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

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

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

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«СИСТЕМА ЗБОРУ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ З БЕЗДРОТОВИХ СЕНСОРІВ ЯКОСТІ ПОВІТРЯ З ВИКОРИСТАННЯМ MQTT-ПРОТОКОЛУ»

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

Освітня програма ***Автоматизація та комп'ютерно інтегровані технології***

Виконавець _____ **Іван ЛІСАК**
(підпис)

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

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка містить вступ, три розділи, висновки, список використаних джерел та додатки. Загальний обсяг – 81 сторінка, із них 46 сторінок основної частини, 10 рисунків, 4 таблиці, 17 джерел, 4 додатки.

Мета роботи – вдосконалення методів і моделей побудови комп'ютерно-інтегрованої системи моніторингу мікроклімату та якості повітря на основі сенсорів BME680 і PMS5003 шляхом застосування MQTT-архітектури «видавець–підписник», методів обробки часових рядів та інтегрального оцінювання стану середовища.

Об'єкт дослідження – процеси автоматизованого моніторингу параметрів мікроклімату та екологічного стану приміщень із використанням сенсорів BME680 і PMS5003 у складі комп'ютерно-інтегрованої системи.

Предмет дослідження – методи вимірювання екологічних показників (BME680, PMS5003), модель інформаційної взаємодії за протоколом MQTT, трирівнева архітектурна модель системи, методи зберігання часових рядів (InfluxDB), математична модель інтегральної оцінки Health Score та алгоритми гістерезисної обробки даних.

Проблема полягає в усуненні недоліків побутових систем моніторингу шляхом мінімізації хибних тривог та забезпечення ретроспективного аналізу "сирих" показників. Область застосування – системи «розумний дім», екологічний моніторинг, HVAC та освітній процес.

Ключові слова: автоматизація та комп'ютерно-інтегровані технології, комп'ютерно-інтегрована система, моніторинг мікроклімату, BME680, PMS5003, MQTT, InfluxDB, Grafana, Telegram-бот, предиктивний аналіз.

ABSTRACT

The explanatory note contains an introduction, three sections, conclusions, a list of references, and appendices. The total volume is 81 pages, including 46 pages of the main text, 10 figures, 4 tables, 17 sources, and 4 appendices.

The purpose of the work is to improve the methods and models for building a computer-integrated system for monitoring microclimate and air quality based on BME680 and PMS5003 sensors by applying the MQTT publisher-subscriber architecture, time-series processing methods, and integrated environmental condition assessment.

The object of research is the processes of automated monitoring of microclimate parameters and environmental conditions in premises using BME680 and PMS5003 sensors within a computer-integrated system.

The subject of research is the methods of measuring ecological indicators (BME680, PMS5003), the information interaction model using the MQTT protocol, a three-tier system architecture model, time-series storage methods (InfluxDB), the Health Score integrated assessment mathematical model, and hysteresis data processing algorithms.

The problem lies in eliminating the shortcomings of household monitoring systems by minimizing false alarms and ensuring retrospective analysis of raw indicators. Area of application – Smart Home systems, environmental monitoring, HVAC, and the educational process.

Keywords: automation and computer-integrated technologies, computer-integrated system, microclimate monitoring, BME680, PMS5003, MQTT, InfluxDB, Grafana, Telegram bot, predictive analysis.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ СИСТЕМ МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ ТА ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ (IoT) 10
1.1. Проблематика контролю мікроклімату та якості повітря в приміщеннях 10
1.2. Огляд існуючих комерційних та відкритих рішень моніторингу 12
1.3. Аналіз мережевих протоколів для IoT (порівняння HTTP, WebSockets, MQTT) 14
1.4. Обґрунтування вибору програмно-апаратних засобів для розробки 17
Висновки до розділу 1 18
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ AEROPULSE 19
2.1. Розробка загальної структурно-функціональної схеми системи 19
2.2. Апаратна реалізація вимірювального вузла на базі мікроконтролера ESP32 20
2.3. Розгортання та налаштування серверної інфраструктури в середовищі Docker 26
2.4. Локальний інтерфейс користувача та математична модель оцінки якості повітря 28
2.5. Програмна реалізація асинхронного клієнтського застосунку (Telegram-бота) 31
2.6. Алгоритми предиктивного аналізу та числового гістерезису сповіщень 34

Висновки до розділу 2 35

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ 37

3.1. Методика та умови проведення експериментальних досліджень 37

3.2. Дослідження реакції системи на зміну концентрації PM2.5 та VOC 40

3.3. Тестування людино-машинного інтерфейсу та алгоритмів скінченних станів (FSM) 42

3.4. Дослідження довготривалої стабільності серверної інфраструктури та вимірювального вузла 44

3.5. Перспективи комерційного масштабування проєкту 45

Висновки до розділу 3 46

ВИСНОВКИ 48

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50

ДОДАТКИ 52

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних
ПЗ – програмне забезпечення
СУБД – система управління базами даних
API – програмний інтерфейс додатку (Application Programming Interface)
AQI – індекс якості повітря (Air Quality Index)
FSM – машина скінченних станів (Finite State Machine)
GPIO – виводи загального призначення (General-Purpose Input/Output)
HMI – людино-машинний інтерфейс (Human-Machine Interface)
HVAC – автоматизовані системи опалення, вентиляції та кондиціювання (Heating, Ventilation, and Air Conditioning)
I2C – послідовна асиметрична шина зв'язку (Inter-Integrated Circuit)
IAQ – якість внутрішнього повітря (Indoor Air Quality)
IoT – Інтернет речей (Internet of Things)
JSON – текстовий формат обміну даними (JavaScript Object Notation)
MEMS – мікроелектромеханічні системи (Microelectromechanical Systems)
MQTT – мережевий протокол передачі телеметрії (Message Queuing Telemetry Transport)
MVP – мінімально життєздатний продукт (Minimum Viable Product)
PM2.5 – дрібнодисперсні тверді частинки діаметром до 2,5 мкм (Particulate Matter)
QoS – рівень якості обслуговування (Quality of Service)
SaaS – програмне забезпечення як послуга (Software as a Service)
SPI – послідовний периферійний інтерфейс (Serial Peripheral Interface)
TSDB – база даних часових рядів (Time-Series Database)
UART – універсальний асинхронний приймач-передавач (Universal Asynchronous Receiver-Transmitter)
UI/UX – інтерфейс користувача / досвід користувача (User Interface / User Experience)
VOC – леткі органічні сполуки (Volatile Organic Compounds)
IHQ – комплексний індекс стану середовища (Health Quality Index)
SPM – субіндекс концентрації твердих частинок пилу
SVOC – субіндекс концентрації летких органічних сполук

ВСТУП

Актуальність теми. Сучасний етап розвитку автоматизації та комп'ютерно-інтегрованих технологій характеризується масовим впровадженням концепцій Інтернету речей (IoT), кіберфізичних систем та мікросервісних архітектур. Одним із важливих та соціально значущих напрямів застосування цих технологій є побудова автоматизованих систем контролю параметрів навколишнього середовища та екологічного моніторингу всередині житлових, офісних та виробничих приміщень (Indoor Air Quality, IAQ). Оскільки сучасна людина проводить у закритих просторах до 90% свого часу, забезпечення оптимального мікроклімату безпосередньо впливає на її здоров'я, працездатність та загальне самопочуття. Особливу небезпеку становлять такі забруднювачі, як дрібнодисперсний пил фракції PM2.5 та леткі органічні сполуки (VOC), які накопичуються внаслідок використання побутової хімії, деструкції будівельних матеріалів та процесів життєдіяльності.

Більшість існуючих комерційних рішень на ринку є пропрієтарними (Vendor Lock-in). Вони жорстко прив'язані до сторонніх закордонних хмарних серверів, використовують закриті протоколи, не надають доступу до «сирих» масивів даних часових рядів для ретроспективного аналізу та позбавлені гнучких алгоритмів інтелектуального аналізу і фільтрації сповіщень. З іншого боку, існуючі рішення з відкритим кодом часто є розрізненими та вимагають високої технічної кваліфікації на етапі налаштування конвеєрів збору інформації.

Тому проектування та розробка цілісного, бюджетного і водночас відмовостійкого комплексу, що поєднує автономний вимірювальний вузол з локальним інтерфейсом, мікросервісний серверний стек та асинхронну інтелектуальну мобільну підсистему, є актуальним інженерним завданням для спеціальності 151.

Мета роботи – вдосконалення методів і моделей побудови комп'ютерно-інтегрованої системи моніторингу мікроклімату та якості повітря на основі сенсорів BME680 і PMS5003 шляхом застосування MQTT-архітектури «видавець–підписник», методів обробки часових рядів та інтегрального оцінювання стану середовища.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести системний аналіз предметної області та існуючих архітектурних патернів побудови бездротових IoT-мереж екологічного моніторингу.
2. Обґрунтувати вибір апаратних засобів та застосувати методи вимірювання параметрів мікроклімату на основі сенсорів BME680 та PMS5003.
3. Розробити трирівневу модель побудови IoT-системи та імплементувати модель взаємодії компонентів за архітектурою «видавець–підписник» на базі протоколу MQTT.
4. Сформувати модель зберігання та обробки часових рядів (у середовищі InfluxDB), а також реалізувати математичну модель інтегрального оцінювання стану середовища Health Score.
5. Розробити та програмно реалізувати алгоритми предиктивного аналізу, гістерезисної обробки даних і формування асинхронних інформаційних повідомлень для користувача.
6. Провести натурні експериментальні дослідження розробленої комп'ютерно-інтегрованої системи для оцінки її точності, стабільності та ресурсних характеристик.

Об'єкт дослідження – процеси автоматизованого моніторингу параметрів мікроклімату та екологічного стану приміщень із використанням сенсорів BME680 і PMS5003 у складі комп'ютерно-інтегрованої системи.

Предмет дослідження – методи вимірювання екологічних показників (BME680, PMS5003), модель інформаційної взаємодії за протоколом MQTT, трирівнева архітектурна модель системи, методи зберігання часових рядів

(InfluxDB), математична модель інтегральної оцінки Health Score та алгоритми гістерезисної обробки даних.

Методи дослідження. При виконанні роботи використано методи системного аналізу (для проектування трирівневої моделі IoT-системи), теорії цифрових автоматів та скінченних станів (для розробки керуючої логіки інтерфейсу користувача), методи математичного моделювання та зваженого підсумовування (для обробки часових рядів та розрахунку індексу Health Score), а також методи натурного експериментального моделювання та стрес-тестування (для верифікації працездатності системи).

Практичне значення одержаних результатів полягає у створенні повноцінного, відкритого та низькобюджетного прототипу комп'ютерно-інтегрованої системи моніторингу «AeroPulse IoT», що базується виключно на ПЗ з відкритим кодом. Розроблена математична модель розрахунку індексу здоров'я середовища у поєднанні з алгоритмами гістерезисної фільтрації сповіщень дозволяє ефективно впроваджувати комплекс у житлових, навчальних та офісних приміщеннях.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ СИСТЕМ МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ ТА ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ (IoT)

1.1. Проблематика контролю мікроклімату та якості повітря в приміщеннях

В останні десятиліття процеси глобальної урбанізації та зміни в підходах до будівництва енергоефективних (герметичних) будівель призвели до виникнення проблеми так званого «синдрому хворої будівлі» (Sick Building Syndrome) [1]. За даними Всесвітньої організації охорони здоров'я (ВООЗ), сучасна людина проводить у закритих приміщеннях до 80-90% свого часу. Відповідно, якість внутрішнього повітря (Indoor Air Quality, IAQ) має критичний вплив не лише на фізичне здоров'я, але й на когнітивні здібності та продуктивність праці [2].

Основними забруднювачами, що вимагають безперервного апаратного моніторингу в побутових та офісних умовах, є дрібнодисперсний пил (PM2.5) та леткі органічні сполуки (VOC).

Частинки PM2.5 – це тверді мікрочастинки та краплі рідини розміром менше 2,5 мікрометрів. Через свій мікроскопічний розмір вони здатні долати біологічні бар'єри дихальної системи людини і потрапляти безпосередньо у кровоносну систему, викликаючи серцево-судинні захворювання та алергічні реакції [3]. Для розуміння масштабу, діаметр такої частинки становить близько 3% від діаметра людської волосини.

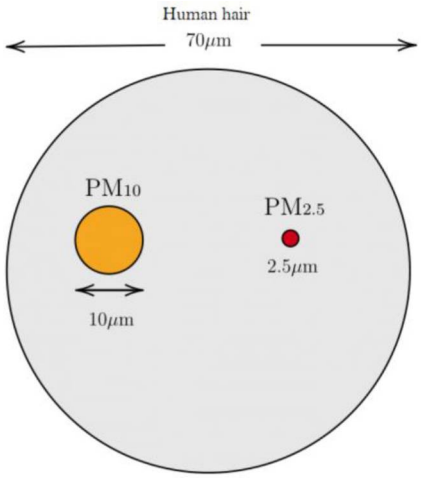


Рисунок 1.1 – Порівняння розмірів частинок PM2.5 з людською волосиною та піщинкою [6].

Леткі органічні сполуки (VOC) являють собою групу вуглеводневих газів, які виділяються з будівельних матеріалів, меблів, миючих засобів, а також накопичуються внаслідок життєдіяльності людини у погано провітрюваних приміщеннях [4]. Найбільш поширеними та небезпечними серед них є токсичні пари формальдегіду, бензолу та толуолу, тривалий вплив яких чинить канцерогенний ефект на організм. Висока концентрація VOC викликає головний біль, подразнення слизових оболонок, зниження концентрації уваги та відчуття втоми. Оскільки ці гази часто не мають вираженого запаху, людина не здатна самостійно об'єктивно оцінити їх рівень, що робить застосування електронних газових сенсорів (наприклад, металооксидних) єдиним надійним методом контролю.

Окрім хімічного та дисперсного складу, базовими параметрами мікроклімату залишаються температура та відносна вологість. Їх відхилення від санітарних норм не лише створює дискомфорт, але й провокує розмноження цвілевих грибів (при вологості >70%) або пересихання слизових оболонок дихальних шляхів (при вологості <30%) [5]. Додатково, різкі температурні коливання здатні порушити тепловий баланс тіла та знизити загальний імунітет. Нормативні показники, на які

спирається алгоритм предиктивного аналізу розробленої системи AeroPulse для прогнозування несприятливих тенденцій зміни мікроклімату, наведено у таблиці 1.1.

