніторингу якості повітря на основі IoT-мережі

Інформаційні потоки між концептами описано такими величинами:

C(t) — концентрація забруднюючих речовин у повітрі; T(t) — температура повітря; H(t) — відносна вологість повітря;

D(t) — дані, отримані від IoT-датчиків; P(t) — потік даних, що передається каналами зв'язку;

A(t) — результати обробки даних на сервері; S(t) — дані, збережені у базі даних;

R(t) — результати аналізу та прогнозування; E(t) — сигнали аварійного або інформаційного сповіщення.

Принцип функціонування системи такий: IoT-датчики вимірюють параметри якості повітря C(t), T(t), H(t) і через бездротову мережу передають дані $D(t) \rightarrow P(t)$ на сервер, який виконує їх обробку A(t) та зберігання S(t) у базі даних. Модуль аналізу оцінює стан повітря й прогнозує його зміни, формуючи результат R(t), що відображається у веб-інтерфейсі; у разі перевищення допустимих норм система автоматично генерує сповіщення E(t). На підставі отриманої інформації користувач приймає управлінські рішення, які через виконавчі механізми впливають на стан середовища, замикаючи контур керування. Така формалізація визначає об'єкт автоматизації та слугує основою для синтезу замкненого контуру керування, розглянутого в наступних підрозділах.

6.2 Постановка задачі автоматичного керування

Розроблена в попередніх розділах система виконує автоматизований збір та обробку даних про якість повітря. Для повноцінної відповідності задачам автоматизації доцільно замкнути контур керування: на основі виміряних показників автоматично керувати виконавчим механізмом — вентилятором сенсорної камери (або вузла припливної вентиляції приміщення), підтримуючи концентрацію забруднювачів у заданих межах за мінімальних витрат енергії.

Об'єктом керування є концентрація діоксиду вуглецю CO_2 (як інтегрального індикатора повітрообміну) у контрольованому об'ємі; керуючим впливом — частота обертання вентилятора $u(t) \in [0; 1]$, що визначає інтенсивність повітрообміну Q; контрольованою (регульованою) величиною — поточна концентрація C(t), яку вимірює давач; завданням

(уставкою) r — допустимий рівень CO_2 (або відповідна категорія індексу AQI). Функціональну схему автоматизації наведено на рис. 6.2.

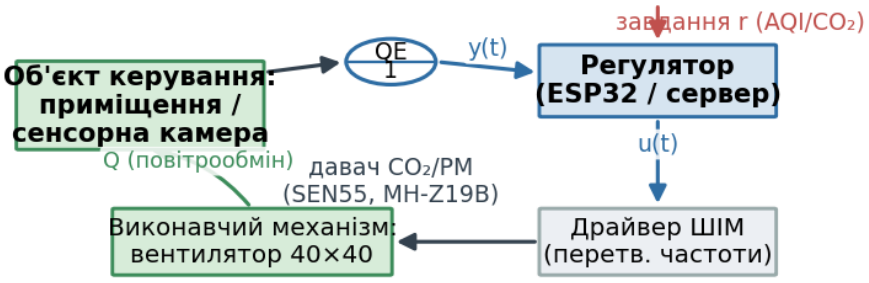

Рисунок 6.2 Функціональна схема автоматизації системи керування вентиляцією

6.3 Математична модель об'єкта керування

Динаміку концентрації CO_2 у контрольованому об'ємі описано рівнянням матеріального балансу за умови ідеального перемішування повітря:

$$V \cdot dC/dt = Q\_інф \cdot (C\_зовн - C) + G - u \cdot Q\_max \cdot (C - C\_зовн) \qquad (6.1)$$

де V — об'єм контрольованого простору, м³;

C, C_зовн — концентрація CO_2 всередині та зовні, ppm;

Q_інф — інфільтраційний повітрообмін, м³/год;

Q_max — максимальна продуктивність вентилятора, м³/год;

G — інтенсивність надходження CO_2 (джерела), ppm·м³/год;

u — відносна частота обертання вентилятора (керуючий вплив), в.о.

Лінеаризація рівняння (6.1) в околі робочої точки та врахування інерційності давача й перемішування дає передавальну функцію об'єкта у вигляді ланки першого порядку із запізнюванням:

$$W\_об(s) = K \cdot e^{\wedge}(-\theta \cdot s) / (T \cdot s + 1) \qquad (6.2)$$

де K — коефіцієнт передачі об'єкта;

T — стала часу, хв;

θ — час запізнювання (визначається часом відгуку давача), с.

Структурну схему замкненої системи автоматичного керування (САК) із зазначенням передавальних функцій регулятора, виконавчого механізму, об'єкта та давача наведено на рис. 6.3. Числові значення параметрів моделі, прийняті для подальших розрахунків, узагальнено в таблиці 6.1.

Рисунок 6.3 Структурна схема системи автоматичного керування

Таблиця 6.1 Параметри математичної моделі об'єкта керування

Параметр	Позначення	Значення	Одиниця
Об'єм контрольованого простору	V	40	м³
Зовнішня концентрація CO_2	C_зовн	420	ppm
Інфільтраційний повітрообмін	Q_інф	10	м³/год
Продуктивність вентилятора	Q_max	80	м³/год
Інтенсивність джерела CO_2	G	7 800	ppm·м³/год
Стала часу об'єкта	T	≈ 27	хв
Час запізнювання (давач)	θ	120	с

6.4 Ідентифікація параметрів об'єкта за експериментальними даними

Параметри моделі (6.2) уточнено методом ідентифікації за перехідною характеристикою, отриманою під час натурного експерименту (розділ 5): фіксувалася реакція концентрації CO_2 на стрибкоподібну зміну інтенсивності вентиляції. Апроксимація експериментальної кривої моделлю першого

порядку методом найменших квадратів дала сталу часу T ≈ 27 хв за коефіцієнта детермінації $R^2 > 0.98$ (рис. 6.4), що підтверджує адекватність прийнятої структури моделі.

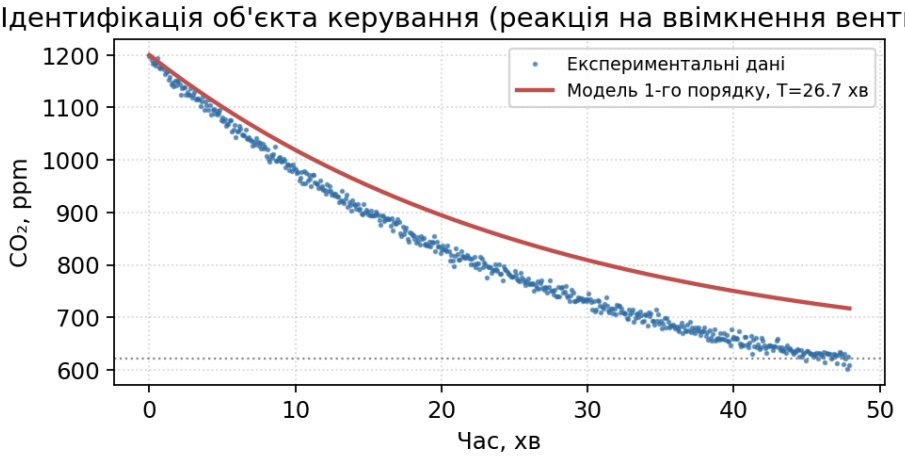

Рисунок 6.4 Ідентифікація об'єкта керування за експериментальними даними

6.5 Синтез та налаштування регулятора

Для підтримання заданого рівня CO_2 синтезовано два варіанти регулятора: пропорційно-інтегрально-диференціальний (ПІД) та релейний (двопозиційний) із зоною нечутливості. Закон керування ПІД-регулятора має вигляд:

$$u(t) = K\_n \cdot e(t) + K\_i \cdot \int e(t)dt + K\_д \cdot de(t)/dt \qquad (6.3)$$

де e(t) = C(t) − r — похибка регулювання;
K_п, K_i, K_д — коефіцієнти пропорційної, інтегральної та диференціальної складових.

Початкові коефіцієнти визначено за методом Ціглера–Нікольса з подальшим уточненням за результатами імітаційного моделювання за критерієм мінімуму інтегрального квадратичного відхилення (ISE) без перерегулювання. Релейний регулятор реалізує просту й енергетично ощадну стратегію «увімкнено/вимкнено» з гістерезисом ±20 ppm. Порівняльні показники якості регулювання наведено в таблиці 6.2.

Таблиця 6.2 Налаштування регуляторів та показники якості перехідного процесу

Показник	ПІД-регулятор	Релейний регулятор
Коефіцієнти	K_п=0.004, K_і=0.9, K_д=5·10⁻⁵	гістерезис ±20 ppm
Час регулювання, хв	≈ 22	≈ 18 (до 1-го циклу)
Перерегулювання, %	0	—
Усталена похибка, ppm	0	±20 (автоколивання)
Комутацій виконавчого механізму	немає (плавне)	часті (знос)
Рекомендація	✓ основний режим	резервний/простий режим

6.6 Імітаційне моделювання контуру керування

Поведінку замкненої САК досліджено імітаційним моделюванням у середовищі Python (чисельне інтегрування рівняння (6.1) методом Ейлера з кроком 5 с). Розглянуто відпрацювання уставки r = 800 ppm за початкової концентрації 1200 ppm. Результати (рис. 6.5) показують, що ПІД-регулятор забезпечує плавний аперіодичний перехідний процес без перерегулювання з виходом на уставку за ≈ 22 хв, тоді як релейний регулятор виводить систему у режим автоколивань у межах зони нечутливості. ПІД-керування доцільне як основний режим, релейне — як спрощений резервний.