Таблиця 1.1

Рекомендовані норми параметрів мікроклімату для житлових приміщень

Параметр	Оптимальне значення	Допустиме значення (Попередження)	Критичне значення (Небезпека)
Температура повітря, °C	20 – 25	18 – 20 або 25 – 28	< 18 або > 28
Відносна вологість, %	40 – 60	30 – 40 або 60 – 70	< 30 або > 70
Пил (PM2.5), мкг/м³	0 – 15	15 – 35	> 35
Гази (VOC), індекс / kΩ	Високий опір (>100)	Середній опір (40 – 100)	Низький опір (<40)

Класичні системи вентиляції часто діють за розкладом або керуються користувачем вручну вже після того, як відчувається дискомфорт. Тому сучасна парадигма автоматизації будівель вимагає переходу до проактивного (предиктивного) моніторингу.

1.2. Огляд існуючих комерційних та відкритих рішень моніторингу

Ринок пристроїв для моніторингу якості повітря всередині приміщень (Indoor Air Quality, IAQ) на сьогоднішній день є досить насиченим. Домінуючі позиції займають готові комерційні рішення від таких компаній, як Xiaomi, Awair, Qingping, Philips та AirVisual. Ці пристрої позиціонуються як рішення класу «plug-and-play», що орієнтовані на масового споживача, мають привабливий дизайн та

базові мобільні додатки для перегляду поточних показників. Однак, незважаючи на зручність використання, комерційним IAQ-моніторам притаманний ряд суттєвих недоліків з точки зору системної інтеграції та аналітики:

1. Закриті екосистеми (Vendor Lock-in). Більшість виробників використовують пропрієтарні протоколи передачі даних. Пристрої жорстко прив'язані до хмарних серверів компанії-розробника (часто розташованих за кордоном), що створює ризики конфіденційності та унеможливлює роботу системи у локальній мережі без доступу до Інтернету.

2. Обмежений доступ до сирих даних. Комерційні додатки зазвичай показують усереднені значення або абстрактні індекси якості повітря (AQI). Отримання необроблених даних (raw data) для глибокого ретроспективного аналізу або експорт масивів у професійні платформи (наприклад, Grafana) часто неможливий або вимагає платної підписки. До того ж закрита архітектура унеможливлює програмне перекалібрування сенсорів користувачем у процесі їх природного старіння та деградації.

3. Відсутність гнучкої логіки сповіщень. Стандартні додатки генерують тривоги за жорстко заданими порогами. Вони не підтримують алгоритми гістерезису або аналіз трендів (швидкість зміни показника за одиницю часу), що призводить до проблеми інформаційного перевантаження (спаму) користувача під час незначних коливань показників на межі норми.

Альтернативою є розробка відкритого (Open-Source) IoT-рішення на базі мікроконтролерів (зокрема, ESP32) та стандартизованих протоколів (MQTT). Такий підхід дозволяє абстрагуватися від конкретних брендів, повністю розгорнути інфраструктуру збору телеметрії на локальному сервері за допомогою Docker-контейнеризації та побудувати систему, в якій користувач є єдиним повноправним власником даних. Порівняльний аналіз характеристик комерційних рішень та розробленої комп'ютерно-інтегрованої системи AeroPulse наведено у таблиці 1.2.

Таблиця 1.2

Порівняльна характеристика рішень моніторингу якості повітря

Характеристика	Комерційні рішення (Smart Home)	Розроблена система (AeroPulse)
Передача даних	Пропрієтарні HTTP/API	Стандартизований MQTT протокол
Зберігання даних	Хмарні сервери виробника	Власна БД часових рядів (InfluxDB v2.7)
Інтерфейси (HMI)	Обмежений мобільний додаток	Локальний TFT-дисплей + Telegram-бот
Візуалізація та аналітика	Базові графіки в додатку	Професійна аналітика у Grafana
Логіка сповіщень	Статичні пороги спрацювання	Предиктивний аналіз трендів та гістерезис

1.3. Аналіз мережевих протоколів для IoT (порівняння HTTP, WebSockets, MQTT)

Вибір оптимального мережевого протоколу прикладного рівня є критичним етапом проєктування систем Інтернету речей. Оскільки кінцеві вузли (в даному випадку мікроконтролер ESP32) працюють у середовищі з потенційно нестабільним бездротовим зв'язком та мають обмежені обчислювальні ресурси, класичні вебпротоколи часто виявляються неефективними [9].

Традиційний протокол HTTP (Hypertext Transfer Protocol) базується на архітектурі «клієнт-сервер» та синхронній моделі «запит-відповідь». Кожне відправлення даних сенсорів через HTTP вимагає встановлення нового TCP-з'єднання та передачі об'ємних HTTP-заголовків, що значно збільшує обсяг мережевого трафіку та час обробки. Крім того, HTTP не підтримує двонаправлену

ініціалізацію зв'язку: сервер не може самостійно надіслати команду на мікроконтролер без попереднього запиту від самого пристрою.

Протокол WebSockets частково вирішує цю проблему, забезпечуючи повнодуплексний зв'язок через єдине довготривале TCP-з'єднання. Це дозволяє передавати дані в режимі реального часу. Проте WebSockets залишається ресурсомістким для мікроконтролерів і не має вбудованих механізмів гарантованої доставки повідомлень при обривах зв'язку.

Найбільш адаптованим стандартом для IoT-систем визнано протокол MQTT (Message Queuing Telemetry Transport). На відміну від HTTP, MQTT використовує асинхронну архітектуру «видавець-підписник» (Publish-Subscribe) та центральний сервер маршрутизації – брокер повідомлень (рис. 1.2).

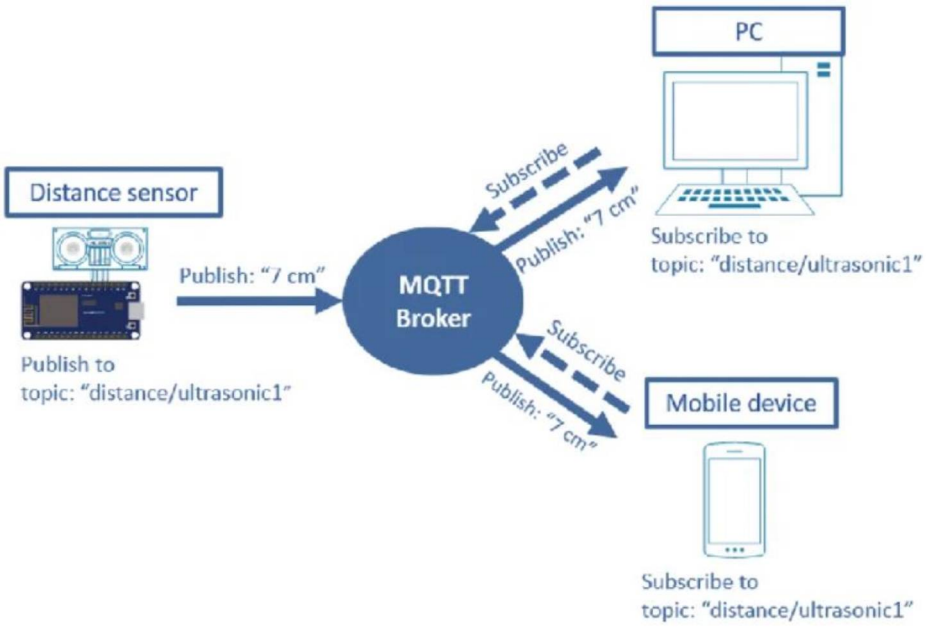


Рисунок 1.2 – Архітектура обміну повідомленнями за протоколом MQTT [10].

У цій архітектурі вимірювальний вузол (ESP32) виступає в ролі видавця (Publisher), відправляючи JSON-пакети з телеметрією у визначений топік (наприклад, diploma/sensors/air). Серверна частина, включаючи Telegram-бота та базу даних, виступає підписником (Subscriber). Брокер бере на себе всю роботу з розподілу повідомлень. Якщо підписник тимчасово офлайн, брокер може зберегти повідомлення і доставити його пізніше [11].

Головною перевагою MQTT є мінімальний розмір заголовка пакета (всього 2 байти) та підтримка трьох рівнів якості обслуговування (QoS – Quality of Service):

- QoS 0 (At most once) – відправлення без підтвердження («вистрілив і забув»).
- QoS 1 (At least once) – гарантована доставка з підтвердженням, але можливе дублювання.
- QoS 2 (Exactly once) – гарантована доставка без дублікатів (найвища надійність).

У таблиці 1.3 наведено порівняльний аналіз протоколів HTTP та MQTT в контексті вимог розроблюваної системи.

Таблиця 1.3

Порівняльний аналіз протоколів HTTP та MQTT для IoT

Критерій порівняння	HTTP (REST API)	MQTT
Архітектурний шаблон	Запит - Відповідь (Синхронно)	Видавець - Підписник (Асинхронно)
Розмір заголовка пакета	Великий (сотні байт)	Мінімальний (від 2 байт)
Витрати енергії та ресурсів	Високі	Низькі (оптимізовано для мікроконтролерів)
Механізм гарантованої доставки	Відсутній (лише статуси помилок TCP)	Вбудований (QoS 0, 1, 2)

Продовження таблиці 1.3

Критерій порівняння	HTTP (REST API)	MQTT
Підтримка обривів зв'язку	Потребує повторного перепідключення	Збереження сесії (Persistent Session, LWT)
Широкомовна розсилка (1 до багатьох)	Неможлива (вимагає N окремих запитів)	Вбудована (через механізм топіків)

З огляду на наведені характеристики, для реалізації програмно-апаратного комплексу AeroPulse обрано протокол MQTT. Він забезпечує мінімальну затримку передачі даних, стійкість до обривів Wi-Fi з'єднання та легку інтеграцію з іншими мікросервісами в екосистемі Docker.

1.4. Обґрунтування вибору програмно-апаратних засобів для розробки

Створення IoT-рішення вимагає підбору апаратних компонентів та програмного стеку з урахуванням критеріїв інтеграції, енергоефективності та вартості.

Обчислювальним ядром вузла обрано мікроконтролер ESP32 завдяки наявності вбудованого Wi-Fi модуля та достатньої продуктивності для паралельної обробки даних. Сенсорний комплекс включає MEMS-сенсор BME680 (шина I2C) для контролю клімату та летких органічних сполук (VOC), а також лазерний лічильник PMS5003 (UART) для точного вимірювання дрібнодисперсного пилу PM2.5.

Локальний людино-машинний інтерфейс (HMI) реалізовано на базі кольорового TFT-дисплея ILI9341 (шина SPI), світлодіодного трирівневого «світлофора» та єдиної тактової кнопки з гібридною логікою роботи. Це забезпечує візуальну інформативність та ергономічність пристрою.

Серверна інфраструктура базується на мікросервісній архітектурі з використанням Docker Compose, що забезпечує незалежність від апаратного середовища. До стеку входять: MQTT-брокер Eclipse Mosquitto, оптимізована база даних часових рядів InfluxDB (версії 2.7) та платформа аналітики Grafana. Клієнтським застосунком виступає асинхронний Telegram-бот (Python, фреймворк aiogram) із підтримкою машин скінченних станів (FSM) та алгоритмів предиктивного сповіщення.

Висновки до розділу 1

У першому розділі проаналізовано проблематику моніторингу якості повітря (IAQ) та доведено критичну необхідність контролю параметрів PM2.5, VOC і мікроклімату. Огляд існуючих комерційних рішень виявив їхні суттєві недоліки (закритість екосистем, відсутність гнучкої логіки сповіщень та доступу до сирих даних), що повністю обґрунтовує потребу в розробці відкритої архітектури системи моніторингу.

На основі порівняльного аналізу мережевих протоколів доведено перевагу асинхронного стандарту MQTT над HTTP для IoT-систем завдяки патерну «видавець-підписник», низькому енергоспоживанню та вбудованим механізмам гарантування доставки (QoS).

Обґрунтовано вибір апаратної бази для вимірювального вузла (мікроконтролер ESP32, сенсори BME680 та PMS5003, локальний TFT-інтерфейс) та відкритого серверного стеку (Docker, Mosquitto, InfluxDB, Grafana, Telegram-бот). Обраний технологічний комплекс формує надійний фундамент для практичного проєктування та реалізації системи у другому розділі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ AEROPULSE

2.1. Розробка загальної структурно-функціональної схеми системи

Основою для успішної реалізації будь-якої складної системи Інтернету речей (IoT) є побудова чіткої структурно-функціональної моделі, яка визначає ієрархію компонентів, розподіл обчислювальних навантажень та напрямки потоків даних. Розроблений програмно-апаратний комплекс AeroPulse має багаторівневу топологію, що логічно поділяється на три основні рівні: рівень периферійного збору даних (Edge Tier), рівень серверної маршрутизації та зберігання (Platform Tier), а також рівень клієнтського застосунку (Enterprise/Application Tier).

Загальну структурно-функціональну схему розробленої системи наведено на рисунку 2.1.

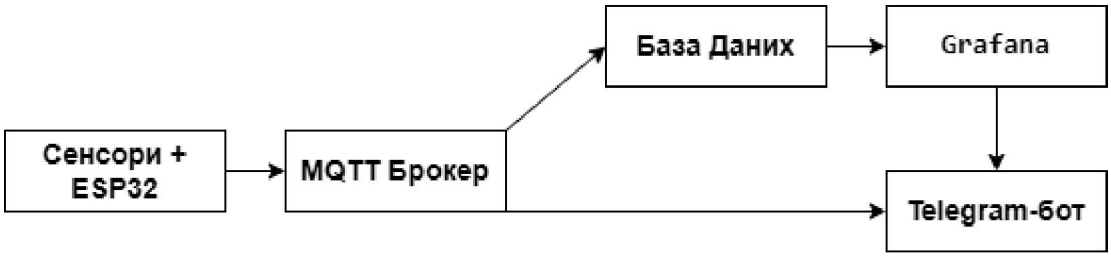


Рисунок 2.1 – Загальна структурно-функціональна схема комплексу AeroPulse.