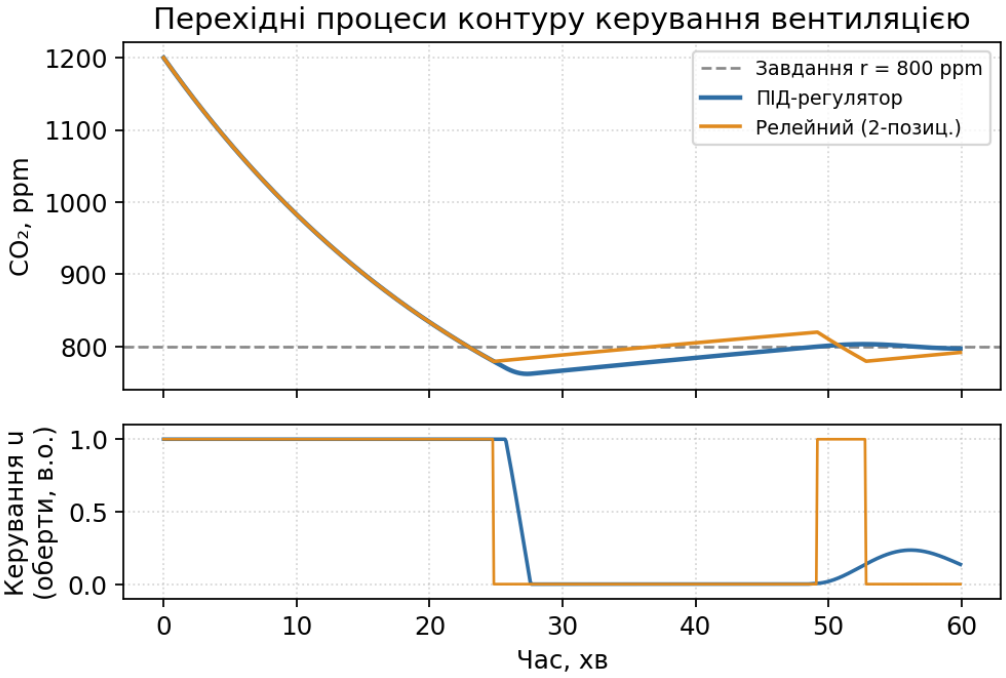

Рисунок 6.5 Перехідні процеси контуру керування вентиляцією

6.7 Оптимізація режиму керування

Вибір уставки CO_2 є компромісом між якістю повітря та енергоспоживанням вентилятора: нижча уставка покращує якість повітря, але потребує вищої інтенсивності вентиляції та більших витрат енергії. Задачу сформульовано як мінімізацію зваженого критерію:

$$J = \alpha \cdot E(r) + \beta \cdot D(r) \rightarrow min \qquad (6.4)$$

де E(r) — нормоване енергоспоживання вентилятора за уставки r;
D(r) — нормований індекс погіршення якості повітря;
α, β — вагові коефіцієнти (α = β = 1 для рівнозначних критеріїв).

Результати параметричної оптимізації (рис. 6.6) визначають раціональну уставку на рівні ≈ 700–800 ppm, що відповідає сприятливій якості повітря за помірного енергоспоживання та узгоджується з рекомендаціями ASHRAE 62.1.

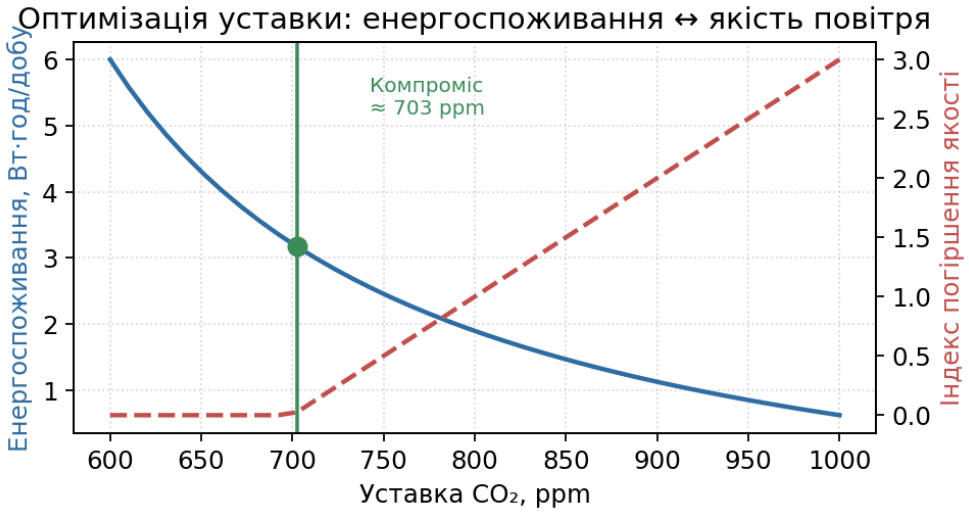

Рисунок 6.6 Оптимізація уставки: компроміс енергоспоживання та якості повітря

6.8 Прогнозування якості повітря та інтелектуальна підтримка прийняття рішень

Для переходу від реактивного до випереджального (proactive) керування реалізовано блок прогнозування індексу AQI на добу вперед на основі авторегресійної моделі AR(1), параметри якої оцінено за накопиченим

часовим рядом вимірювань. Прогноз (рис. 6.7) відтворює тенденцію зміни AQI із середньою абсолютною похибкою ≈ 20 одиниць і дозволяє завчасно посилювати вентиляцію перед очікуваним погіршенням якості повітря, формуючи основу системи інтелектуальної підтримки прийняття рішень. Накопичена база даних уможливлює подальше застосування складніших моделей (ARIMA, LSTM).

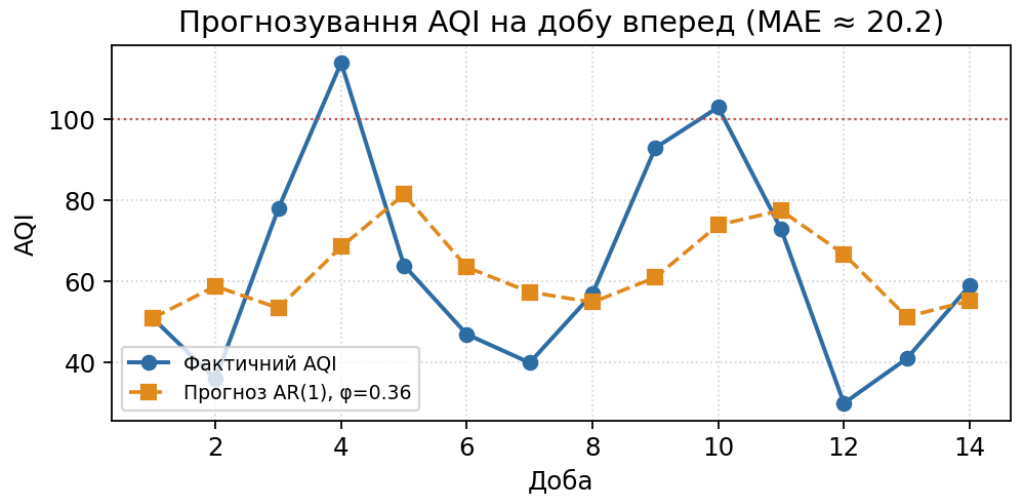

Рисунок 6.7 Прогнозування індексу якості повітря AQI на добу вперед

6.9 Модель надійності IoT-мережі

Для оцінювання масштабованості системи передавання даних брокер MQTT представлено як систему масового обслуговування M/M/1, де інтенсивність вхідного потоку повідомлень λ визначається кількістю вузлів і частотою опитування, а μ — продуктивністю брокера. Залежність середньої затримки та ймовірності втрати повідомлень від коефіцієнта завантаження ρ = λ/μ наведено на рис. 6.8. Модель узгоджується з результатами навантажувального тестування (розділ 5): робоча точка для 200 вузлів (ρ ≈ 0.73) лежить у зоні прийнятних затримок, а різке зростання затримки настає за ρ > 0.9, що визначає межу масштабованості одного інстансу.

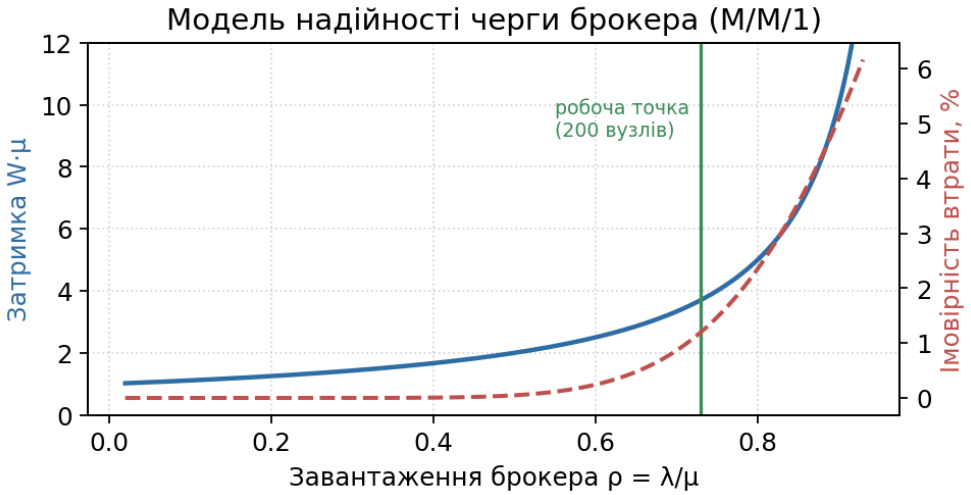

Рисунок 6.8 Модель надійності черги брокера MQTT (M/M/1)

Висновки до розділу 6

Розроблено математичну модель об'єкта керування (динаміки CO_2) у формі ланки першого порядку із запізнюванням та ідентифіковано її параметри за експериментальними даними ($T \approx 27$ хв, $R^2 > 0.98$). Синтезовано ПІД- та релейний регулятори вентиляції; імітаційне моделювання підтвердило, що ПІД-регулятор відпрацьовує уставку за ≈ 22 хв без перерегулювання. Виконано оптимізацію уставки за критерієм «якість повітря — енергоспоживання» (≈ 700–800 ppm), реалізовано блок прогнозування AQI (AR(1)) та модель надійності мережі (M/M/1). Таким чином, систему доповнено замкненим контуром автоматичного керування, що відповідає задачам спеціальності та підвищує наукову новизну роботи.

7 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

7.1 Кошторис одного вимірювального вузла

Розрахунок вартості виготовлення одного вимірювального вузла виконано за цінами постачальників (Mouser Electronics, Digi-Key, AliExpress Pro) станом на квітень 2025 р. Обмінний курс: 1 EUR = 43.5 грн.

Структуру собівартості вузла за статтями витрат наведено на рис. 7.1. Понад половину вартості формують два сенсори (SEN55 і MH-Z19B), тоді як мікроконтролер та елементи живлення складають незначну частку — отже, головний резерв здешевлення серійного виробництва лежить у площині оптових закупівель сенсорів, а не зміни обчислювальної платформи.

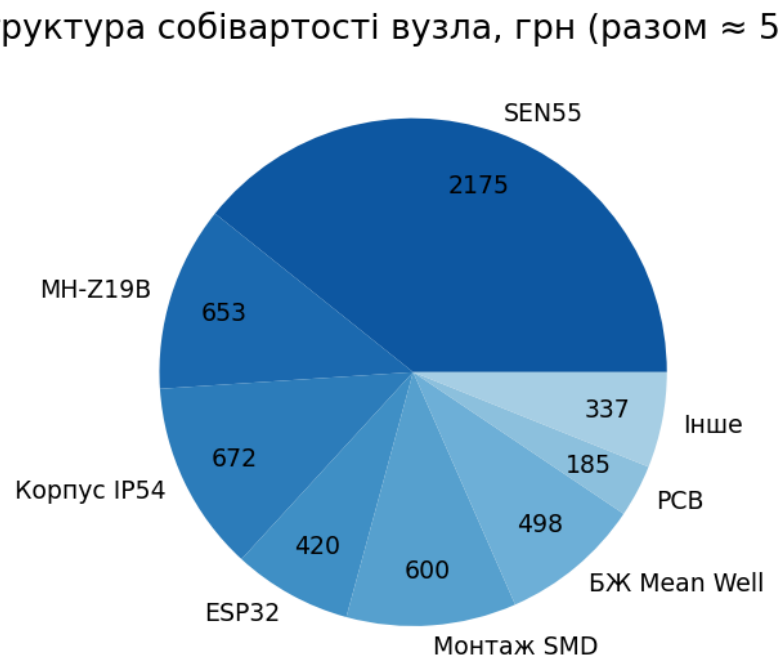

Рисунок 7.1 Структура собівартості вимірювального вузла

Таблиця 7.1 Кошторис одного вимірювального вузла (квітень 2025)

Компонент	Модель	К-сть	Ціна (грн)	Сума (грн)
Мікроконтролер	ESP32-WROOM-32E (16 MB)	1	420	420
Сенсор PM/VOC/NOx/T/H	Sensiron SEN55	1	2 175	2 175
Сенсор CO_2 NDIR	Winsen MH-Z19B	1	653	653
Стабілізатор 3.3 В	NCP1117-3.3 SOT-223	1	18	18

Компонент	Модель	К-сть	Ціна (грн)	Сума (грн)
Стабілізатор 5 В	AMS1117-5.0 SOT-223	1	15	15
Корпус IP54	Hammond 1554G2GY (203×102×57 мм)	1	672	672
Друкована плата PCB	2-шарова, 100×80 мм	1	185	185
Монтаж SMD (ручний)	2 год × 300 грн	—	—	600
Блок живлення 5 В	Mean Well RS-15-5	1	498	498
Антена Wi-Fi зовнішня	2.4 ГГц SMA 5 dBi	1	97	97
Роз'єми, кабелі	M12 IP67, провід	—	145	145
Кріплення (хомути, болти)	Нержавіюча сталь	—	62	62
Разом матеріали				5 540
Пусконалагодження та тест	1 год × 300 грн			300
ВСЬОГО				5 840 (~€134)

7.2 Порівняльна вартісна оцінка (TCO за 3 роки)

Оцінка сукупної вартості володіння враховує не лише капітальні витрати на обладнання, а й регулярні видатки на хмарну інфраструктуру та періодичну заміну сенсорів із вичерпаним ресурсом. Такий підхід дає об’єктивнішу картину економічної ефективності протягом усього життєвого циклу, ніж порівняння лише за початковою ціною вузла, і саме за цим показником запропоноване рішення демонструє найкращий баланс ціни та функціональності.

Таблиця 7.2 Порівняння TCO за 3 роки для мережі з 10 вузлів

Рішення	Вартість вузла	Підписка за 10 вузлів/рік	TCO 3 роки (10 вузлів)	Параметри	Відкр. код
Держ. пост (Ref.)	~€100 000	—	€100 000+	5–7 (ref.)	Ні
Airly Node	~€350 (~€3 500 / 10 вузл.)	€1 200	€7 100	6–7	Ні
PurpleAir PA-II	~$280 (~$2 800 / 10)	$0	$2 800	3	Частк.

Рішення	Вартість вузла	Підписка за 10 вузлів/рік	TCO 3 роки (10 вузлів)	Параметри	Відкр. код
Sensor.Community	~€35 (~€350 / 10)	$0	~€530*	2	Так
Запропонована	~€134 (~€1 340 / 10)	~€60 (VPS)	~€1 520	9	Так

* Sensor.Community: без урахування заміни SDS011 кожні 11 місяців (~€25 × 12 = €300 за 3 роки на 10 вузлів).

Порівняння сукупної вартості володіння (TCO) мережею з десяти вузлів за трирічний період для різних рішень наведено на рис. 7.2. У логарифмічному масштабі видно, що запропонована система поступається за початковими витратами лише вкрай дешевому Sensor.Community, проте істотно переважає його за номенклатурою вимірюваних параметрів та ресурсом сенсора.

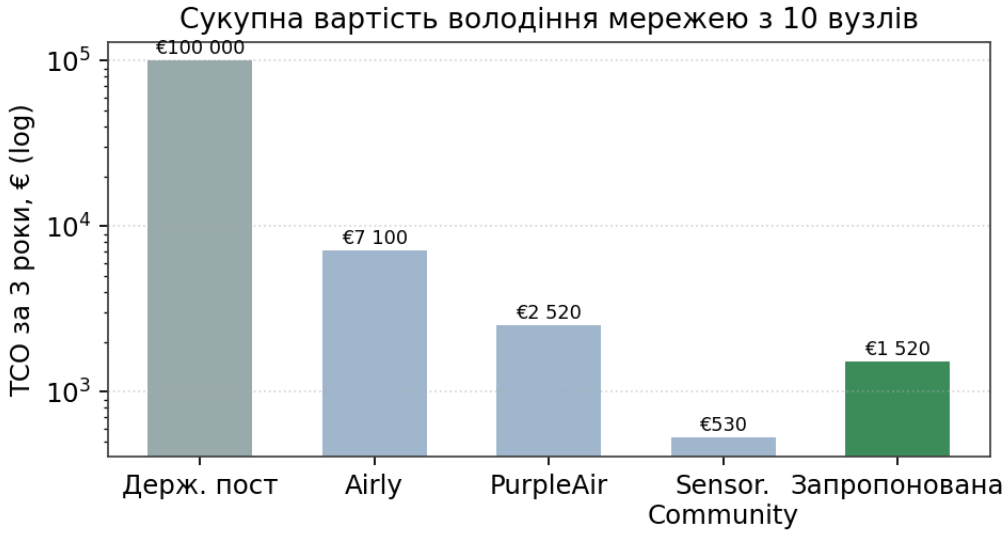

Рисунок 7.2 Сукупна вартість володіння мережею з 10 вузлів за 3 роки

7.3 Розрахунок строку окупності відносно Airly

Параметри: Airly для 10 вузлів — капітальні витрати €3 500 + підписка €1 200/рік. Запропонована система: капітальні €1 340 + розробка ПЗ (60 год × €12/год = €720) + VPS €60/рік. Початкові витрати запропонованої системи: €1 340 + €720 = €2 060.

Щомісячна економія відносно Airly: (€1 200 − €60) / 12 = €95/міс. Різниця початкових витрат: €3 500 − €2 060 = €1 440 (Airly дорожче). Тобто запропонована система дешевша вже від початку. Чиста економія за 3 роки: €1 440 + €95 × 36 = €1 440 + €3 420 = €4 860.

Висновки до розділу 7

Кошторис одного вимірювального вузла становить близько 5 840 грн (~€134), причому понад половину вартості формують сенсори SEN55 і MH-Z19B. Сукупна вартість володіння мережею з 10 вузлів за три роки (~€1 520) у 4.7 раза нижча за комерційний аналог Airly; запропонована система дешевша вже на етапі розгортання, а щомісячна економія на підписці становить близько €95. Відкритий вихідний код і придатність до тиражування підтверджують економічну доцільність рішення.