Рівень периферійного збору даних (Edge Tier) представлений автономним апаратним вимірювальним вузлом на базі 32-бітного мікроконтролера ESP32. До ядра системи за допомогою цифрових шин I2C та UART підключено сенсори екологічних параметрів: BME680 та PMS5003. Окрім збору телеметрії, цей рівень відповідає за первинну математичну обробку даних (розрахунок Health Score) та забезпечення локального людино-машинного інтерфейсу за допомогою TFT-дисплея, світлодіодної індикації та апаратної кнопки. Передача сформованих JSON-

пакетів до хмарного середовища здійснюється по бездротовій мережі Wi-Fi з використанням протоколу MQTT.

Рівень серверної інфраструктури (Platform Tier) розгорнуто з використанням технології контейнеризації Docker Compose у єдиній віртуальній мережі (bridge network). Центральним диспетчером виступає брокер повідомлень Eclipse Mosquitto. Для забезпечення надійного збереження часових рядів розгорнуто СУБД InfluxDB версії 2.7. Інтеграція між брокером та базою даних реалізована через сервісний агент Telegraf, який автоматично підписується на MQTT-топік, здійснює парсинг JSON-пакетів та конвертує їх у Line Protocol для запису в БД. Візуалізація та агрегація накопичених метрик виконується в аналітичній платформі Grafana за допомогою спеціалізованого модуля рендерингу (Image Renderer).

Рівень клієнтського застосунку (Application Tier) реалізовано у вигляді серверного Python-скрипта (Telegram-бота). Бот асинхронно взаємодіє з MQTT-брокером для отримання поточних метрик і паралельно обробляє користувацькі запити через інтерфейс месенджера Telegram. Використання машини скінченних станів (FSM) дозволяє боту формувати складні API-запити до Grafana, а впроваджені алгоритми предиктивного аналізу трендів і числового гістерезису відповідають за генерацію інтелектуальних Push-сповіщень про критичні зміни мікроклімату.

2.2. Апаратна реалізація вимірювального вузла на базі мікроконтролера ESP32

Апаратний вимірювальний вузол (Edge Device) є первинною ланкою IoT-системи, що відповідає за безпосередній збір фізичних показників навколишнього середовища, їх первинну математичну обробку та оцифрування. До апаратної частини висуваються жорсткі вимоги щодо стабільності функціонування у цілодобовому режимі 24/7, точності вимірювань, низького енергоспоживання та здатності підтримувати захищене бездротове з'єднання в умовах побутових завад.

Ядром розробленого апаратного вузла обрано налагоджувальну плату на базі мікроконтролера ESP32 (модель ESP-WROOM-32). Вибір цієї платформи обґрунтований наявністю високопродуктивного двоядерного 32-бітного процесора Xtensa LX6 з тактовою частотою 240 МГц та обсягом оперативної пам'яті 520 КБ SRAM. Це забезпечує достатню обчислювальну потужність для паралельного асинхронного виконання завдань: потокового зчитування масивів даних із сенсорів, динамічного оновлення графічних елементів локального екрана та підтримки постійного TCP-з'єднання з MQTT-брокером. Вбудований радіомодуль Wi-Fi (стандарт 802.11 b/g/n) дозволяє повністю відмовитися від використання зовнішніх мережевих плат, що суттєво зменшує габарити пристрою та його кінцеву вартість.

Для отримання комплексних та достовірних даних про екологічний стан і мікроклімат приміщення до обчислювального ядра підключено два спеціалізовані цифрові сенсорні модулі:

1. Екологічний сенсор BME680 (Bosch Sensortec). Це високоточний датчик класу інтегрованих кіберфізичних компонентів 4-в-1, який одночасно вимірює температуру повітря, відносну вологість, атмосферний тиск та еквівалентний опір летких органічних сполук (VOC). Зв'язок із мікроконтролером здійснюється по синхронній цифровій шині I2C (Inter-Integrated Circuit) за адресою 0x76. Використання шини I2C дозволяє підключати кілька периферійних пристроїв, використовуючи лише дві лінії даних: SDA (Serial Data) та SCL (Serial Clock), що заощаджує виводи загального призначення (GPIO) мікроконтролера.
2. Лазерний датчик пилу PMS5003 (Plantower). Цей сенсор використовує принцип лазерного розсіювання (оцінка оптичної щільності середовища) для точного підрахунку кількості зважених у повітрі частинок діаметром понад 0.3 мкм, обчислюючи масову концентрацію дрібнодисперсного пилу фракції PM2.5. Датчик PMS5003 передає дані через асинхронний інтерфейс UART (Universal Asynchronous Receiver-Transmitter). Він підключений до другого

апаратного порту UART2 мікроконтролера ESP32, що гарантує безперебійне фонове зчитування бінарного потоку даних без затримок і блокування обчислень.

Важливою архітектурною особливістю вимірювального вузла системи AeroPulse є наявність повноцінного локального людино-машинного інтерфейсу (HMI), який забезпечує автономність пристрою у разі тимчасової відсутності мережевого зв'язку. Локальний інтерфейс побудовано на базі рідкокристалічного кольорового TFT-дисплея з драйвером ILI9341, що підключений до високошвидкісної апаратної шини VSPI мікроконтролера.

Для експрес-діагностики стану середовища без зчитування дрібних числових значень інтегровано трирівневий світлодіодний «світлофор» (зелений, жовтий та червоний індикатори), які безпосередньо відображають поточну зону безпеки розрахованого індексу здоров'я повітря. Керування режимами відображення покладено на одну апаратну тактову кнопку, яка працює за гібридною логікою (коротке натискання змінює інформаційні сторінки екрана, а тривале — активує процедуру повного скидання налаштувань).

Структурну схему апаратного підключення периферійних компонентів та інтерфейсів вимірювального вузла наведено на рисунку 2.2.

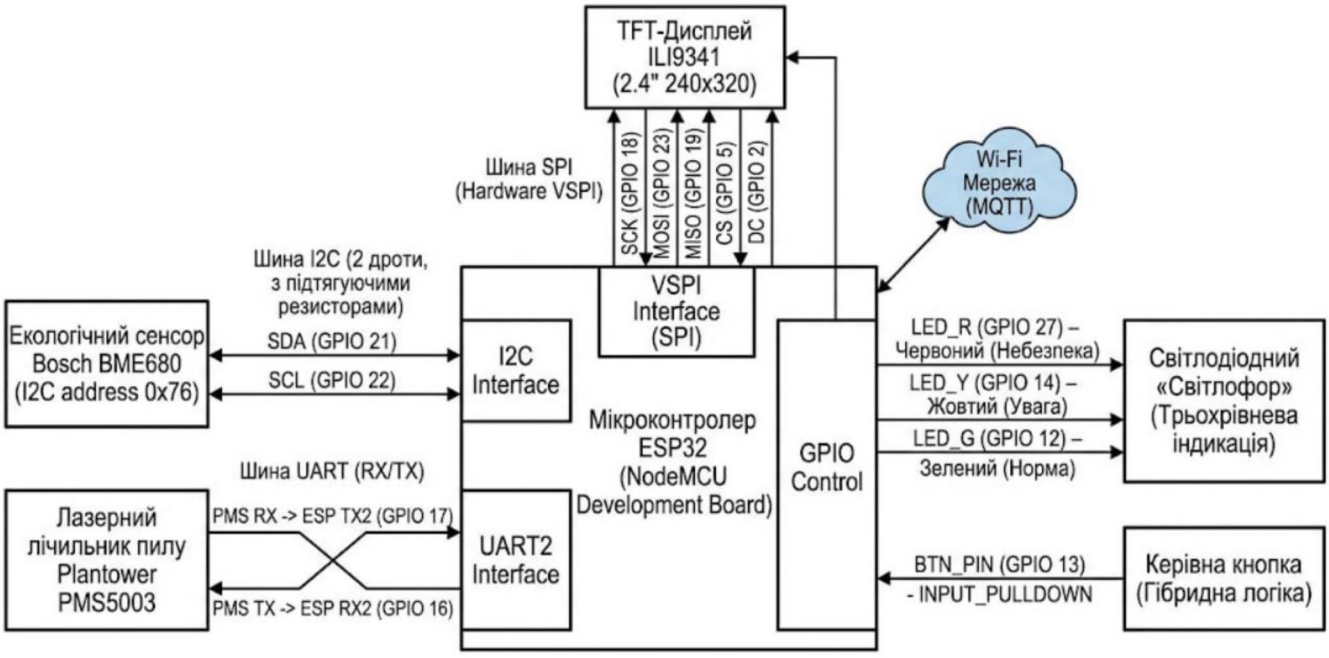


Рисунок 2.2 – Структурна схема підключення периферії до мікроконтролера ESP32.

Точну схему розподілу сигналів, типів ліній зв'язку та конфігурацію виводів мікроконтролера для розробленого апаратного вузла наведено у таблиці 2.1.

Таблиця 2.1

Розподіл сигналів, типів ліній зв'язку та конфігурація виводів мікроконтролера ESP32

Назва компонента периферії	Пін ESP32	Назва виводу	Тип сигналу	Апаратний інтерфейс та роль
Екологічний сенсор BME680	GPIO 21 GPIO 22	SDA SCL	Цифровий (двонаправлений) Цифровий (вихід)	Синхронна шина I2C, зчитування основних параметрів клімату та газів VOC

Продовження таблиці 2.1

Датчик пилу PMS5003	GPIO 16 GPIO 17	RX2 TX2	Цифровий (вхід) Цифровий (вихід)	Асинхронний інтерфейс UART2, отримання бінарних фреймів PM2.5
Керівна тактова кнопка	GPIO 13	BTN_PIN	Цифровий (вхід)	Вхід зі специфікацією INPUT_PULLDOWN для апаратного перемикання та ресету
Світлодіодний світлофор	GPIO 12 GPIO 14 GPIO 27	LED_G LED_Y LED_R	Цифровий (вихід) Цифровий (вихід) Цифровий (вихід)	Дискретні виходи для світлової сигналізації рівня безпеки середовища
Керування інфраструктурою	GPIO 32	POWER_EN	Цифровий (вихід)	Активація транзисторного ключа живлення або підсвічування екрана
TFT-дисплей ILI9341	GPIO 18 GPIO 23 GPIO 19 GPIO 5 GPIO 2	SCK MOSI MISO TFT_CS TFT_DC	Цифровий (вихід) Цифровий (вихід) Цифровий (вхід) Цифровий (вихід) Цифровий (вихід)	Високошвидкісна апаратна шина VSPI для динамічного рендерингу графіки

Для практичного налагодження апаратної взаємодії між мікроконтролером, цифровими сенсорами та периферією було попередньо зібрано лабораторний макет пристрою. Це дозволило верифікувати коректність підключення шин I2C (для датчика BME680) та UART (для лічильника PMS5003), а також протестувати

роботу TFT-дисплея та світлодіодної індикації до моменту фінального монтажу компонентів у корпус. Зовнішній вигляд зібраного макета в активному процесі тестування виводу телеметрії наведено на рисунку 2.3.

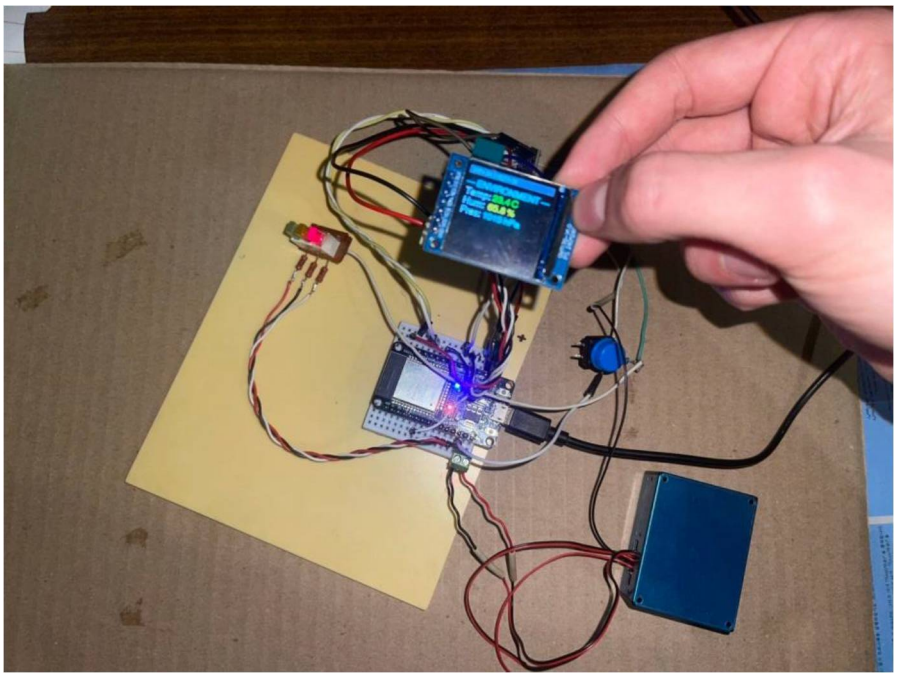


Рисунок 2.3 – Зовнішній вигляд лабораторного макета вимірювального вузла в процесі апаратного налагодження.

Вбудоване програмне забезпечення (прошивка) для мікроконтролера розроблено на мові C++ з використанням фреймворку Arduino Core для ESP32. Важливою інженерною особливістю коду є повна відмова від блокуючих інструкцій очікування (таких як функція delay()). Логіка програми побудована на базі неблокуючих інтервалів із використанням системної функції millis(). Такий підхід дозволяє реалізувати кооперативну багатозадачність на рівні одного процесорного ядра, коли тривала операція (наприклад, прогрів нагрівального елемента датчика VOC чи очікування відповіді від мережі) не зупиняє зчитування показників пилу та обробку натискань кнопки.

Алгоритм роботи вбудованого програмного забезпечення є циклічним і базується на неблокуючих таймерах millis(). Він складається з таких ключових фаз:

1. Ініціалізація та прогрів: Конфігурування інтерфейсів (I2C, UART, SPI), перевірка зв'язку з BME680 та подача напруги на його нагрівальну пластину для виходу на робочий режим.
2. Динамічне підключення: Використання бібліотеки WiFiManager для підключення до мережі через Captive Portal без жорсткого «хардкодингу» паролів.
3. Опитування та обчислення (кожні 1 с): Зчитування даних із сенсорів, розрахунок комплексного індексу Health Score, оновлення TFT-екрана та стану світлодіодної індикації.
4. Серіалізація та публікація (кожні 15 с): Упаковка масиву виміряних даних у JSON-формат та їх відправлення на MQTT-брокер у топік diploma/sensors/air.