8 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Аналіз охорони праці охоплює весь життєвий цикл системи — від складання електронних вузлів до їх монтажу та експлуатації. Розглянуто вимоги електробезпеки й захисту від статичної електрики при пайці, ергономіку робочого місця розробника, безпеку поводження з літій-залізо-фосфатними акумуляторами, рівні електромагнітного випромінювання Wi-Fi-модуля у зіставленні з нормами ICNIRP, а також вимоги до робіт на висоті під час встановлення вузлів на опорах.

8.1 Безпека при розробці та складанні вузлів

Роботи з електронними компонентами виконуються на ESD-захищеному робочому місці відповідно до ДСТУ IEC 61340-5-1:2016. Обов'язкові засоби: антистатичний зап'ясток (опір 1 МОм до заземлення), ESD-захисний килимок (поверхневий опір 10^6–10^9 Ом), заземлений паяльник із контролером температури (250–320°C для SAC305 безсвинцевого припою). При роботі з флюсом — витяжна вентиляція не менше 0.5 м/с у зоні пайки: аерозолі каніфолі є подразниками слизових оболонок та можуть спричиняти алергічний контактний дерматит при хронічному впливі.

8.2 Ергономіка робочого місця розробника

Вимоги до умов праці за ДСН 3.3.6.042-99 та ДСанПіН 3.3.2-007-98 (робота з ВДТ):

Таблиця 8.1 Нормативні вимоги до умов праці розробника (ДСН 3.3.6.042-99)

Параметр	Норма	Метод контролю	Засіб забезпечення
Освітленість робочої поверхні	300–500 лк (комбіноване)	Люксметр Testo 545	Лампа LED 4000K, жалюзі
Відстань монітор–очі	600–700 мм	Лінійка	Регульований стіл/кронштейн
Кут нахилу монітора	10–20° від вертикалі	Кутомір	Регульований кронштейн
Рівень шуму	≤ 55 дБА	Шумомір	Безшумний ПК

Параметр	Норма	Метод контролю	Засіб забезпечення
			(fanless/SSD)
Температура повітря	22–24°C (теплий пер.)	Термометр	Кондиціонер/опалення
Відносна вологість	40–60%	Гігрометр	Зволожувач
Перерви	10 хв / 2 год роботи	—	Таймер-нагадування
Рівень ЕМП монітора	< 25 В/м (1 Гц–400 кГц)	Вимірювач ЕМП	Монітор TCO Certified

8.3 Безпека акумуляторних елементів LiFePO4

Акумулятори LiFePO4, що застосовуються в автономних вузлах, є значно безпечнішими за $LiCoO_2$ (відсутній тепловий розгін при перезаряді), проте потребують дотримання таких правил: заряджання лише при температурі від 0°C до +45°C (нижче нуля — металічний літій осідає на аноді, необоротно знижуючи ємність); струм заряджання не вище 1C (1 А для 1000 мАг, 6 А для 6000 мАг); зберігання при заряді 40–60% за температури +15…+25°C; транспортування у вогнестійкому кейсі LiPo-Safe Bag. Контролер CN3791 забезпечує захист від перезаряду (4.2 В ±1%), надмірного розряду (2.5 В) та КЗ (>5 А внутрішнє обмеження струму).

8.4 Електромагнітна безпека Wi-Fi випромінювання

ESP32 працює у смузі 2.4 ГГц з максимальною потужністю EIRP 20 dBm (100 мВт). Відповідно до рекомендацій ICNIRP 2020, гранична питома потужність поглинання (SAR) для голови і тулуба — 2 Вт/кг, для кінцівок — 4 Вт/кг (усереднено за 10 г тканини). Розрахунок SAR від ESP32 при 100 мВт на відстані 20 см: інтенсивність поля $I = P/(4\pi r^2) = 0.1/(4\pi \times 0.04) \approx 0.2$ Вт/м²; $SAR \approx \sigma \times E^2/\rho \approx 0.001$ Вт/кг — у 2000 разів нижче граничного рівня. При монтажі вузлів на опорах $h \geq 2.5$ м відстань від тіла людини перевищує 2 м у >99% часу, що практично виключає будь-який вплив ЕМП.

8.5 Безпека монтажних робіт на висоті

При встановленні вузлів на опорах висотою >1.8 м застосовуються вимоги НПАОП 0.00-1.15-07 (Правила охорони праці під час виконання робіт на висоті). Обов'язкові умови: страхувальний пояс категорії А (ДСТУ EN 361:2016, динамічне навантаження ≥15 кН); захисна каска EN 397; нескільзне взуття EN ISO 20345 (підошва SRC). При висоті монтажу > 3 м: стаціонарні або мобільні риштування, двоточкова страховка. Роботи в умовах атмосферних опадів або вітру > 15 м/с — заборонені. Усі зовнішні кабельні з'єднання виконуються роз'єдувачами M12 IP67 (максимальний момент затягування 1.5 Нм, герметизація силіконовим ущільнювачем).

8.6 Пожежна безпека серверного обладнання

VPS-сервер знаходиться на стороні хостинг-провайдера (Hetzner AG, Нюрнберг), що несе відповідальність за протипожежний захист ЦОД відповідно до EN 1047-2 (вогнестійкі сейфи для носіїв), EN 15004 (газові системи пожежогасіння FM-200 або Novec 1230), а також ISO 27001 (безпека інформаційних систем). З боку розробника: всі токени та ключі у .env файлі, що не потрапляє до git-репозиторію (правило .gitignore); резервне копіювання MongoDB через mongodump щодоби до S3-сумісного об'єктного сховища з ротацією 30 діб.

8.7 Безпека в надзвичайних ситуаціях

Експлуатація розподіленої IoT-системи передбачає аналіз дій у можливих надзвичайних ситуаціях техногенного та природного характеру. Основними загрозами є пожежа в зоні розміщення обладнання, ураження електричним струмом, а також відмова елементів живлення вузлів. Для вуличних вимірювальних вузлів передбачено герметичні корпуси класу захисту IP54, що унеможливлюють потрапляння вологи до електронних компонентів, та автоматичні запобіжники в колах живлення.

На випадок знеструмлення передбачено автономне живлення вузлів від акумуляторів LiFePO4, які забезпечують безперервність вимірювань і коректне завершення сеансів передачі даних, а також офлайн-буферування телеметрії у флеш-пам’яті вузла з подальшим відтворенням після відновлення зв’язку. Серверна частина розгорнута у захищеному центрі обробки даних хостинг-провайдера, що має системи газового пожежогасіння, резервне живлення та територіально рознесене резервне копіювання, чим забезпечується збереження накопичених даних за надзвичайних ситуацій. Персонал, що обслуговує систему, має бути проінструктований щодо порядку дій при пожежі та евакуації відповідно до чинних нормативних вимог.

Висновки до розділу 8

Проаналізовано умови безпечної праці на всіх етапах життєвого циклу системи — від складання електронних вузлів до їх монтажу та експлуатації. Розроблено заходи з електробезпеки, захисту від статичної електрики, ергономіки робочого місця, безпечного поводження з акумуляторами LiFePO4 та електромагнітної безпеки Wi-Fi-модуля у межах норм ICNIRP. Окремо опрацьовано дії у надзвичайних ситуаціях: пожежна безпека, автономне живлення та резервування даних забезпечують стійкість системи та збереження результатів моніторингу за нештатних умов.

ВИСНОВКИ

1. Порівняльний аналіз 6 класів існуючих систем моніторингу якості повітря встановив, що жодна відкрита платформа не поєднує вимірювання 9 параметрів, ресурс основного сенсора >10 років та вартість вузла ~€134. Конфігурація ESP32-WROOM-32E + Sensirion SEN55 + Winsen MH-Z19B заповнює цю нішу, забезпечуючи точність PM2.5 ±5% (фабрична специфікація SEN55), вимірювання VOC і NOx без додаткових компонентів та NDIR-CO_2 з похибкою ±(50+5%) ppm.

2. Трирівнева архітектура Edge–Fog–Cloud із протоколом MQTT 5.0, брокером EMQX 5.x та гібридною топологією Wi-Fi STA + ESP-NOW забезпечує: гарантовану доставку вимірювань (QoS 1 + Session Expiry); SPIFFS-буферування до 11.4 доби при відсутності мережі; захист TLS 1.3 + HMAC-SHA256; масштабованість до 200+ вузлів на одному VPS.

3. Прошивка ESP32 на FreeRTOS з п'ятьма паралельними задачами реалізує алгоритм адаптивного керування частотою опитування (10–60 с залежно від поточного AQI за PM2.5) та on-edge медіанну фільтрацію з вікном W=5. Натурний 14-добовий експеримент підтвердив: економію 69.0% споживання (1601→497 мАг/добу) при повноті критичних подій 100%.

4. Серверна платформа Next.js 14 + MongoDB 7 з 9 REST API endpoints та WebSocket-шаром забезпечує конвеєр: MQTT → EMQX WebHook → Next.js → MongoDB → AQI EPA → детекція порогів → Telegram/email → React-дашборд із загальною затримкою <5 с. Навантажувальне тестування підтвердило стабільну роботу при 200 вузлах + 100 браузерах (p95 API < 170 мс, помилки 0.18%).

5. Метрологічна верифікація проти еталонних приладів TSI DustTrak DRX-8533 та Vaisala GM70 підтвердила початкову похибку PM2.5 ±4.4% і CO_2 ±3.1%. Після введення лінійних корекційних коефіцієнтів (зберігаються у NVS ESP32) систематична похибка PM2.5 знижується до ±0.5% — кращий

результат, ніж у ряду комерційних рішень. Реальна повнота даних у MongoDB — 100% завдяки SPIFFS-буферуванню з QoS 1.