У разі втрати Wi-Fi з'єднання або падіння MQTT-сесії мікроконтролер автоматично ініціює фонове перепідключення, не переривaючи локальний збір даних та роботу дисплея, що робить вимірювальний вузол повністю відмовостійким. Повний лістинг прошивки наведено у Додатку Г.

2.3. Розгортання та налаштування серверної інфраструктури в середовищі Docker

Традиційний монолітний підхід до розробки серверного програмного забезпечення, за якого всі компоненти системи (вебсервер, база даних, брокер) встановлюються безпосередньо на хост-операційну систему, має суттєві недоліки. До них належать складнощі при перенесенні проєкту на інше обладнання, ризики конфліктів версій залежностей бібліотек та висока вразливість до загальносистемних збоїв.

Для вирішення цих проблем серверну інфраструктуру (Platform Tier) комплексу AeroPulse спроєктовано на базі мікросервісної архітектури з використанням технології контейнеризації Docker. Кожен логічний вузол системи працює в ізольованому середовищі (контейнері). Оркестрація контейнерів здійснюється за допомогою утиліти Docker Compose, яка об'єднує їх у єдину

віртуальну мережу (bridge network) під назвою aeropulse-net. Взаємодія компонентів описується декларативним шляхом у конфігураційному файлі docker-compose.yml (повний лістинг наведено у Додатку А).

До складу серверного ядра системи входять п'ять взаємопов'язаних мікросервісів:

1. MQTT-брокер Eclipse Mosquitto. Виконує роль центрального диспетчера мережі. Він не зберігає дані довгостроково, а забезпечує їх миттєву маршрутизацію між видавцем (ESP32) та підписниками. Контейнер налаштовано на прослуховування стандартного порту 1883. Для забезпечення надійності в конфігурації увімкнено механізм персистенції (persistence true), що дозволяє брокеру зберігати невідправлені повідомлення на диск у разі раптового перезавантаження сервера.
2. База даних InfluxDB (версії 2.7). Оскільки дані екологічного моніторингу мають яскраво виражену хронологічну прив'язку (кожен запис — це мітка часу та числове значення), використання класичних реляційних баз даних (наприклад, PostgreSQL чи MySQL) є неефективним через повільний пошук за часовими індексами. Тому використано спеціалізовану базу даних часових рядів (Time-Series Database, TSDB). InfluxDB v2.7 оптимізована для надшвидкого запису великих масивів телеметрії (Append-Only) та агрегаційних запитів. На відміну від попередніх версій, версія 2.7 використовує безпечну токен-авторизацію (Admin Token), логічні простори Organization (AeroPulse) та Bucket (air_quality) для ізоляції даних, що є необхідним кроком для майбутнього комерційного масштабування.
3. Сервісний агент Telegraf. Виконує роль програмного мосту між MQTT-брокером та базою даних. Прямий запис даних з мікроконтролера в базу InfluxDB вимагав би формування складних HTTP-запитів на рівні ESP32, що невиправдано перевантажило б його ресурси. Замість цього агент Telegraf працює у фоновому режимі на сервері, автоматично підписується на топік,

розбирає вхідний JSON-пакет і трансформує його у лінійний протокол (Line Protocol), виконуючи безпечний запис в InfluxDB за допомогою авторизаційного токена.

4. Аналітична платформа Grafana. Підключається до InfluxDB за допомогою вбудованого конектора та мови запитів Flux. У середовищі Grafana створюється комплексний інформаційний дашборд, який агрегує метрики та будує інтерактивні графіки зміни температури, вологості, концентрації дрібнодисперсного пилу PM2.5 та індексу летких органічних сполук (VOC) у реальному часі.
5. Grafana Image Renderer. Додатковий мікросервіс, що функціонує на базі браузерного рушія Chromium у режимі «без графічного інтерфейсу» (headless). Він необхідний для конвертації динамічних графіків Grafana у статичні растрові PNG-зображення (через порт 8081). Це критично важливий компонент для Telegram-бота, оскільки клієнтські програми месенджерів не підтримують рендеринг інтерактивних HTML/JavaScript вебсторінок.

Для забезпечення стійкості до збоїв (High Availability) усі контейнери налаштовано з параметром перезапуску restart: unless-stopped. Використання іменованих томів (Docker Volumes), таких як influxdb_data та grafana_data, гарантує, що накопичена база вимірювань і налаштування дашбордів не будуть втрачені навіть у разі повного зупинення, видалення або оновлення самих контейнерів.

2.4. Локальний інтерфейс користувача та математична модель оцінки якості повітря

Важливою вимогою до сучасних систем автоматизації та Інтернету речей у рамках спеціальності 151 є проектування ефективних засобів людино-машинної взаємодії (HMI) безпосередньо на периферійному пристрої. Це забезпечує повну автономність функціонування вимірювального вузла «AeroPulse» незалежно від наявності зв'язку із хмарним сервером чи доступності мобільного додатка.

Локальний інтерфейс розробленого пристрою базується на поєднанні двох підсистем відображення інформації та одного універсального органу керування:

1. Графічна підсистема: Кольоровий рідкокристалічний TFT-дисплей діагоналлю 2.4 дюйма на базі драйвера ILI9341 (роздільна здатність 240 на 320 пікселів). Обмін графічними даними здійснюється по високошвидкісній апаратній шині SPI (інтерфейс VSPI мікроконтролера ESP32). Програмна логіка реалізує дві інформаційні сторінки («Environment» для кліматичних параметрів та «Air Quality» для концентрації забруднювачів і загального індексу), перемикання між якими відбувається миттєво шляхом повної очистки та перемальовування екрана через функцію tft.fillScreen(TFT_BLACK).
2. Дискретна світлова сигналізація («Світлофор»): Три над'ясні світлодіоди (зелений, жовтий, червоний), підключені до виводів GPIO 12, 14 та 27 відповідно. Ця підсистема дублює критичні стани пристрою і дозволяє користувачеві миттєво, на відстані, оцінити безпеку повітряного середовища в кімнаті без необхідності підходити та зчитувати дрібний шрифт із TFT-екрана, що повністю відповідає принципам ергономіки засобів автоматизації.
3. Гібридний елемент керування: Одна тактова арараатна кнопка на піні GPIO 13. Для економії місця на корпусі та спрощення конструкції, у прошивці реалізовано асинхронну гібридну логіку обробки натискань. Програма використовує неблокуючі таймери для диференціації короткого кліку (менше 1 с) для гортання сторінок дисплея та тривалого утримання. Якщо кнопку утримувати понад 1 с, пристрій переходить у режим фабричного скидання (isResetCountdown = true), запускаючи на екрані графічний зворотний відлік від 5 до 1 секунди. Утримання кнопки сумарно протягом 5 секунд викликає повне очищення енергонезалежної пам'яті NVS (wm.resetSettings()) та перезавантаження, що дозволяє скинути налаштування Wi-Fi без перепрошивки заліза.

Для керування станом локального людино-машинного інтерфейсу та світлодіодного «світлофора» було розроблено математичну модель інтегральної оцінки стану повітряного середовища. Оскільки фізичні величини концентрації дрібнодисперсного пилу PM2.5 (вимірюється в мкг/м³) та опору напівпровідникового шару сенсора VOC (вимірюється в кОм) мають різну розмірність і протилежні вектори небезпеки (рост концентрації пилу — це погано, а рост опору датчика газів — це добре), виникла необхідність їх нормування та зведення до єдиної 100-бальної шкали індексу здоров'я (Health Score).

На першому етапі алгоритм обчислює два безрозмірні субіндекси для кожного типу забруднювача окремо, використовуючи математичний апарат лінійної інтерполяції (мапування) у межах санітарних норм.

Субіндекс твердих частинок S_{PM} розраховується за формулою зворотної пропорції в діапазоні від 0 до 50 мкг/м³:

$$S_{PM} = \begin{cases} 100 - (C_{PM}/50 \cdot 100), & \text{якщо } 0 \le C_{PM} \le 50 \\ 0, & \text{якщо } C_{PM} > 50 \end{cases}$$

де C_{PM} — виміряна лазерним датчиком PMS5003 поточна маса мікрочастинок PM2.5 у повітрі.

Субіндекс летких органічних сполук S_{VOC} обчислюється на основі прямої лінійної інтерполяції електричного опору напівпровідника в межах від 10 кОм (максимальне хімічне забруднення) до 150 кОм (абсолютно чисте еталонне повітря):

$$S_{VOC} = \begin{cases} 0, & \text{якщо } R_{VOC} < 10 \\ \dfrac{R_{VOC} - 10}{140} \cdot 100, & \text{якщо } 10 \le R_{VOC} \le 150 \\ 100, & \text{якщо } R_{VOC} > 150 \end{cases}$$

де R_{VOC} — виміряний газоаналізатором BME680 опір датчика, виражений у кілоомах (sys.gas / 1000.0).

На другому етапі математичної обробки здійснюється обчислення фінального комплексного індексу якості повітря I_{HQ} (Health Quality Index). Якщо застосувати

звичайне середнє арифметичне, виникне небезпечний архітектурний дефект: якщо один із показників буде в ідеальній нормі (100 балів), а інший — на критично небезпечному рівні (0 балів), середній індекс покаже 50 балів (задовільно), що дезінформує користувача.

Для уникнення цього ефекту в системі AeroPulse застосовано консервативну математичну модель зваженого мінімуму. Вона надає 70% вагового коефіцієнта найгіршому субіндексу із двох, і лише 30% — їхньому спільному середньому значенню:

$$I_{HQ} = 0.7 \cdot \min(S_{PM}, S_{VOC}) + 0.3 \cdot \frac{S_{PM} + S_{VOC}}{2}$$

Розраховане за формулою I_{HQ} числове значення комплексного індексу відсоткового стану здоров'я середовища передається у логічний блок керування апаратними виходами GPIO за наступним алгоритмом прийняття рішень:

$$\text{Активний } LED = \begin{cases} LED_G\ (GPIO\ 12), & \text{якщо } I_{HQ} > 75 \\ LED_Y(GPIO\ 14), & \text{якщо } 40 < I_{HQ} \leq 75 \\ LED_R(GPIO\ 27), & \text{якщо } I_{HQ} \leq 40 \end{cases}$$

Дана математична модель повністю інтегрована у функцію calculateHealthScore() вбудованого програмного забезпечення мікроконтролера ESP32. Вона дозволяє кіберфізичному пристрою самостійно, без залучення серверних потужностей, проводити первинний інтелектуальний аналіз багатовимірних масивів екологічних даних, забезпечуючи надійну роботу локального людино-машинного інтерфейсу та систем аварійної сигналізації.

2.5. Програмна реалізація асинхронного клієнтського застосунку (Telegram-бота)

Для забезпечення зручного кросплатформенного доступу до кіберфізичної системи моніторингу AeroPulse та оперативного інформування користувачів про небезпечні зміни стану навколишнього середовища, в якості мобільного людино-машинного інтерфейсу (HMI) було розроблено та програмно реалізовано

спеціалізований Telegram-бот. Вибір месенджера Telegram як клієнтського термінала автоматизації усуває інженерну необхідність проектування, компіляції та підтримки окремих мобільних додатків під різні операційні системи (Android, iOS), мінімізує витрати на розгортання та гарантує миттєву доставку Push-сповіщень через георозподілену хмарну інфраструктуру серверів месенджера.

Програмний модуль клієнтського застосунку реалізовано мовою Python версії 3.11 з використанням сучасного асинхронного фреймворку aiogram (версії 3.4.0+).

Для забезпечення високого рівня інформаційної безпеки та унеможливлення витоку конфіденційних даних, у розробленому програмному забезпеченні імплементовано механізм змінних оточення (Environment Variables). Уся чутлива конфігураційна інформація зберігається в окремому локальному файлі .env, який ізольований від основного вихідного коду та виключений із системи контролю версій. Структура цього файлу містить ключові параметри доступу: унікальний криптографічний токен Telegram-бота (BOT_TOKEN), ідентифікатор чату адміністратора для маршрутизації тривожних сповіщень (ADMIN_CHAT_ID), а також реквізити для інтеграції з платформою візуалізації (внутрішні та зовнішні URL-адреси серверів, ідентифікатор цільового дашборду та авторизаційний ключ GRAFANA_TOKEN). Такий архітектурний підхід дозволяє гнучко налаштовувати та безпечно розгортати систему на нових апаратних платформах без необхідності модифікації чи перекомпіляції логіки програми.

Ключовою архітектурною особливістю розробленого ПЗ є його побудова на базі обробки подій (Event-Driven Architecture) та використання системного подієвого циклу (Event Loop) бібліотеки asyncio. На відміну від класичних синхронних скриптів, які блокують потоки обчислень під час очікування відповідей від мережевих сокетів, розроблений бот у межах одного процесу паралельно та асинхронно виконує дві незалежні глобальні інженерні задачі:

1. Фоновий MQTT-слухач (Background Listener): Ініційований за допомогою асинхронного клієнта aiomqtt, він безперервно підтримує стійке TCP-

з'єднання з брокером Eclipse Mosquitto всередині ізольованої Docker-мережі. Серверний потік автоматично підписується на топік diploma/sensors/air, перехоплює транзитні JSON-пакети від мікроконтролера ESP32, десеріалізує їх та у реальному часі оновлює значення глобального оперативнго словника станів shared_sensor_data.

2. Асинхронний диспетчер інтерфейсу (Dispatcher): Обробляє чергу вхідних запитів від користувачів (текстові команди, активація кнопок Inline-клавіатур), миттєво генеруючи відповіді та не перериваючи при цьому критичний процес постійного збору екологічної телеметрії з брокера.

Взаємодія користувача з графічним інтерфейсом бота побудована на базі дворівневої системи меню. Головна панель керування реалізована за допомогою класу ReplyKeyboardMarkup і закріплюється у нижній частині екрана застосунку. Вона містить навігаційні кнопки «Поточний стан», «Графіки та Аналітика» та динамічний перемикач режимів роботи «Розумні сповіщення», текстовий підпис якого автоматично змінюється залежно від стану булевої змінної smart_mode в оперативній пам'яті (УВІМК або ВИМК).