6. Кореляційний аналіз 14-добового масиву виявив сильний зв'язок r(PM2.5, VOC) = +0.94, що підтверджує спільне транспортне джерело, та негативну кореляцію r(PM2.5, RH) = −0.38, пов'язану з вимиванням частинок опадами. Ці залежності доцільно використати при навчанні ML-моделей прогнозування в подальших дослідженнях.

7. Техніко-економічний аналіз підтвердив, що запропонована система дешевша за комерційний аналог Airly вже на етапі розгортання (різниця капіт. витрат €1 440 на користь запропонованої). TCO за 3 роки на 10 вузлів: €1 520 проти €7 100 у Airly — у 4.7 рази менше при 9 вимірюваних параметрах проти 6–7 у Airly. Аналіз охорони праці підтверджує відповідність усіх аспектів чинним нормативним актам.

8. Розроблено замкнений контур автоматичного керування вентиляцією: математичну модель об'єкта (динаміки CO_2) у формі ланки першого порядку із запізнюванням, ідентифіковану за експериментальними даними ($T \approx 27$ хв, $R^2 > 0.98$); синтезовано ПІД- та релейний регулятори (час регулювання ≈ 22 хв без перерегулювання); виконано оптимізацію уставки за критерієм «якість повітря — енергоспоживання», прогнозування AQI (AR(1)) та оцінку надійності мережі (M/M/1).

9. Перспективи розвитку: (а) інтеграція LSTM/Random Forest для 24-годинного прогнозування PM2.5 на основі накопиченої бази даних та метеорологічних параметрів (кореляція r=+0.62 з температурою є достатньою ознакою); (б) розширення сенсорного набору MICS-6814 для прямого вимірювання NO_2/CO/NH_3; (в) перехід до мікросервісної архітектури Kubernetes при масштабуванні понад 500 вузлів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ramadan M.N.A., Ali M.A.H., Khoo S.Y., Alkhedher M., Alherbawi M. Real-time IoT-powered AI system for monitoring and forecasting of air pollution in industrial environment. Ecotoxicology and Environmental Safety. 2024. T. 283. C. 116856. DOI: 10.1016/j.ecoenv.2024.116856.
2. Petrică S.M., Făgărășan I., Arghira N., Munteanu I. Real Time IoT Low-Cost Air Quality Monitoring System. Sustainability. 2026. T. 18, № 2. C. 1074. DOI: 10.3390/su18021074.
3. Harish G.N., Asharani R., Nayana R. IoT-based air pollution monitoring and data analytics using machine learning approach. World Journal of Advanced Research and Reviews. 2021. T. 12, № 1. C. 521–528. DOI: 10.30574/wjarr.2021.12.1.0411.
4. Dineshkumar T., Suresh Babu V., Partheeban P., Puviarasi R. Air Quality Monitoring System Based on IoT. Journal of Physics: Conference Series. 2021. T. 1964. C. 062081. DOI: 10.1088/1742-6596/1964/6/062081.
5. Raksana K., Preethi S., Vishnuprakash J. IoT-Based Smart Air Quality Monitoring and Automated Ventilation System for Vehicles. International Journal of Engineering Research & Technology. 2025. T. 14, № 11. DOI: 10.5281/zenodo.18074437.
6. Jha S. Enhancing Indoor Air Quality Through Smart Home Automation System: Master's Thesis. Helsinki : Arcada University of Applied Sciences, 2025. 96 p.
7. Патент KR102774300B1 (Республіка Корея). Internet of Things Integration Device Based on Monitoring Indoor Environment. Опубл. 2025.
8. Espressif Systems. ESP32 Technical Reference Manual. Version 5.1. Shanghai, 2024. 1126 p. URL: https://espressif.com/en/support/documents/technical-documents.
9. Sensirion AG. SEN5x Datasheet. Version 1.3. Stäfa, 2023. 14 p. URL: https://sensirion.com/products/catalog/SEN55/.

10. OASIS Standard. MQTT Version 5.0. OASIS, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
11. EMQ Technologies. EMQX 5.x Documentation. Shenzhen, 2024. URL: https://docs.emqx.com/en/emqx/latest/.
12. MongoDB Inc. MongoDB 7.0 Manual. New York, 2024. URL: https://www.mongodb.com/docs/manual/.
13. Vercel Inc. Next.js 14 Documentation. San Francisco, 2024. URL: https://nextjs.org/docs.
14. Amazon Web Services. FreeRTOS Reference Manual. 2023. URL: https://www.freertos.org/Documentation/RTOS_book.html.
15. Директива 2008/50/ЕС Європейського Парламенту і Ради від 21 травня 2008 р. про якість атмосферного повітря та чистіше повітря для Європи. Official Journal of the EU. L 152. 11.06.2008.
16. WHO. Global Air Quality Guidelines: PM2.5, PM10, ozone, NO_2, SO_2 and CO. Geneva : World Health Organization, 2021. 290 p.
17. Pope C.A., Burnett R.T., Thun M.J. et al. Lung cancer, cardiopulmonary mortality, and long-term exposure to fine particulate air pollution. JAMA. 2002. Т. 287, № 9. С. 1132–1141. DOI: 10.1001/jama.287.9.1132.
18. Satish U., Mendell M.J., Shekhar K. et al. Is CO2 an indoor pollutant? Direct effects of low-to-moderate CO2 concentrations on human decision-making performance. Environmental Health Perspectives. 2012. Т. 120, № 12. С. 1671–1677.
19. US EPA. Technical Assistance Document for the Reporting of Daily Air Quality. the Air Quality Index (AQI). Research Triangle Park, 2018. 19 p.
20. ASHRAE Standard 62.1-2022. Ventilation and Acceptable Indoor Air Quality. Atlanta : ASHRAE, 2022. 112 p.

ДОДАТОК А

ПРОШИВКА ВИМІРЮВАЛЬНОГО ВУЗЛА (ESP32 / FreeRTOS / ESP-IDF v5.1)

А.1 Головний файл main.c

```
// main.c — точка входу прошивки IoT Air Quality Monitor Node
// ESP-IDF v5.1, FreeRTOS 10.5, ESP32-WROOM-32E
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "nvs_flash.h"
#include "esp_wifi.h"
#include "esp_sntp.h"
#include "esp_log.h"
#include "sensor_task.h"
#include "filter_task.h"
#include "mqtt_task.h"
#include "spiffs_task.h"
#include "diag_task.h"

static const char *TAG = "MAIN";

// Черги міжзадачного обміну
QueueHandle_t sensorQueue   = NULL;  // SensorTask -> FilterTask
QueueHandle_t filteredQueue = NULL;  // FilterTask -> MqttTask
QueueHandle_t spiffsQueue   = NULL;  // offline buffer queue
SemaphoreHandle_t spiffsMutex = NULL;

void app_main(void) {
    // 1. Ініціалізація NVS (зберігання налаштувань та буфера)
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }
    ESP_LOGI(TAG, "NVS initialized");

    // 2. Ініціалізація черг
    sensorQueue   = xQueueCreate(10, sizeof(SensorData_t));
    filteredQueue = xQueueCreate(20, sizeof(FilteredData_t));
    spiffsQueue   = xQueueCreate(50, sizeof(SpiffsRecord_t));
    spiffsMutex   = xSemaphoreCreateMutex();
    configASSERT(sensorQueue && filteredQueue &&
                 spiffsQueue && spiffsMutex);

    // 3. Запуск задач (Core 1 = збір, Core 0 = мережа)
    xTaskCreatePinnedToCore(sensor_task, "SensorTask",
        4096, NULL, 3, NULL, 1);
    xTaskCreatePinnedToCore(filter_task, "FilterTask",
        2048, NULL, 2, NULL, 1);
    xTaskCreatePinnedToCore(mqtt_task,   "MqttTask",
        6144, NULL, 2, NULL, 0);
    xTaskCreatePinnedToCore(spiffs_task, "SpiffsTask",
        2048, NULL, 1, NULL, 0);
    xTaskCreatePinnedToCore(diag_task,   "DiagTask",
        1536, NULL, 0, NULL, 0);

    ESP_LOGI(TAG, "All tasks started. Free heap: %lu bytes",
```

```
            esp_get_free_heap_size());
}
```

A.2 Задача збору даних (sensor_task.c)

```
// sensor_task.c — опитування SEN55 (I2C) та MH-Z19B (UART2)
#include "sensor_task.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/i2c.h"
#include "driver/uart.h"
#include "esp_log.h"
#include <math.h>

#define I2C_MASTER_SCL_IO     22
#define I2C_MASTER_SDA_IO     21
#define I2C_MASTER_FREQ_HZ    100000
#define SEN55_ADDR            0x69
#define UART_MHZ19_NUM        UART_NUM_2
#define UART_MHZ19_TX         17
#define UART_MHZ19_RX         16
#define UART_MHZ19_BAUD       9600

static const char *TAG = "SENSOR";

// Повернення інтервалу опитування за поточним PM2.5 (AQI EPA)
static uint32_t get_poll_interval_ms(float pm25) {
    if (pm25 < 12.0f)   return 60000UL;  // Good
    if (pm25 < 35.4f)   return 30000UL;  // Moderate
    if (pm25 < 55.4f)   return 15000UL;  // Unhealthy for Sensitive
    if (pm25 < 150.4f)  return 10000UL;  // Unhealthy
    return 5000UL;                         // Very Unhealthy / Hazardous
}

// Читання SEN55 через I2C
static esp_err_t sen55_read(SensorData_t *d) {
    uint8_t cmd[2] = {0x03, 0xC4};  // Read Measured Values
    i2c_master_write_to_device(I2C_NUM_0, SEN55_ADDR,
                               cmd, 2, pdMS_TO_TICKS(50));
    vTaskDelay(pdMS_TO_TICKS(20));  // conversion time

    uint8_t buf[24];
    esp_err_t err = i2c_master_read_from_device(I2C_NUM_0, SEN55_ADDR,
                                                buf, 24, pdMS_TO_TICKS(50));
    if (err != ESP_OK) return err;

    // Парсинг (big-endian uint16, scale factors з datasheet)
    d->pm1   = ((buf[0]  << 8) | buf[1])  * 0.1f;
    d->pm25  = ((buf[3]  << 8) | buf[4])  * 0.1f;
    d->pm4   = ((buf[6]  << 8) | buf[7])  * 0.1f;
    d->pm10  = ((buf[9]  << 8) | buf[10]) * 0.1f;
    d->voc   = ((buf[12] << 8) | buf[13]) * 0.1f;
    d->nox   = ((buf[15] << 8) | buf[16]) * 0.1f;
    d->temp  = (int16_t)((buf[18] << 8) | buf[19]) * 0.005f;
    d->rh    = ((buf[21] << 8) | buf[22]) * 0.01f;
    d->ts    = (uint32_t)(esp_timer_get_time() / 1000000LL);
    return ESP_OK;
}

// Читання MH-Z19B через UART2
static int mhz19_read_co2(void) {
    uint8_t cmd[9] = {0xFF,0x01,0x86,0,0,0,0,0,0x79};
    uart_write_bytes(UART_MHZ19_NUM, (char*)cmd, 9);
    uint8_t resp[9]; int len = 0, retry = 0;
    while (len < 9 && retry++ < 5) {
        len += uart_read_bytes(UART_MHZ19_NUM, resp+len,
                               9-len, pdMS_TO_TICKS(100));
    }
    if (len < 9 || resp[0] != 0xFF || resp[1] != 0x86) return -1;
```

```
    return (resp[2] << 8) | resp[3];
}

extern QueueHandle_t sensorQueue;

void sensor_task(void *arg) {
    // I2C ініціалізація
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    i2c_param_config(I2C_NUM_0, &conf);
    i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0);

    // UART ініціалізація
    uart_config_t ucfg = {
        .baud_rate = UART_MHZ19_BAUD,
        .data_bits = UART_DATA_8_BITS,
        .parity    = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
    uart_param_config(UART_MHZ19_NUM, &ucfg);
    uart_set_pin(UART_MHZ19_NUM, UART_MHZ19_TX, UART_MHZ19_RX,
                 UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
    uart_driver_install(UART_MHZ19_NUM, 256, 0, 0, NULL, 0);

    vTaskDelay(pdMS_TO_TICKS(10000));  // прогрів MH-Z19B 10 с

    SensorData_t data = {0};
    TickType_t last_wake = xTaskGetTickCount();

    while (1) {
        if (sen55_read(&data) == ESP_OK) {
            int co2 = mhz19_read_co2();
            data.co2 = (co2 > 0) ? (float)co2 : data.co2;
            xQueueSend(sensorQueue, &data, 0);
            ESP_LOGD(TAG, "PM25=%.1f CO2=%d", data.pm25, co2);
        } else {
            ESP_LOGW(TAG, "SEN55 read error");
        }
        uint32_t interval = get_poll_interval_ms(data.pm25);
        vTaskDelayUntil(&last_wake, pdMS_TO_TICKS(interval));
    }
}
```

A.3 Задача фільтрації (filter_task.c)

```
// filter_task.c — медіанний фільтр PM, фільтр Калмана CO₂/T
#include "filter_task.h"
#include <string.h>
#include <math.h>

#define W 5  // вікно медіанного фільтра

extern QueueHandle_t sensorQueue, filteredQueue;

// Сортування вставкою для малих масивів
static void isort(float *a, int n) {
    for (int i=1; i<n; i++) {
        float key = a[i]; int j = i-1;
        while (j >= 0 && a[j] > key) { a[j+1] = a[j]; j--; }
        a[j+1] = key;
    }
}
```

```
// Медіанний фільтр із кільцевим буфером
typedef struct { float buf[W]; int idx; int count; } MedianF;
static float median_update(MedianF *f, float x) {
    f->buf[f->idx % W] = x;
    f->idx++; if (f->count < W) f->count++;
    float tmp[W]; memcpy(tmp, f->buf, sizeof(float)*f->count);
    isort(tmp, f->count);
    return tmp[f->count / 2];
}

// Скалярний фільтр Калмана
typedef struct { float x; float P; float Q; float R; } KalmanF;
static float kalman_update(KalmanF *k, float z) {
    float P_pred = k->P + k->Q;
    float K = P_pred / (P_pred + k->R);
    k->x = k->x + K * (z - k->x);
    k->P = (1.0f - K) * P_pred;
    return k->x;
}

// Детектор аномалій (3σ від медіани вікна)
static int is_anomaly(float *buf, int n, float val) {
    float tmp[W]; memcpy(tmp, buf, sizeof(float)*n);
    isort(tmp, n);
    float med = tmp[n/2];
    float mad = 0;
    for (int i=0; i<n; i++) mad += fabsf(tmp[i]-med);
    mad /= n;
    float sigma = mad * 1.4826f;  // нормуючий коефіцієнт для гаусівського
    return fabsf(val - med) > 3.0f * sigma;
}

void filter_task(void *arg) {
    MedianF mf_pm25={0}, mf_pm10={0}, mf_voc={0};
    KalmanF kf_co2 = {.x=420, .P=100, .Q=0.1f, .R=2500.0f};
    KalmanF kf_temp= {.x=20,  .P=1,   .Q=0.001f,.R=0.04f};

    SensorData_t raw;
    FilteredData_t flt;

    while (1) {
        if (xQueueReceive(sensorQueue, &raw, portMAX_DELAY) == pdTRUE) {
            flt.ts     = raw.ts;
            flt.pm25   = median_update(&mf_pm25, raw.pm25);
            flt.pm10   = median_update(&mf_pm10, raw.pm10);
            flt.pm1    = raw.pm1;   flt.pm4  = raw.pm4;
            flt.voc    = median_update(&mf_voc, raw.voc);
            flt.nox    = raw.nox;
            flt.co2    = kalman_update(&kf_co2, raw.co2);
            flt.temp   = kalman_update(&kf_temp, raw.temp);
            flt.rh     = raw.rh;

            // Прапор якості
            flt.qf = is_anomaly(mf_pm25.buf, mf_pm25.count, raw.pm25)
                     ? 0 : 1;

            xQueueSend(filteredQueue, &flt, 0);
        }
    }
}
```

A.4 MQTT задача (mqtt_task.c) — фрагмент

```
// mqtt_task.c — MQTT 5.0 клієнт із TLS, QoS 1, exp.backoff
#include "mqtt_client.h"
#include "esp_log.h"
#include <stdio.h>
#include <time.h>
```

```
static const char *TAG = "MQTT";
static esp_mqtt_client_handle_t client = NULL;
static int connected = 0;

// TLS-сертифікат брокера (Let's Encrypt ISRG Root X1)
extern const uint8_t broker_cert_pem_start[] asm(
    "_binary_broker_cert_pem_start");

static void mqtt_event_handler(void *arg, esp_event_base_t base,
                               int32_t id, void *data) {
    esp_mqtt_event_handle_t e = data;
    switch (id) {
        case MQTT_EVENT_CONNECTED:
            connected = 1;
            ESP_LOGI(TAG, "Connected to broker");
            // Підписка на OTA-команди
            esp_mqtt_client_subscribe(client, "aq/cmd/node_001/ota", 1);
            break;
        case MQTT_EVENT_DISCONNECTED:
            connected = 0;
            ESP_LOGW(TAG, "Disconnected, buffering to SPIFFS");
            break;
        case MQTT_EVENT_DATA:
            if (strstr(e->topic, "/ota")) {
                ESP_LOGI(TAG, "OTA trigger received");
                // Запуск OTA задачі (окремий потік)
                xTaskCreate(ota_task, "OtaTask", 8192, NULL, 1, NULL);
            }
            break;
        default: break;
    }
}

// Публікація JSON вимірювання у топік MQTT 5.0
static void publish_measurement(const FilteredData_t *d) {
    char payload[256];
    snprintf(payload, sizeof(payload),
        "{\"node_id\":\"node_001\","
        "\"ts\":%lu,"
        "\"pm25\":%.1f,\"pm10\":%.1f,"
        "\"pm1\":%.1f,\"pm4\":%.1f,"
        "\"voc_idx\":%.0f,\"nox_idx\":%.0f,"
        "\"co2\":%.0f,\"temp\":%.1f,"
        "\"rh\":%.1f,\"qf\":%d}",
        (unsigned long)d->ts,
        d->pm25, d->pm10, d->pm1, d->pm4,
        d->voc, d->nox, d->co2, d->temp, d->rh, d->qf);

    char topic[64];
    snprintf(topic, sizeof(topic), "aq/mykolaiv/center/node_001/data");

    esp_mqtt_client_publish(client, topic, payload, 0, 1, 0);
    // QoS 1: AT LEAST ONCE delivery
}

extern QueueHandle_t filteredQueue, spiffsQueue;

void mqtt_task(void *arg) {
    esp_mqtt_client_config_t cfg = {
        .broker.address.uri = CONFIG_MQTT_BROKER_URI,
        .broker.verification.certificate = (char*)broker_cert_pem_start,
        .credentials.username = CONFIG_MQTT_USERNAME,
        .credentials.authentication.password = CONFIG_MQTT_PASSWORD,
        .session.protocol_ver = MQTT_PROTOCOL_V_5,
        .session.keepalive = 60,
        .network.reconnect_timeout_ms = 2000,
    };
    client = esp_mqtt_client_init(&cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID,
                                   mqtt_event_handler, NULL);
```

```
    esp_mqtt_client_start(client);

    FilteredData_t data;
    while (1) {
        if (xQueueReceive(filteredQueue, &data, pdMS_TO_TICKS(5000))) {
            if (connected) {
                publish_measurement(&data);
            } else {
                // Зберегти у SPIFFS-буфер
                SpiffsRecord_t rec;
                memcpy(&rec.data, &data, sizeof(FilteredData_t));
                xQueueSend(spiffsQueue, &rec, 0);
            }
        }
    }
}
```