При натисканні кнопки «Поточний стан» програма витягує з пам'яті актуальні фізичні величини (температуру, вологість, масову концентрацію частинок PM2.5, тиск та опір VOC) і передає їх у логічний модуль експертної оцінки get_expert_analysis(). Даний модуль зіставляє отримані дані із санітарно-гігієнічними нормами та генерує розгорнутий текстовий висновок із безпечним HTML-форматуванням, сповіщаючи користувача про поточний рівень індексу повітря (Відмінно, Задовільно або Небезпечно) та надаючи чіткі інженерні рекомендації (наприклад, зачинити вікна чи увімкнути кліматичне обладнання).

Для реалізації нестандартних користувацьких запитів (наприклад, побудова графіків за довільний період) у коді бота імплементовано патерн машини скінченних станів (FSM). При виборі опції ручного налаштування часу бот блокує

обробку стандартних команд і переходить у стан очікування (GraphStates.waiting_for_period).

Отриманий від користувача текст валідується спеціальною функцією, яка конвертує його у стандартизовані інтервали Grafana (наприклад, 45m, 6h). Після успішної валідації бот формує асинхронний HTTP-запит до мікросервісу Grafana Image Renderer. Важливим архітектурним рішенням стало усунення розсинхронізації часових поясів (Timezone Mismatch) між Docker-контейнерами (UTC+0) та локальним часом користувача шляхом додавання параметра &tz=Europe/Kyiv до URL-запиту. Рендерер формує статичне PNG-зображення графіка і повертає його боту, який миттєво відправляє файл у чат разом із посиланням на інтерактивний Live-дашборд, після чого стан FSM автоматично скидається. Повний лістинг коду обробки запитів наведено у Додатку Г.

2.6. Алгоритми предиктивного аналізу та числового гістерезису сповіщень

Найбільш наукомісткою частиною програмного коду Telegram-бота є блок аналізу даних та прийняття рішень (check_and_send_alerts). Більшість існуючих систем розумного дому використовують статичні пороги (наприклад, надсилають тривогу строго при перевищенні концентрації пилу понад 35 мкг/м³). Це призводить до двох критичних проблем: пізнього виявлення небезпеки та надмірного інформаційного спаму під час коливань показника на межі норми (осциляції).

Для усунення цих недоліків у логіку бота інтегровано два взаємодоповнюючі математичні алгоритми:

1. Алгоритм предиктивного аналізу трендів (Smart Mode). Система не чекає досягнення критичної позначки, а аналізує швидкість погіршення ситуації. Для цього в оперативній пам'яті бота організовано ковзне часове вікно (sliding window) тривалістю 300 секунд (5 хвилин) для кожної метрики. Бот постійно обчислює різницю між поточним показником та найстарішим значенням у буфері. Якщо, наприклад, концентрація пилу зросла на Δ PM > 10 мкг/м³ за 5

хвилин, бот миттєво генерує попередження про аномальний стрибок (ймовірне задимлення), дозволяючи користувачеві відреагувати до того, як рівень стане критичним.

2. Алгоритм числового гістерезису з відстеженням контексту. Для боротьби з осциляцією сигналів на межі зон (наприклад, коли пил коливається в межах 14.9 – 15.2 мкг/м³) імплементовано систему двох порогів — порогу активації тривоги та порогу деактивації (стабілізації). Наприклад, статус «Небезпека» для пилу вмикається при > 35 мкг/м³. Поки пил не опуститься нижче безпечного порогу стабілізації (≤12 мкг/м³), повторні тривоги блокуються глобальним таймером (cooldown).

З метою покращення користувацького досвіду (UX), бот фіксує ідентифікатор повідомлення (message_id) кожної надісланої тривоги в словнику active_alerts. Коли алгоритм гістерезису фіксує стабілізацію показників (наприклад, приміщення успішно провітрено), бот використовує метод reply_to_message_id. Завдяки цьому повідомлення «Стабілізація: Рівень пилу повернувся в норму» надходить як пряма відповідь (Reply) на конкретне попереднє повідомлення про тривогу, що дозволяє користувачеві легко відстежувати причинно-наслідкові зв'язки в історії чату.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано повний цикл проєктування, структурної інтеграції та програмно-апаратної реалізації комплексу AeroPulse. Розроблено трирівневу архітектуру системи (Edge, Platform, Application), що дозволило чітко розділити процеси збору, збереження та візуалізації телеметрії. Створено апаратний вимірювальний вузол на базі мікроконтролера ESP32 із застосуванням неблокуючої кооперативної багатозадачності. Розроблено та імплементовано математичну модель розрахунку зваженого 100-бального індексу якості повітря (Health Score), який керує локальним TFT-дисплеєм та трирівневою світлодіодною сигналізацією.

Серверну інфраструктуру успішно контейнеризовано за допомогою Docker Compose, об'єднавши брокер Mosquitto, сервіс Telegraf та базу даних часових рядів InfluxDB 2.7 із токен-авторизацією. Завдяки розробці асинхронного Telegram-бота на базі машини скінченних станів (FSM) реалізовано надійний дистанційний HMI-інтерфейс. Впровадження інноваційних алгоритмів предиктивного аналізу 5-хвилинних трендів та двопорогового числового гістерезису з контекстними відповідями (Reply) дозволило повністю вирішити проблему хибних спрацьовувань та перетворити вимірювальний комплекс на інтелектуальну експертну систему моніторингу.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Методика та умови проведення експериментальних досліджень

Головною метою проведення етапу експериментальних досліджень є практична верифікація працездатності розробленого програмно-апаратного комплексу AeroPulse у реальних умовах експлуатації. Це передбачає перевірку коректності зчитування та оцифрування телеметрії сенсорними вузлами, оцінку швидкодії локальної та хмарної обробки даних, а також тестування стійкості алгоритмів Telegram-бота та серверної мікросервісної інфраструктури до штучно створених стресових навантажень.

Для комплексного та об'єктивного підтвердження заявлених технічних характеристик системи було розроблено багатокрокову програму натурних випробувань. Вона охоплює всі рівні архітектури проєкту і складається з трьох основних етапів:

Апаратне тестування сенсорної периферії (Edge-рівень): Штучна різка зміна екологічних показників навколишнього середовища. Цей етап включає імітацію задимлення для перевірки динаміки лазерного лічильника PMS5003, а також підвищення концентрації метаболічних газів і вологи для оцінки швидкості хімічної реакції металооксидного сенсора BME680.

Програмне тестування логіки та інтерфейсів (Application-рівень): Перевірка коректності роботи математичної моделі розрахунку комплексного індексу здоров'я середовища (Health Score). Оцінка миттєвої реакції локальної світлодіодної індикації та графічного дисплея, а також верифікація спрацьовування тривожних сповіщень Telegram-бота за розробленими алгоритмами предиктивного аналізу і числового гістерезису.

Тестування стабільності та відмовостійкості системи (Platform-рівень): Перевірка довготривалої роботи комплексу. Оцінка здатності апаратного вузла до

безперебійної роботи, перевірка цілісності запису даних у СУБД InfluxDB та аналіз безвідмовності функціонування Docker-контейнерів під час багатогодинного безперервного моніторингу.

Для забезпечення чистоти експериментів натурні випробування проводилися у типовому закритому житловому приміщенні в умовах відсутності сильних протягів та прямого впливу кліматичної техніки. Апаратний вимірювальний вузол розміщувався на висоті близько 1,5 метра від рівня підлоги, що відповідає стандартній зоні дихання людини. Перед початком активної фази випробувань пристрій обов'язково проходив етап апаратного «прогріву» для виведення нагрівальної пластини сенсора VOC у робочий температурний режим.

Зовнішній вигляд повністю зібраного та змонтованого в корпусі вимірювального вузла системи AeroPulse, підготовленого до початку проведення натурних експериментів, наведено на рисунку 3.1. На дисплеї пристрою відображаються штатні стартові показники якості повітря, а активний зелений індикатор підтверджує відмінний екологічний фон у приміщенні до початку проведення штучних стрес-тестів.



Рисунок 3.1 – Зовнішній вигляд готового програмно-апаратного комплексу AeroPulse в умовах експлуатації.

Початкові кліматичні умови перед початком кожного стрес-тесту відповідали нормальним санітарно-гігієнічним нормам: температурний режим та відносна вологість перебували у комфортній зоні, концентрація дрібнодисперсного пилу PM2.5 була мінімальною, а опір напівпровідникового газоаналізатора фіксував відмінну якість повітряного середовища. Фіксація цих початкових значень дозволила сформувати стабільну базову лінію (Baseline) для подальшого точного відстеження градієнтів відхилень під час проведення штучних екологічних аномалій.

3.2. Дослідження реакції системи на зміну концентрації PM2.5 та VOC

Перший етап натурних досліджень був спрямований на перевірку швидкості реакції лазерного лічильника частинок PMS5003 та точності відпрацювання алгоритму числового гістерезису. Моделювання задимлення здійснювалося шляхом короткочасного спалювання побутового сірника біля повітрозабірних отворів пристрою.

До початку експерименту базовий рівень концентрації дрібнодисперсного пилу становив ідеальні 0 мкг/м³. Після появи диму лазерний датчик миттєво зафіксував різке посилення інтенсивності розсіяного світла, і показник PM2.5 стрибкоподібно зріс до пікового значення 298 мкг/м³. Мікроконтролер штатно обробив ці дані, обчислив критичне падіння індексу Health Score і передав телеметрію на MQTT-брокер. Telegram-бот миттєво згенерував повідомлення найвищого рівня загрози: «КРИТИЧНО: Пил PM2.5 > 35», зберігши ідентифікатор цього повідомлення в оперативній пам'яті.

Після провітрювання приміщення система блискуче продемонструвала роботу алгоритму гістерезису. Хоча поріг тривоги становить 35 мкг/м³, під час падіння пилу до позначки 15 мкг/м³ бот продовжував утримувати стан тривоги, блокуючи осциляцію (миготіння) статусів. Лише після того, як концентрація перетнула нижній поріг деактивації ($V_{deact} \leq 12$ мкг/м³), зафіксувавши значення 6 мкг/м³, бот надіслав повідомлення про стабілізацію у вигляді контекстної відповіді (Reply). Комбінований графік падіння індексу здоров'я та повідомлень Telegram-бота наведено на рисунку 3.1.



Рисунок 3.2 – Комплексна реакція системи на задимлення (стрибок PM2.5 та відпрацювання гістерезису).

Другий етап досліджень був розроблений для верифікації чутливості сенсора BME680 до хімічного забруднення (VOC) та перевірки алгоритму предиктивного аналізу трендів («Розумного режиму» бота). Експеримент полягав у спрямованому видиху людини на датчик протягом 20 секунд.

Базовий опір напівпровідникового шару до початку тесту становив стабільні 217 кОм (чисте повітря). Під час тесту газова суміш із видиху (яка містить вуглекислий газ та інші метаболіти) спровокувала миттєве падіння електричного опору до мінімальних 75 кОм.

Паралельно було зафіксовано цікавий фізичний ефект (рис. 3.2): разом із газами відбулося зростання концентрації пилу PM2.5 (з 0 до 18 мкг/м³), стрімке підвищення відносної вологості (з 55% до понад 80%) та незначне зростання температури на 1 °C. Це пояснюється тим, що лазерний датчик пилу фіксує мікрокраплі вологи з теплого видиху як дисперсні частинки.

Алгоритм предиктивного аналізу в Telegram-боті успішно обробив ці швидкі зміни в рамках 5-хвилинного ковзного вікна і згенерував попередження про аномалію ще до того, як опір газу впав до абсолютно критичних значень.



Рисунок 3.3 – Фіксація аномалії хімічного складу повітря (VOC) та реакція предиктивного алгоритму.

3.3. Тестування людино-машинного інтерфейсу та алгоритмів скінченних станів (FSM)

Окремим важливим етапом досліджень стала верифікація зручності (UX) та коректності роботи (UI) клієнтського застосунку. Під час штатної експлуатації було протестовано роботу модуля генерації експертних висновків («Поточний стан») та логіку машини скінченних станів (FSM) при запиті графіків за нестандартний період часу.

Користувач ініціював запит на побудову графіка, обравши опцію ручного налаштування періоду. Диспетчер бота успішно перевів поточну сесію в режим очікування тексту, призупинивши обробку стандартних команд. Після введення користувачем текстового значення, програмний валідатор коректно розпізнав та розпарсив нестандартний часовий проміжок. Відбувся успішний API-запит до

внутрішнього мікросервісу Grafana Image Renderer, який повернув готове статичне PNG-зображення сформованого графіка безпосередньо у чат бота.

Окрім цього, перевірено роботу багаторівневої Reply-клавіатури, яка забезпечує швидку навігацію між режимами без необхідності ручного введення команд. Загальний процес тестування інтерфейсу користувача та відпрацювання алгоритмів FSM наведено на рисунку 3.3.



Рисунок 3.4 – Демонстрація роботи багаторівневого меню та логіки машини скінченних станів (FSM) Telegram-бота.

3.4. Дослідження довготривалої стабільності серверної інфраструктури та вимірювального вузла

Для кіберфізичних систем та пристроїв Інтернету речей критично важливим показником є здатність працювати автономно протягом тривалого часу (Reliability) без програмних «зависань», витоків оперативної пам'яті (Memory Leaks) або розривів мережевого з'єднання. Для верифікації цих параметрів було проведено 8-годинний тест безперервного пасивного моніторингу (Stability Test) у нічний та ранковий час (з 02:00 до 10:00).

Протягом усього періоду тестування мікроконтролер ESP32 успішно виконував цикл опитування датчиків та передачу телеметрії через протокол MQTT до серверного вузла. Серверна інфраструктура, розгорнута в ізольованих Docker-контейнерах (брокер Mosquitto, база даних InfluxDB та платформа візуалізації Grafana), продемонструвала абсолютну стабільність: зафіксовано повну відсутність перезавантажень сервісів або критичних помилок парсингу даних.