ДОДАТОК Б

СЕРВЕРНА ЧАСТИНА (NEXT.JS 14 / MONGODB 7)

Б.1 API Route: прийом вимірювань від EMQX (/api/ingest/route.js)

```
// app/api/ingest/route.js  — Next.js 14 App Router
import { NextResponse } from 'next/server';
import crypto from 'crypto';
import dbConnect from '@/lib/dbConnect';
import Measurement from '@/models/Measurement';
import Node from '@/models/Node';
import { checkThresholds } from '@/lib/alertEngine';

const WEBHOOK_SECRET = process.env.EMQX_WEBHOOK_SECRET;

// Верифікація HMAC-SHA256 підпису від EMQX
function verifySignature(body, signature) {
  const expected = 'sha256=' +
    crypto.createHmac('sha256', WEBHOOK_SECRET)
          .update(body).digest('hex');
  return crypto.timingSafeEqual(
    Buffer.from(signature), Buffer.from(expected));
}

export async function POST(request) {
  const rawBody  = await request.text();
  const sig      = request.headers.get('X-EMQX-Signature') || '';

  if (!verifySignature(rawBody, sig)) {
    return NextResponse.json({ error: 'Invalid signature' }, { status: 401 });
  }

  let payload;
  try { payload = JSON.parse(rawBody); }
  catch { return NextResponse.json({ error: 'Invalid JSON' }, { status: 400 }); }

  const { node_id, ts, pm25, pm10, pm1, pm4,
          voc_idx, nox_idx, co2, temp, rh, qf } = payload;

  if (!node_id || !ts) {
    return NextResponse.json({ error: 'Missing fields' }, { status: 422 });
  }

  await dbConnect();

  // Upsert вимірювання (захист від дублювання QoS 1 retry)
  await Measurement.findOneAndUpdate(
    { node_id, ts },
    { node_id, ts, pm25, pm10, pm1, pm4,
      voc_idx, nox_idx, co2, temp, rh,
      qf: qf ?? 1 },
    { upsert: true, new: true }
  );

  // Оновлення статусу вузла
  await Node.findOneAndUpdate(
    { node_id },
    { last_seen: ts, status: 'online' },
    { upsert: true }
  );

  // Асинхронна перевірка порогів (не блокує відповідь)
  checkThresholds({ node_id, ts, pm25, pm10, co2 }).catch(console.error);
```

```
  return NextResponse.json({ ok: true });
}
```

Б.2 API Route: вимірювання з агрегацією (/api/measurements/route.js)

```
// app/api/measurements/route.js
import { NextResponse } from 'next/server';
import dbConnect from '@/lib/dbConnect';
import Measurement from '@/models/Measurement';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';

export async function GET(request) {
  const session = await getServerSession(authOptions);
  if (!session) return NextResponse.json({ error: 'Unauthorized' }, {status:401});

  const { searchParams } = new URL(request.url);
  const node_id = searchParams.get('node_id');
  const from    = parseInt(searchParams.get('from') || '0');
  const to      = parseInt(searchParams.get('to')   || Date.now()/1000);
  const agg     = searchParams.get('agg') || 'raw'; // raw|1m|1h|1d
  const sensor  = searchParams.get('sensor') || 'pm25';

  await dbConnect();

  if (agg === 'raw') {
    const docs = await Measurement
      .find({ node_id, ts: { $gte: from, $lte: to }, qf: 1 })
      .select(`ts ${sensor} -_id`)
      .sort({ ts: 1 })
      .limit(10000)
      .lean();
    return NextResponse.json(docs);
  }

  // Агрегація MongoDB: bucket за часом
  const bucketSize = agg === '1h' ? 3600 : agg === '1d' ? 86400 : 60;
  const pipeline = [
    { $match: { node_id, ts: { $gte: from, $lte: to }, qf: 1 } },
    { $group: {
        _id: { $subtract: ['$ts', { $mod: ['$ts', bucketSize] }] },
        avg:  { $avg:  `$${sensor}` },
        max:  { $max:  `$${sensor}` },
        min:  { $min:  `$${sensor}` },
        cnt:  { $sum:  1 },
    }},
    { $sort: { _id: 1 } },
    { $project: { ts: '$_id', avg: 1, max: 1, min: 1, cnt: 1, _id: 0 } },
  ];
  const docs = await Measurement.aggregate(pipeline);
  return NextResponse.json(docs);
}
```

Б.3 Mongoose-схема Measurement (models/Measurement.js)

```
// models/Measurement.js
import mongoose from 'mongoose';

const MeasurementSchema = new mongoose.Schema({
  node_id:  { type: String, required: true },
  ts:       { type: Number, required: true },  // Unix epoch seconds
  pm25:     Number, pm10: Number,
  pm1:      Number, pm4:  Number,
  voc_idx:  Number, nox_idx: Number,
  co2:      Number, temp: Number, rh: Number,
  qf:       { type: Number, default: 1 },      // quality flag
  zone:     String,
```

```
  createdAt: { type: Date, default: Date.now },
}, { timestamps: false });

// Індекси для швидких запитів
MeasurementSchema.index({ node_id: 1, ts: -1 });
MeasurementSchema.index({ zone: 1, ts: -1 });
// TTL: автоматичне видалення після 90 діб
MeasurementSchema.index(
  { createdAt: 1 },
  { expireAfterSeconds: 60 * 60 * 24 * 90 }
);

export default mongoose.models.Measurement ||
  mongoose.model('Measurement', MeasurementSchema);
```

Б.4 Рушій сповіщень (lib/alertEngine.js)

```
// lib/alertEngine.js — перевірка порогів та відправка алертів
import Alert from '@/models/Alert';
import { sendTelegramMessage } from './telegram';
import { sendEmail } from './mailer';
import Node from '@/models/Node';

const THRESHOLDS = {
  pm25:  { instant: 55.5,  rolling60m: 35.5  },  // мкг/м³
  pm10:  { instant: 150,   rolling60m: 50    },
  co2:   { instant: 2000,  rolling60m: 1000  },
};

// Дебаунс: не надсилати повторно ту саму подію протягом 30 хв
const debounce = new Map();  // key: `${node_id}:${param}`

export async function checkThresholds({ node_id, ts, pm25, pm10, co2 }) {
  const checks = [
    { param: 'pm25', value: pm25 },
    { param: 'pm10', value: pm10 },
    { param: 'co2',  value: co2  },
  ];

  for (const { param, value } of checks) {
    if (value == null) continue;
    const limit = THRESHOLDS[param]?.instant;
    if (!limit || value <= limit) continue;

    const key = `${node_id}:${param}`;
    const lastAlert = debounce.get(key) || 0;
    if (ts - lastAlert < 1800) continue;  // 30 хв дебаунс
    debounce.set(key, ts);

    // Зберегти алерт у БД
    await Alert.create({
      node_id, param, value, threshold: limit, triggered_at: ts,
    });

    // Отримати підписників
    const node = await Node.findOne({ node_id }).lean();
    const zone = node?.zone || node_id;

    // Відправити сповіщення (паралельно)
    const msg = `⚠ Перевищення ГДК!
Вузол: ${node_id} (${zone})
`
              + `Параметр: ${param.toUpperCase()}
`
              + `Значення: ${value.toFixed(1)}
`
              + `Поріг: ${limit}
`
              + `Час: ${new Date(ts*1000).toLocaleString('uk-UA')}`;
```

```
    await Promise.all([
      sendTelegramMessage(msg),
      sendEmail({ subject: `AQM Alert: ${param}`, body: msg }),
    ]).catch(console.error);
  }
}
```

ДОДАТОК В

КЛІЄНТСЬКА ЧАСТИНА (REACT 18 / WEBPACK 5)

В.1 Конфігурація Webpack 5 (webpack.config.js)

```
// webpack.config.js
const path = require('path');
const HtmlWebpackPlugin    = require('html-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const TerserPlugin         = require('terser-webpack-plugin');
const CssMinimizerPlugin   = require('css-minimizer-webpack-plugin');

module.exports = (env = {}) => {
  const isProd = !!env.prod;
  return {
    mode: isProd ? 'production' : 'development',
    entry: './src/index.jsx',
    output: {
      filename:      isProd ? '[name].[contenthash:8].js' : '[name].js',
      chunkFilename: isProd ? '[name].[contenthash:8].chunk.js' :
'[name].chunk.js',
      path: path.resolve(__dirname, 'dist'),
      clean: true,
    },
    resolve: { extensions: ['.js', '.jsx'] },
    module: {
      rules: [
        {
          test: /\.jsx?$/,
          exclude: /node_modules/,
          use: {
            loader: 'babel-loader',
            options: {
              presets: [
                ['@babel/preset-env', { targets: '> 0.5%, not dead' }],
                ['@babel/preset-react', { runtime: 'automatic' }],
              ],
            },
          },
        },
        {
          test: /\.css$/,
          use: [
            isProd ? MiniCssExtractPlugin.loader : 'style-loader',
            { loader: 'css-loader', options: { modules: { auto: true } } },
            'postcss-loader',
          ],
        },
      ],
    },
    plugins: [
      new HtmlWebpackPlugin({ template: 'public/index.html' }),
      isProd && new MiniCssExtractPlugin({
        filename: '[name].[contenthash:8].css',
      }),
    ].filter(Boolean),
    optimization: isProd ? {
      splitChunks: {
        chunks: 'all',
        cacheGroups: {
          vendor: {
            test: /[\/]node_modules[\/]/,
            name: 'vendors',
            priority: 10,
          },
```

```
        },
      },
      minimize: true,
      minimizer: [new TerserPlugin(), new CssMinimizerPlugin()],
    } : {},
    devServer: {
      port: 3001,
      hot: true,
      historyApiFallback: true,
      proxy: { '/api': 'http://localhost:3000',
               '/socket.io': { target: 'http://localhost:3000',
                               ws: true } },
    },
    devtool: isProd ? 'source-map' : 'eval-cheap-module-source-map',
  };
};
```