Загальний вигляд аналітичного дашборду за результатами 8-годинного моніторингу наведено на рисунку 3.4. Аналіз часових рядів підтверджує наступне:

1. Безперервність потоку даних: На графіках температури, вологості та показників якості повітря відсутні «білі плями» (провали даних), що свідчить про стабільну роботу Wi-Fi модуля та відсутність втрат пакетів (Packet Loss) у мережі.
2. Адекватність динаміки: Система коректно відображає природну динаміку зміни температури та вологості у приміщенні протягом нічного періоду, що підтверджує чутливість та калібрування сенсорів у довгостроковій перспективі.
3. Стабільність серверного стеку: Відсутність викидів («шумів») або аномальних стрибків значень при незмінних фізичних умовах підтверджує

надійність методів обробки даних у InfluxDB та коректність роботи API-запитів до Grafana.



Рисунок 3.5 – Візуалізація результатів 8-годинного тесту на стабільність (Stability Test) у платформі Grafana.

3.5. Перспективи комерційного масштабування проєкту

Розроблена система AeroPulse є повнофункціональним мінімально життєздатним продуктом (MVP). Однак, архітектурні рішення, закладені на етапі проектування (зокрема, використання контейнеризації та бази даних InfluxDB v2.7), дозволяють трансформувати лабораторний прототип у повноцінну хмарну SaaS-платформу (Software as a Service) комерційного рівня.

Для подальшого масштабування системи до рівня багатоквартирного будинку, офісного центру або промислового підприємства передбачається імплементація наступних інженерних кроків:

1. Мультитенантність (Multi-tenancy) та ізоляція даних. Завдяки структурі InfluxDB 2.7, для кожного нового клієнта (або будівлі) можна створювати

окремі сутності Organization та Bucket, що гарантує сувору криптографічну ізоляцію даних користувачів.

2. Апаратна ідентифікація. Інтеграція алгоритмів зчитування унікальної MAC-адреси мережевого інтерфейсу кожного мікроконтролера ESP32. Цей ідентифікатор буде додаватися до JSON-пакету, що дозволить брокеру маршрутизувати дані від сотень пристроїв у відповідні комірки бази даних без плутанини.
3. Динамічна шаблонізація Grafana. Замість створення окремого дашборду для кожного вузла, буде застосовано механізм змінних (Variables) у Grafana, який дозволить користувачеві вибирати потрібний датчик (наприклад, «Кімната 1», «Конференц-зал») з випадаючого списку, динамічно перемальовуючи єдиний шаблонний інтерфейс.
4. Хмарна міграція. Перенесення локальних Docker-контейнерів на георозподілені потужності хмарних провайдерів (наприклад, AWS EC2 або Google Cloud Platform) із застосуванням систем оркестрації рівня Kubernetes для забезпечення балансування навантаження при одночасному підключенні тисяч пристроїв.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи проведено комплексне експериментальне тестування розробленої системи AeroPulse. Натурні випробування із симуляцією задимлення (стрибкоподібне підвищення PM2.5 до 298 мкг/м^3) та погіршення якості повітря (падіння опору VOC до 75 кОм) підтвердили високу чутливість сенсорів PMS5003 та BME680. Оцінка стану середовища миттєво і штатно відображалася на локальному HMI.

Тестування мобільного клієнтського застосунку підтвердило дієвість впроваджених алгоритмів предиктивного аналізу та числового гістерезису. Система успішно блокувала хибні повідомлення під час падіння рівня пилу (дочекавшись стабілізації на рівні 6 мкг/м^3) і надсилала підтвердження у вигляді Reply-

повідомлень. Успішно підтверджено роботу алгоритмів машини скінченних станів (FSM) при ручному введенні часових параметрів.

Проведення 8-годинного тесту на стабільність довело високу надійність програмно-апаратного стеку: система працювала безперебійно, не зафіксувавши втрат пакетів телеметрії або збоїв у роботі Docker-контейнерів. Теоретично обґрунтовано шляхи комерційного масштабування прототипу до рівня повноцінної хмарної SaaS-платформи з використанням механізмів мультитенантності, що підтверджує високий інженерний рівень розробленого комплексу.

ВИСНОВКИ

Мета кваліфікаційної роботи, яка полягала у вдосконаленні методів і моделей побудови комп'ютерно-інтегрованої системи моніторингу мікроклімату та якості повітря на основі сенсорів BME680 і PMS5003 шляхом застосування MQTT-архітектури «видавець–підписник», методів обробки часових рядів та інтегрального оцінювання стану середовища, була успішно досягнута.

Відповідно до поставленої мети було вирішено актуальну науково-практичну задачу проєктування та розробки програмно-апаратного комплексу для моніторингу екологічного стану приміщень AeroPulse, побудованого на принципах Інтернету речей (IoT) та мікросервісної архітектури.

На основі аналізу існуючих рішень обґрунтовано вибір мікроконтролера ESP32 як обчислювального ядра, що забезпечує необхідну продуктивність для асинхронної обробки даних та підтримки захищеного Wi-Fi з'єднання. Розроблено та імплементовано трирівневу архітектуру системи (Edge, Platform, Application), що дозволило забезпечити автономність вимірювального вузла, стабільність серверної інфраструктури та гнучкість клієнтського інтерфейсу.

В ході роботи отримано наступні результати:

1. Апаратна реалізація: розроблено вимірювальний вузол, який інтегрує високоточні сенсори якості повітря (BME680, PMS5003), HMI-інтерфейс на базі TFT-дисплея та світлодіодну систему сигналізації. Реалізовано кооперативну багатозадачність, яка дозволяє пристрою працювати стабільно 24/7 навіть при тимчасових мережевих збоях.
2. Математичне забезпечення: розроблено та впроваджено модель інтегральної оцінки якості повітря (Health Score), що базується на адаптивному зважуванні субіндексів пилу та газів. Це дозволило трансформувати сирі дані сенсорів у зрозумілий для користувача 100-бальний індекс здоров'я середовища.

3. Серверна інфраструктура: створено масштабовану платформу на базі Docker, що об'єднує брокер Mosquitto, сервіс Telegraf та базу даних часових рядів InfluxDB. Такий підхід гарантує цілісність даних та високу швидкість обробки телеметрії.
4. Програмний інтерфейс: розроблено асинхронного Telegram-бота з використанням машини скінченних станів (FSM). Впроваджені алгоритми предиктивного аналізу трендів та числового гістерезису дозволили мінімізувати кількість хибних тривог, забезпечуючи користувача лише релевантною інформацією.
5. Експериментальна верифікація: натурні випробування підтвердили коректність функціонування системи у критичних режимах (імітація задимлення, хімічне забруднення). 8-годинний тест на стабільність підтвердив відсутність втрат пакетів даних та збоїв у роботі мікросервісів, що доводить інженерну надійність розробленого рішення.

Розроблена система AeroPulse володіє високим потенціалом для подальшого комерційного масштабування до рівня хмарної SaaS-платформи завдяки використанню технологій мультитенантності та контейнеризації, що підтверджує наукову новизну та практичну цінність виконаної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Joshi S. M. The sick building syndrome // Indian Journal of Occupational and Environmental Medicine. 2008. Vol. 12, No. 2. P. 61–64.
2. WHO guidelines for indoor air quality: selected pollutants. Geneva : World Health Organization, 2010. 454 p.
3. The impact of PM2.5 on the human respiratory system / Y. F. Xing, Y. H. Xu, M. H. Shi, Y. X. Lian // Journal of Thoracic Disease. 2016. Vol. 8, No. 1. P. 69–74.
4. Salthammer T., Mentese S., Marutzky R. Formaldehyde in the indoor environment // Chemical Reviews. 2010. Vol. 110, No. 4. P. 2536–2572.
5. Державні санітарні норми мікроклімату виробничих приміщень: ДСН 3.3.6.042-99. Київ : МОЗ України, 1999. 16 с.
6. Comparison of a human hair vs. PM10 vs. PM2.5 [Електронний ресурс] // ResearchGate. URL: https://www.researchgate.net/figure/Comparison-of-a-human-hair-vs-PM10-vs-PM25_fig1_346549289 (дата звернення: 18.01.2026).
7. ESP32 Series Datasheet [Електронний ресурс] // Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата звернення: 10.02.2026).
8. BME680 Low power gas, pressure, temperature & humidity sensor [Електронний ресурс] // Bosch Sensortec. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf (дата звернення: 12.02.2026).
9. Internet of Things: Survey and open issues of MQTT protocol / M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, R. Al-Hatmi // 2017 International Conference on Engineering & MIS (ICEMIS). Monastir, 2017. P. 1–6.
10. MQTT publish/subscribe architecture [Електронний ресурс] // ResearchGate. URL: https://www.researchgate.net/figure/MQTT-publish-subscribe-architecture_fig1_374143113 (дата звернення: 28.02.2026).

11. MQTT Version 5.0. OASIS Standard [Електронний ресурс]. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (дата звернення: 02.03.2026).
12. Digital universal particle concentration sensor PMS5003 manual [Електронний ресурс] // Plantower. URL: https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf (дата звернення: 15.02.2026).
13. Docker Documentation. Containerize an application [Електронний ресурс]. URL: https://docs.docker.com/ (дата звернення: 15.03.2026).
14. InfluxDB 2.7 Documentation. Time Series Data Platform [Електронний ресурс]. URL: https://docs.influxdata.com/influxdb/v2/ (дата звернення: 20.03.2026).
15. Grafana Documentation. Dashboards and Visualizations [Електронний ресурс]. URL: https://grafana.com/docs/grafana/latest/ (дата звернення: 05.04.2026).
16. Aiogram 3.x Documentation. Asynchronous framework for Telegram Bot API [Електронний ресурс]. URL: https://docs.aiogram.dev/en/latest/ (дата звернення: 10.04.2026).
17. Banzi M., Shiloh M. Getting Started with Arduino. Maker Media, Inc, 2014. 262 p.

ДОДАТКИ

Додаток А

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

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

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача <u>першого (бакалаврського) рівня вищої освіти</u>
(першого (бакалаврського) / другого (магістерського))
<u>ЛІСАКА ІВАНА БОРИСОВИЧА</u>
(прізвище, ім'я, по батькові здобувача)

Спеціальність: 151 – Автоматизація, комп'ютерно-інтегровані технології
Освітня програма: «Автоматизація, комп'ютерно-інтегровані технології»
1. Тема роботи **<u>СИСТЕМА ЗБОРУ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ З БЕЗДРОТОВИХ СЕНСОРІВ ЯКОСТІ ПОВІТРЯ З ВИКОРИСТАННЯМ MQTT-ПРОТОКОЛУ</u>**
керівник роботи: **<u>Бережний Артем Андрійович, старший викладач</u>**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи: ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) мікроконтролерна платформа ESP32;
   2) комплект сенсорів мікроклімату та дрібнодисперсного пилу;
   3) середовища розробки ПЗ та інструменти контейнеризації додатків;
   4) бездротова мережа Wi-Fi та IoT-протокол MQTT;
   5) серверне ПЗ (брокер повідомлень, база даних часових рядів, платформа візуалізації);
   6) ПК для розробки та відлагодження системи.
4. Перелік питань, які потрібно розробити:
   1) Аналіз сучасних підходів до побудови автоматизованих систем моніторингу параметрів навколишнього середовища.
   2) Дослідження та порівняльний аналіз бездротових протоколів передачі даних в мережах Інтернету речей (IoT).
   3) Проєктування структурно-функціональної моделі системи збору та візуалізації даних на основі MQTT-технології.
   4) Програмно-апаратна реалізація бездротового вимірювального вузла та алгоритмів первинної обробки інформації.
   5) Розробка підсистеми дистанційної візуалізації, накопичення даних та автоматизованого сповіщення користувачів.
   6) Експериментальне дослідження роботи системи та аналіз достовірності отриманих результатів вимірювання.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Визначення актуальності теми, об'єкта, предмета та мети дослідження.	13.10.2025 – 05.11.2025
2	Огляд літературних джерел та існуючих технічних рішень у сфері автоматизованого екологічного моніторингу.	06.11.2025 – 25.11.2025
3	Порівняльний аналіз мережевих протоколів та технічне обґрунтування вибору MQTT-протоколу.	26.11.2025 – 15.12.2025
4	Обґрунтування вибору мікропроцесорної платформи та інтелектуальних сенсорів якості повітря.	16.12.2025 – 30.12.2025
5	Розробка структурної та функціональної моделей системи з урахуванням бездротової архітектури.	06.01.2026 – 21.01.2026
6	Проєктування алгоритмів роботи вбудованого програмного забезпечення для вимірювального вузла.	22.01.2026 – 05.02.2026
7	Розробка програмних модулів для зчитування, фільтрації та цифрової обробки сигналів із сенсорів.	06.02.2026 – 25.02.2026
8	Реалізація механізмів бездротового обміну даними та конфігурування MQTT-клієнтів.	26.02.2026 – 12.03.2026
9	Розробка бази даних для накопичення результатів вимірювань та системи управління сховищем.	13.03.2026 – 30.03.2026
10	Створення людино-машинного інтерфейсу (дашборду) та підсистеми активного сповіщення користувачів.	01.04.2026 – 12.04.2026
11	Складання апаратної частини лабораторного прототипу та налагодження взаємодії між рівнями системи.	15.04.2026 – 30.04.2026
12	Проведення серії експериментів для перевірки точності та стабільності роботи системи.	01.05.2026 – 06.05.2026
13	Систематизація отриманих результатів та оформлення пояснювальної записки.	06.05.2026 – 14.05.2026
14	Підготовка демонстраційних матеріалів та проходження процедури передзахисту.	14.05.2026 – 20.05.2026
15	Остаточне редагування тексту роботи та перевірка на відповідність вимогам нормоконтролю.	до 25.05.2026
16	Подання кваліфікаційної роботи на рецензування та підготовка до захисту.	до 30.05.2026

6. Дата видачі завдання: ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Іван ЛІСАК
(підпис)

Керівник роботи _____ Артем БЕРЕЖНИЙ
(підпис)

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу «Система збору та візуалізації даних з бездротових сенсорів якості повітря з використанням MQTT-протоколу»

1.	Введення	1.1. Назва: Система збору та візуалізації даних з бездротових сенсорів якості повітря з використанням MQTT-протоколу 1.2. Галузь застосування: комп'ютерні технології.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/225 від «30» травня 2026 р. (Додаток А до пояснюва-льної записки).
3.	Призначення розробки	3.1. Мета розробки: вдосконалення методів і моделей побудови комп'ютерно-інтегрованої системи моніторингу мікроклімату та якості повітря на основі сенсорів BME680 і PMS5003 шляхом застосування MQTT-архітектури «видавець–підписник», методів обробки часових рядів та інтегрального оцінювання стану середовища. 3.2. Призначення розробки: безперервний збір, збереження, візуалізація та предиктивний аналіз параметрів мікроклімату та якості повітря (дрібнодисперсний пил PM2.5, гази VOC, температура, вологість) з інформуванням користувачів через локальний та мобільний інтерфейси. 3.3. Вихідні дані розробки: санітарно-гігієнічні норми мікроклімату (ДСН 3.3.6.042-99), технічні документації на сенсори BME680 та PMS5003, специфікація протоколу MQTT.
4.	Технічні вимоги до	4.1. Вимоги до функціональних характеристик: збір даних з датчиків, розрахунок інтегрального індексу Health Score, передача даних по MQTT, збереження в

<table>
<tr><td></td><td>програмного виробу</td><td>базі даних часових рядів, візуалізація графіків, генерація тривог через Telegram-бот.
4.2. Вимоги до надійності: забезпечувати безперебійну роботу мікроконтролера 24/7 (за рахунок неблокуючої багатозадачності), автоматичне відновлення зв'язку при збоях Wi-Fi та запобігання втратам даних.
4.3. Вимоги до умов експлуатації: житлові та офісні приміщення, температурний режим експлуатації вимірювального вузла від 0 до +40 °C.
4.4. Вимоги до технічних засобів: мікроконтролер ESP32, цифрові датчики BME680 та PMS5003, TFT-дисплей ILI9341, сервер (ПК або мікрокомп'ютер) з підтримкою Docker.
4.5. Вимоги до програмної сумісності: підтримка стандарту MQTT (Mosquitto), взаємодія з InfluxDB 2.7, Grafana та Telegram API.
4.6. Вимоги до маркування та упаковки: немає.
4.7. Вимоги до транспортування і зберігання: немає.
4.8. Спеціальні вимоги: використання виключно відкритого програмного забезпечення (Open-Source).</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td>Програмною документацією до виробу «Система збору та візуалізації даних з бездротових сенсорів якості повітря з використанням MQTT-протоколу» вважати:
1) Справжнє Технічне завдання на розробку виробу (Додаток Б).
2) Опис апаратної та серверної частини виробу (Розділ 2 пояснювальної записки).
3) Програма і методика випробувань (Розділ 3 пояснювальної записки).
4) Лістинг вихідного коду (Додаток Г).</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td>1) Забезпечення низької вартості апаратного вимірювального вузла порівняно з комерційними пропрієтарними аналогами.
2) Відсутність абонентської плати за використання хмарних сервісів завдяки розгортанню власної локальної серверної інфраструктури (Docker, Mosquitto, InfluxDB).</td></tr>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td><table><tr><td>Дата</td><td>Назва етапу</td></tr><tr><td>20.10.2025 –</td><td>Визначення актуальності теми,</td></tr></table></td></tr>
</table>

		05.11.2025	об’єкта, предмета та мети дослідження.
		05.11.2025 – 25.11.2025	Огляд літературних джерел та існуючих технічних рішень у сфері автоматизованого екологічного моніторингу.
		25.11.2025 – 15.12.2025	Порівняльний аналіз мережевих протоколів та технічне обґрунтування вибору MQTT-протоколу.
		15.12.2025 – 30.12.2025	Обґрунтування вибору мікропроцесорної платформи та інтелектуальних сенсорів якості повітря.
		30.12.2025 – 21.01.2026	Розробка структурної та функціональної моделей системи з урахуванням бездротової архітектури.
		21.01.2026 – 05.02.2026	Проєктування алгоритмів роботи вбудованого програмного забезпечення для вимірювального вузла.
		05.02.2026 – 25.02.2026	Розробка програмних модулів для зчитування, фільтрації та цифрової обробки сигналів із сенсорів.
		25.02.2026 – 12.03.2026	Реалізація механізмів бездротового обміну даними та конфігурування MQTT-клієнтів.
		12.03.2026 – 30.03.2026	Розробка базы даних для накопичення результатів вимірювань та системи управління сховищем.
		30.03.2026 – 12.04.2026	Створення людино-машинного інтерфейсу (дашборду) та підсистеми активного сповіщення користувачів.
		12.04.2026 – 30.04.2026	Складання апаратної частини лабораторного прототипу та налагодження взаємодії між рівнями системи.
		30.04.2026 –	Проведення серії експериментів для

		06.05.2026	перевірки точності та стабільності роботи системи.
		06.05.2026 – 14.05.2026	Систематизація отриманих результатів та оформлення пояснювальної записки.
		14.05.2026 – 20.05.2026	Підготовка демонстраційних матеріалів та проходження процедури передзахисту.
		до 25.05.2026	Остаточне редагування тексту роботи та перевірка на відповідність вимогам нормоконтролю.
		до 30.05.2026	Подання кваліфікаційної роботи на рецензування та підготовка до захисту.
8.	Порядок контролю і приймання програмного продукту	1. Контроль виконання згідно календарного плану; 2. Захист на засіданні Екзаменаційної комісії; 3. Подання пояснювальної записки у паперовому та електронному вигляді.	

Виконавець
студент групи КУ- 41
ЛІСАК І. Б.
_____

Замовник
д. т. н., проф.
БЕРЕЖНИЙ А. А.
_____

Додаток В

Програма і методика випробувань програмного виробу

«Система збору та візуалізації даних з бездротових сенсорів якості повітря з використанням MQTT-протоколу»

1 Об'єкт випробувань

1. Назва програмного виробу: «Система збору та візуалізації даних з бездротових сенсорів якості повітря з використанням MQTT-протоколу»
2. Галузь застосування: Автоматизація та приладобудування

2 Мета випробувань

Перевірка відповідності функціональності програмної та апаратної реалізації розподіленої системи заявленим технічним і технологічним можливостям, що зафіксовані в Технічному завданні (Додаток Б).

3 Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення приймальних випробувань є виконання календарного плану кваліфікаційної роботи бакалавра та наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. Обсяг випробувань

Випробування здійснюються в обсязі, визначеному цією програмою, і охоплюють як Edge-рівень (залізо, локальний HMI), так і Cloud/Platform-рівень (Docker-інфраструктура) та Application-рівень (інтерфейс користувача).

4. Організації, які беруть участь у випробуваннях

Приймальні випробування Приймальні випробування проводяться екзаменаційною комісією за участю керівника роботи, замовника та виконавця (студента групи КУ-41 Лісака І. Б.).

4. Вимоги до програми або програмного виробу

Розроблене програмно-апаратне забезпечення повинно задовольняти таким вимогам:

1. стабільно функціонувати в режимі безперервного моніторингу середовища 24/7;

2. забезпечувати неблокуюче паралельне виконання задач збору даних та оновлення локального екрана (кооперативна багатозадачність);
3. підтримувати кросплатформеність мобільного людино-машинного інтерфейсу (через хмару Telegram);
4. мати захист від некоректних дій користувача (валідація вхідних повідомлень у боті);
5. автоматично відновлювати TCP-з'єднання з брокером повідомлень після мережевих аварій без втрати працездатності.

5. Вимоги до програмної документації

Програмною документацією до розроблюваного комплексу вважати:

1. Пояснювальну записку кваліфікаційної роботи (включаючи архітектурні описи Розділу 2 та аналітичні графіки Розділу 3).
2. Справжнє технічне завдання (Додаток Б).
3. Програму і методику випробувань (Додаток В).
4. Повні лістинги конфігураційних файлів Docker Compose та вихідного коду прошивки мікроконтролера (Додаток Г).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань використовується:

1. Лабораторний макет пристрою на базі мікроконтролера ESP32 із підключеними сенсорами BME680, PMS5003 та TFT-екраном ILI9341.
2. Персональний комп'ютер/сервер із розгорнутим середовищем Docker Compose (сервіси aeropulse-mosquitto, aeropulse-telegraf, aeropulse-influxdb, aeropulse-grafana).
3. Смартфон із встановленим клієнтом месенджера Telegram для взаємодії з ботом @main_aeropulse_bot.
4. Побутові засоби штучного впливу на екологічний фон (сірник — для моделювання задимлення, спрямований видих — для моделювання хімічних аномалій газів VOC та вологості).

6.2 Порядок проведення випробувань

Випробування проводяться у два етапи: перевірка комплектності документації та інфраструктури (Етап 1) і функціональне стрес-тестування алгоритмів комплексу (Етап 2). На Етапі 2 виконуються чотири послідовні тести.

Як правило, випробування проводяться в два етапи:

Тест 1. Перевірка реакції на аерозольне забруднення (PM2.5) та алгоритму гістерезису

- Суть перевірки: Штучне задимлення датчика PMS5003 сірником.
- Очікуваний результат: Стрибок показника PM2.5, падіння Health Score на екрані та в базі даних, миттєве сповіщення в боті. При провітрюванні — утримання тривоги до повного падіння показника ≤ 12 мкг/м³ та надсилання Reply-відповіді про стабілізацію.
- Результат випробувань: Початкове значення пилу PM2.5 дорівнювало 0 мкг/м³. Під час тесту зафіксовано пікове значення 298 мкг/м³. Бот надіслав повідомлення «КРИТИЧНО: Пил PM2.5 > 35 (298.0)». Після провітрювання при значенні 15 мкг/м³ бот утримував тривогу (блок осциляцій). При досягненні безпечного значення 6 мкг/м³ надіслано контекстну відповідь (Reply) про стабілізацію фону середовища.



Рис. В.1 Комплексна реакція системи на задимлення (стрибок PM2.5 та відпрацювання гістерезису).

Тест 2. Перевірка предиктивного аналізу трендів («Розумного режиму» бота) при хімічних аномаліях (VOC)

- Суть перевірки: Спрямований тривалий видих (20 секунд) на металооксидну мембрану датчика BME680.

- Очікуваний результат: Стрімке падіння опору напівпровідника, зростання відносної вологості. Бот повинен виявити швидку зміну градієнта у 5-хвилинному ковзному вікні та надіслати попередження «АНОМАЛІЯ» до перетину статичного аварійного порогу.
- Результат випробувань: Базовий опір VOC становив 217 кОм. При впливі видиху опір впав до 75 кОм. Предиктивний алгоритм бота зафіксував падіння опору на 175.2 кОм за 5 хв і миттєво згенерував алерти. Зафіксовано супутній фізичний ефект: вологість перевищила 80%, температура зросла на 1 °C, а лазерний лічильник зчитав мікрокраплі як частинки пилу (стрибок PM2.5 до 18 мкг/м³), що штатно відобразилося на комплексному дашборді.



Рис. В.2 Фіксація аномалії хімічного складу повітря (VOC) та реакція предиктивного алгоритму.

Тест 3. Перевірка людино-машинного інтерфейсу та працездатності машини скінченних станів (FSM)

- Суть перевірки: Запит поточного стану середовища та кастомна генерація графіка через Inline-клавіатуру бота за допомогою ручного введення тексту.
- Очікуваний результат: Бот коректно генерує структурований HTML-звіт «Поточний стан». При виборі «Налаштувати свій період» диспетчер блокує стандартні команди (перехід FSM у стан очікування), валідує введений час та надсилає статичний графік від Grafana Image Renderer.
- Результат випробувань: Бот успішно згенерував експертний висновок із зеленим маркером «ІНДЕКС ПОВІТРЯ: ВІДМІННО». При тестуванні FSM на

некоректне введення («12345») бот надіслав попередження про помилку формату. Після введення валідного значення «0.5 год» бот перевів запит в інтервал «30m», викликав рендерер Chromium і надіслав у чат готове PNG-зображення графіка разом із кнопкою-посиланням на живий дашборд.



Рис. В3 Демонстрація роботи багаторівневого меню та логіки машини скінченних станів (FSM) Telegram-бота.

Тест 4. Дослідження довготривалої надійності та стабільності серверних мікросервісів

- Суть перевірки: Безперервне пасивне функціонування системи у нічний та ранковий часові періоди тривалістю 8 годин (з 02:00 до 10:00).
- Очікуваний результат: Відсутність розривів ліній на графіках часових рядів Grafana (0% Packet Loss), стабільне утримання сесії MQTT-клієнтом, відсутність перезапускання Docker-контейнерів.
- Результат випробувань: Система опрацювала 8-годинну сесію пасивного збору даних в автоматичному режимі без збоїв. На аналітичних панелях Grafana зафіксовано суцільні unbroken-лінії трендів без аномальних викидів. База даних InfluxDB 2.7 успішно акумулювала та проіндексувала весь масив хронологічної телеметрії, а Health Score стабільно утримувався на позначці 100%, підтверджуючи високу відмовостійкость платформи.



Рис. В4 Візуалізація результатів 8-годинного тесту на стабільність (Stability Test) у платформі Grafana.

Висновки: Програма та методика приймальних випробувань виконана в повному обсязі. Тест 1, Тест 2, Тест 3 та Тест 4 пройшли успішно. Усі елементи комп'ютерно-інтегрованої системи моніторингу екологічного стану «AeroPulse» функціонують злагоджено, алгоритми обробки подій та фільтрації сповіщень підтвердили свою математичну та інженерну спроможність. Вироб вважається готовим до штатної експлуатації.

Виконавець: студент групи КУ-41, Лісак І. Б. 