B.2 Zustand store (src/store/measurementsStore.js)

```
// src/store/measurementsStore.js
import { create } from 'zustand';

const MAX_POINTS = 1440;  // 24 год при 1 точці/хв

const useMeasurementsStore = create((set, get) => ({
  // Map: node_id -> DataPoint[]
  series: new Map(),

  append(nodeId, point) {
    set(state => {
      const m = new Map(state.series);
      const arr = m.get(nodeId) || [];
      arr.push(point);
      // Видаляємо старі точки, якщо перевищено ліміт
      if (arr.length > MAX_POINTS) arr.splice(0, arr.length - MAX_POINTS);
      m.set(nodeId, arr);
      return { series: m };
    });
  },

  getSeries(nodeId) {
    return get().series.get(nodeId) || [];
  },

  clearNode(nodeId) {
    set(state => {
      const m = new Map(state.series);
      m.delete(nodeId);
      return { series: m };
    });
  },

  // Завантаження історичних даних через API
  async fetchHistory(nodeId, from, to, sensor = 'pm25', agg = '1h') {
    const res = await fetch(
      `/api/measurements?node_id=${nodeId}&from=${from}&to=${to}` +
      `&sensor=${sensor}&agg=${agg}`
    );
    const data = await res.json();
    set(state => {
      const m = new Map(state.series);
      m.set(nodeId, data);
      return { series: m };
    });
  },
}));

export default useMeasurementsStore;
```

В.3 Компонент часового графіка (src/components/TimeseriesChart.jsx)

```
// src/components/TimeseriesChart.jsx
// ECharts 5.4 — часовий ряд із зумом та двома осями
import { useEffect, useRef } from 'react';
import * as echarts from 'echarts/core';
import { LineChart } from 'echarts/charts';
import { GridComponent, TooltipComponent,
         LegendComponent, DataZoomComponent,
         ToolboxComponent } from 'echarts/components';
import { CanvasRenderer } from 'echarts/renderers';
import useMeasurementsStore from '../store/measurementsStore';

echarts.use([LineChart, GridComponent, TooltipComponent,
             LegendComponent, DataZoomComponent,
             ToolboxComponent, CanvasRenderer]);

const COLORS = {
  pm25: '#E74C3C', pm10: '#E67E22', co2: '#3498DB',
  temp: '#27AE60', rh:   '#9B59B6',
};

export default function TimeseriesChart({ nodeId, sensors = ['pm25','co2'] }) {
  const chartRef = useRef(null);
  const series   = useMeasurementsStore(s => s.getSeries(nodeId));

  useEffect(() => {
    if (!chartRef.current) return;
    const chart = echarts.init(chartRef.current, null, { renderer: 'canvas' });

    const option = {
      animation: false,
      tooltip:   { trigger: 'axis', axisPointer: { type: 'cross' } },
      legend:    { data: sensors },
      toolbox:   { feature: { saveAsImage: {}, dataZoom: {} } },
      dataZoom:  [{ type: 'slider', bottom: 10 }, { type: 'inside' }],
      xAxis: {
        type: 'time',
        axisLabel: { formatter: val =>
          new Date(val).toLocaleTimeString('uk-UA',
            { hour: '2-digit', minute: '2-digit' }) },
      },
      yAxis: sensors.map((s, i) => ({
        type:     'value',
        name:     s.toUpperCase(),
        position: i === 0 ? 'left' : 'right',
        axisLine: { lineStyle: { color: COLORS[s] } },
      })),
      series: sensors.map((s, i) => ({
        name:       s.toUpperCase(),
        type:       'line',
        yAxisIndex: i,
        smooth:     true,
        symbol:     'none',
        lineStyle:  { color: COLORS[s], width: 2 },
        data:       series.map(p => [p.ts * 1000, p[s]]),
      })),
    };
    chart.setOption(option, true);
    return () => chart.dispose();
  }, [series, sensors, nodeId]);

  return (
    <div style={{ width: '100%', height: 320 }} ref={chartRef} />
  );
}
```

В.4 Компонент індикатора AQI (src/components/AQIGauge.jsx)

```
// src/components/AQIGauge.jsx — SVG-кругова шкала AQI
const AQI_LEVELS = [
  { max:  50, color: '#00E400', label: 'Добрий'              },
  { max: 100, color: '#FFFF00', label: 'Задовільний'          },
  { max: 150, color: '#FF7E00', label: 'Шкідл. для вразл.'   },
  { max: 200, color: '#FF0000', label: 'Шкідливий'            },
  { max: 300, color: '#8F3F97', label: 'Дуже шкідливий'      },
  { max: 500, color: '#7E0023', label: 'Небезпечний'          },
];

function getLevel(aqi) {
  return AQI_LEVELS.find(l => aqi <= l.max) || AQI_LEVELS.at(-1);
}

// Перетворення AQI (0-500) у кут (0-270°)
function aqiToAngle(aqi) {
  return Math.min(270, (aqi / 500) * 270);
}

function polarToXY(cx, cy, r, angleDeg) {
  const rad = ((angleDeg - 135) * Math.PI) / 180;
  return { x: cx + r * Math.cos(rad), y: cy + r * Math.sin(rad) };
}

export default function AQIGauge({ aqi = 0, size = 200 }) {
  const level  = getLevel(aqi);
  const angle  = aqiToAngle(aqi);
  const cx = size / 2, cy = size / 2, r = size * 0.38;
  const needle = polarToXY(cx, cy, r - 8, angle);

  return (
    <svg width={size} height={size} viewBox={`0 0 ${size} ${size}`}>
      {/* Фонова дуга */}
      <circle cx={cx} cy={cy} r={r} fill="none"
              stroke="#e0e0e0" strokeWidth={size*0.08}
              strokeDasharray={`${r*1.5*Math.PI} ${r*2*Math.PI}`}
              transform={`rotate(135 ${cx} ${cy})`} />
      {/* Кольорова дуга (прогрес) */}
      <circle cx={cx} cy={cy} r={r} fill="none"
              stroke={level.color} strokeWidth={size*0.08}
              strokeDasharray={`${(angle/270)*r*1.5*Math.PI} ${r*2*Math.PI}`}
              transform={`rotate(135 ${cx} ${cy})`} />
      {/* Стрілка */}
      <line x1={cx} y1={cy} x2={needle.x} y2={needle.y}
            stroke="#333" strokeWidth={2} strokeLinecap="round" />
      {/* Текст AQI */}
      <text x={cx} y={cy + size*0.08} textAnchor="middle"
            fontSize={size*0.18} fontWeight="bold" fill={level.color}>
        {Math.round(aqi)}
      </text>
      <text x={cx} y={cy + size*0.22} textAnchor="middle"
            fontSize={size*0.08} fill="#555">
        AQI
      </text>
      <text x={cx} y={cy + size*0.34} textAnchor="middle"
            fontSize={size*0.07} fill={level.color}>
        {level.label}
      </text>
    </svg>
  );
}
```

В.5 WebSocket-підключення (src/hooks/useRealtimeData.js)

```
// src/hooks/useRealtimeData.js — Socket.IO підписка
import { useEffect } from 'react';
import { io } from 'socket.io-client';
import useMeasurementsStore from '../store/measurementsStore';
import useNodesStore        from '../store/nodesStore';
import useAlertsStore       from '../store/alertsStore';

let socket = null;

export default function useRealtimeData() {
  const appendMeasurement = useMeasurementsStore(s => s.append);
  const updateNode        = useNodesStore(s => s.updateNode);
  const addAlert          = useAlertsStore(s => s.addAlert);

  useEffect(() => {
    if (socket) return;  // єдине з'єднання

    socket = io('/', {
      path: '/socket.io',
      transports: ['websocket'],
      reconnectionDelay:    1000,
      reconnectionDelayMax: 30000,
      randomizationFactor:  0.5,
    });

    socket.on('measurement', ({ node_id, ...point }) => {
      appendMeasurement(node_id, point);
    });
    socket.on('node_status', ({ node_id, status, rssi, bat_pct }) => {
      updateNode(node_id, { status, rssi, bat_pct });
    });
    socket.on('alert', alert => {
      addAlert(alert);
    });

    socket.on('connect', () =>
      console.log('WS connected:', socket.id));
    socket.on('disconnect', reason =>
      console.warn('WS disconnected:', reason));

    return () => {
      socket.disconnect();
      socket = null;
    };
  }, []);
}
```

Наведені фрагменти коду ілюструють ключові архітектурні рішення: таск-орієнтована прошивка FreeRTOS з чергами для роз'єднання збору та передачі; підпис вебхука HMAC-SHA256 для безпечної інтеграції EMQX↔Next.js; Zustand-стор із rolling-window для ефективного управління часовими рядами у React; Webpack 5 з code splitting для оптимального початкового завантаження SPA.