Додаток Г

Лістинги вихідного коду та конфігураційних файлів програмно-апаратного комплексу

Г.1 Лістинг конфігураційного файлу серверної інфраструктури docker-compose.yml

```
version: "3.9"

services:
  mosquitto:
    image: eclipse-mosquitto:2
    container_name: aeropulse-mosquitto
    restart: unless-stopped
    ports:
      - "1883:1883" # MQTT-порт для ESP32
    networks:
      - aeropulse-net
    volumes:
      - mosquitto_data:/mosquitto/data
      - mosquitto_config:/mosquitto/config
    command: >
      sh -c "if [ ! -f /mosquitto/config/mosquitto.conf ]; then
      printf 'persistence true\npersistence_location /mosquitto/data/\nlistener
1883\nallow_anonymous true\n' > /mosquitto/config/mosquitto.conf;
      fi;
      exec mosquitto -c /mosquitto/config/mosquitto.conf"

  influxdb:
    image: influxdb:2.7
    container_name: aeropulse-influxdb
    restart: unless-stopped
    ports:
      - "8086:8086" # HTTP API InfluxDB
    networks:
      - aeropulse-net
    volumes:
      - influxdb_data:/var/lib/influxdb2
    environment:
      DOCKER_INFLUXDB_INIT_MODE: setup
      DOCKER_INFLUXDB_INIT_USERNAME: admin
      DOCKER_INFLUXDB_INIT_PASSWORD: admin12345
      DOCKER_INFLUXDB_INIT_ORG: AeroPulse
      DOCKER_INFLUXDB_INIT_BUCKET: air_quality
      DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: aeropulse-super-token

  telegraf:
    image: telegraf:1.30
    container_name: aeropulse-telegraf
    restart: unless-stopped
    depends_on:
      - mosquitto
      - influxdb
    networks:
      - aeropulse-net
```

```
    environment:
      INFLUX_TOKEN: aeropulse-super-token
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro


  grafana:
    image: grafana/grafana:10.4.6
    container_name: aeropulse-grafana
    restart: unless-stopped
    ports:
      - "3000:3000" # Веб-інтерфейс Grafana
    networks:
      - aeropulse-net
    volumes:
      - grafana_data:/var/lib/grafana
    depends_on:
      - influxdb
    environment:
      GF_SECURITY_ADMIN_USER: admin
      GF_SECURITY_ADMIN_PASSWORD: admin12345

      GF_RENDERING_SERVER_URL: http://renderer:8081/render
      GF_RENDERING_CALLBACK_URL: http://grafana:3000/

  renderer:
    image: grafana/grafana-image-renderer:latest
    container_name: aeropulse-renderer
    restart: unless-stopped
    ports:
      - "8081:8081"
    networks:
      - aeropulse-net

  bot:
      build: ./bot
      container_name: aeropulse-bot
      restart: unless-stopped
      depends_on:
        - mosquitto
        - grafana
      networks:
        - aeropulse-net

networks:
  aeropulse-net:
    driver: bridge

volumes:
  mosquitto_data:
  mosquitto_config:
  influxdb_data:
  grafana_data:
```

Г.2 Лістинг повного вихідного коду вбудованого програмного забезпечення мікроконтролера ESP32

```
#include <WiFi.h>
```

```
#include <WiFiManager.h>
#include <FS.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <TFT_eSPI.h>
#include <Adafruit_BME680.h>


const char* mqtt_server = "broker.emqx.io"; // IP-адреса хоста з Docker-
контейнерами

#define BTN_PIN 13
#define LED_G   12
#define LED_Y   14
#define LED_R   27

WiFiClient espClient;
PubSubClient client(espClient);
TFT_eSPI tft = TFT_eSPI();
Adafruit_BME680 bme;
HardwareSerial pmsSerial(2);

struct SystemData {
  float temp, hum, pres, gas, score;
  uint16_t pm25;
} sys;

int currentPage = 0;
unsigned long lastUpdate = 0;
unsigned long lastBtnPress = 0;
unsigned long lastMqttSend = 0;

unsigned long btnPressStart = 0;
bool isResetCountdown = false;
int lastSecondsLeft = -1;


void setup_wifi() {
  tft.fillRect(0, 0, 240, 22, TFT_BLUE);
  tft.setCursor(5, 3, 2);
  tft.setTextColor(TFT_WHITE);
  tft.print("Starting AP Mode...");

  WiFiManager wm;
  bool res = wm.autoConnect("AeroPulse_Setup");

  if(!res) {
    tft.fillRect(0, 0, 240, 22, TFT_RED);
    tft.setCursor(5, 3, 2);
    tft.print("WiFi Failed! Restarting");
    delay(3000);
    ESP.restart();
  }

  tft.fillRect(0, 0, 240, 22, TFT_DARKGREY);
  tft.setCursor(5, 3, 2);
  tft.print("WiFi Connected!");
}
```

```
void reconnectMQTT() {
  static unsigned long lastRetry = 0;
  if (!client.connected() && millis() - lastRetry > 5000) {
    if (client.connect("ESP32_AeroPulse_Diploma")) {
      client.subscribe("diploma/commands");
    }
    lastRetry = millis();
  }
}

// ФУНКЦІЇ АНАЛІЗУ

float calculateHealthScore(int pm, float gas) {
  float s_pm = map(constrain(pm, 0, 50), 0, 50, 100, 0);
  float s_gas = map(constrain(gas/1000, 10, 150), 10, 150, 0, 100);
  return (min(s_pm, s_gas) * 0.7) + ((s_pm + s_gas)/2 * 0.3);
}

bool readPMS() {
  if (pmsSerial.available()) {
    while (pmsSerial.available() > 32) pmsSerial.read();
    if (pmsSerial.peek() != 0x42) { pmsSerial.read(); return false; }
    if (pmsSerial.available() < 32) return false;

    uint8_t buffer[32];
    pmsSerial.readBytes(buffer, 32);
    if (buffer[1] != 0x4D) return false;

    sys.pm25 = (buffer[12] << 8) | buffer[13];
    return true;
  }
  return false;
}

// ВІЗУАЛІЗАЦІЯ

void updateVisuals() {
  uint16_t barColor = (WiFi.status() == WL_CONNECTED) ? TFT_DARKGREY : TFT_MAROON;
  tft.fillRect(0, 0, 240, 22, barColor);
  tft.setCursor(5, 3, 2);
  tft.setTextColor(TFT_WHITE, barColor);
  tft.printf(" WiFi: %s | MQTT: %s",
             (WiFi.status() == WL_CONNECTED ? "OK" : "DIS"),
             (client.connected() ? "OK" : "OFF"));

  tft.setCursor(0, 35, 4);
  if (currentPage == 0) {
    tft.setTextColor(TFT_CYAN, TFT_BLACK);
    tft.println("--- ENVIRONMENT ---");

    uint16_t tColor = (sys.temp >= 18 && sys.temp <= 26) ? TFT_GREEN : (sys.temp <
18 ? TFT_BLUE : TFT_RED);
    tft.setTextColor(TFT_WHITE, TFT_BLACK); tft.print(" Temp: ");
    tft.setTextColor(tColor, TFT_BLACK); tft.printf("%.1f C \n", sys.temp);

    uint16_t hColor = (sys.hum >= 40 && sys.hum <= 60) ? TFT_GREEN : TFT_YELLOW;
    tft.setTextColor(TFT_WHITE, TFT_BLACK); tft.print(" Hum:  ");
    tft.setTextColor(hColor, TFT_BLACK); tft.printf("%.1f %% \n", sys.hum);
```

```
    tft.setTextColor(TFT_WHITE, TFT_BLACK); tft.print(" Pres: ");
    tft.setTextColor(TFT_CYAN, TFT_BLACK); tft.printf("%.0f hPa \n", sys.pres);
  } else {
    tft.setTextColor(TFT_MAGENTA, TFT_BLACK);
    tft.println(" --- AIR QUALITY ---");

    uint16_t pmColor = (sys.pm25 < 15) ? TFT_GREEN : (sys.pm25 < 35 ? TFT_YELLOW :
TFT_RED);
    tft.setTextColor(TFT_WHITE, TFT_BLACK); tft.print(" PM2.5: ");
    tft.setTextColor(pmColor, TFT_BLACK); tft.printf("%d ug/m3 \n", sys.pm25);

    uint16_t vColor = (sys.gas/1000.0 > 100) ? TFT_GREEN : (sys.gas/1000.0 > 40 ?
TFT_YELLOW : TFT_RED);
    tft.setTextColor(TFT_WHITE, TFT_BLACK); tft.print(" VOC:   ");
    tft.setTextColor(vColor, TFT_BLACK); tft.printf("%.1f kO \n", sys.gas/1000.0);

    tft.setCursor(10, 165, 4);
    uint16_t sColor = (sys.score > 75) ? TFT_GREEN : (sys.score > 40 ? TFT_YELLOW :
TFT_RED);
    tft.setTextColor(sColor, TFT_BLACK);
    tft.printf(" HEALTH: %.0f %%  ", sys.score);
  }
}

// SETUP TA LOOP

void setup() {
  Serial.begin(115200);
  pmsSerial.begin(9600, SERIAL_8N1, 16, 17);

  pinMode(32, OUTPUT); digitalWrite(32, HIGH);
  pinMode(BTN_PIN, INPUT_PULLDOWN);
  pinMode(LED_G, OUTPUT); pinMode(LED_Y, OUTPUT); pinMode(LED_R, OUTPUT);

  tft.init(); tft.setRotation(1); tft.fillScreen(TFT_BLACK);

  if (!bme.begin(0x76)) {
    tft.setTextColor(TFT_RED); tft.drawString("BME680 ERR", 10, 100, 4);
    while (1);
  }
  bme.setGasHeater(320, 150);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  bool currentBtnState = (digitalRead(BTN_PIN) == HIGH);
  static bool lastBtnState = false;

  if (currentBtnState == true && lastBtnState == false && (millis() - lastBtnPress
> 300)) {
    lastBtnPress = millis();
    btnPressStart = millis();

    if (!isResetCountdown) {
      currentPage = !currentPage;
      tft.fillScreen(TFT_BLACK);
      updateVisuals();
```

```
    lastUpdate = millis();
  }
}

if (currentBtnState == true && btnPressStart > 0) {
  unsigned long pressDuration = millis() - btnPressStart;

  if (pressDuration >= 1000) {
    if (!isResetCountdown) {
      isResetCountdown = true;
      tft.fillScreen(TFT_BLACK);
      tft.setTextColor(TFT_RED, TFT_BLACK);
      tft.drawCentreString("!!! FACTORY RESET !!!", 120, 25, 4);
      tft.setTextColor(TFT_WHITE, TFT_BLACK);
      tft.drawCentreString("Hold button to wipe WiFi", 120, 65, 2);
    }

    int secondsPassed = (pressDuration - 1000) / 1000;
    int secondsLeft = 4 - secondsPassed;

    if (secondsLeft >= 0 && secondsLeft != lastSecondsLeft) {
      lastSecondsLeft = secondsLeft;

      uint16_t numColor = TFT_GREEN;
      if (secondsLeft <= 1) numColor = TFT_RED;
      else if (secondsLeft == 2) numColor = TFT_YELLOW;

      tft.fillRect(90, 105, 60, 60, TFT_BLACK);
      tft.setTextColor(numColor, TFT_BLACK);
      tft.setCursor(108, 110, 7);
      tft.printf("%d", secondsLeft + 1);
    }

    if (pressDuration >= 5000) {
      tft.fillScreen(TFT_MAROON);
      tft.setTextColor(TFT_WHITE);
      tft.drawCentreString("SYSTEM WIPED!", 120, 90, 4);
      tft.drawCentreString("Restarting...", 120, 130, 2);
      delay(2000);

      WiFiManager wm;
      wm.resetSettings();
      ESP.restart();
    }
  }
}

if (currentBtnState == false) {
  if (lastBtnState == true) {
      lastBtnPress = millis();
  }

  if (isResetCountdown) {
    isResetCountdown = false;
    lastSecondsLeft = -1;
    btnPressStart = 0;
    tft.fillScreen(TFT_BLACK);
    updateVisuals();
  } else {
```

```
      btnPressStart = 0;
    }
  }

  lastBtnState = currentBtnState;

  // МЕНЕДЖЕР МЕРЕЖІ ТА ДАТЧИКІВ
  if (!isResetCountdown) {
    if (WiFi.status() == WL_CONNECTED) {
      if (!client.connected()) reconnectMQTT();
      client.loop();
    }

    if (millis() - lastUpdate > 1000) {
      if (bme.performReading()) {
        sys.temp = bme.temperature; sys.hum = bme.humidity;
        sys.pres = bme.pressure / 100.0; sys.gas = bme.gas_resistance;
      }
      readPMS();
      sys.score = calculateHealthScore(sys.pm25, sys.gas);

      digitalWrite(LED_G, sys.score > 75);
      digitalWrite(LED_Y, sys.score <= 75 && sys.score > 40);
      digitalWrite(LED_R, sys.score <= 40);

      updateVisuals();
      lastUpdate = millis();
    }

    if (millis() - lastMqttSend > 15000 && client.connected()) {
      char msg[256];
      snprintf(msg, sizeof(msg),

"{\"temp\":%.2f,\"hum\":%.2f,\"pres\":%.1f,\"voc\":%.1f,\"pm25\":%d,\"score\":%.2f}
",
               sys.temp, sys.hum, sys.pres, sys.gas/1000.0, sys.pm25, sys.score);

      client.publish("diploma/sensors/air", msg);
      lastMqttSend = millis();
    }
  }
}
```

Г.3 Вихідний код головного модуля асинхронного Telegram-бота (bot.py)

```
import asyncio
import json
import logging
import sys
import os
import time
import aiomqtt
from aiogram import Bot, Dispatcher
from dotenv import load_dotenv

load_dotenv()
from handlers import router, shared_sensor_data, shared_system_state
```

```
if sys.platform == 'win32':
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())

BOT_TOKEN = os.getenv("BOT_TOKEN")
ADMIN_CHAT_ID = os.getenv("ADMIN_CHAT_ID")
MQTT_BROKER = "mosquitto"  # Для Docker
MQTT_TOPIC = "diploma/sensors/air"

if not BOT_TOKEN or not ADMIN_CHAT_ID:
    exit("❌ Помилка: BOT_TOKEN або ADMIN_CHAT_ID не знайдено у файлі .env!")

history_buffer = {
    "pm25": [], "voc": [], "temp": []
}
HISTORY_WINDOW = 300

active_alerts = {
    "pm25_danger": None, "pm25_warning": None,
    "voc_danger": None,  "voc_warning": None,
    "temp_danger": None, "hum_warning": None
}

last_spam_time = {"pm25": 0, "voc": 0, "temp": 0, "hum": 0}
COOLDOWN = 3600

def update_trend_history(current_time, data):
    for key in ["pm25", "voc", "temp"]:
        if key in data and data[key] != "N/A":
            history_buffer[key].append((current_time, float(data[key])))
            history_buffer[key] = [(t, v) for t, v in history_buffer[key] if
current_time - t <= HISTORY_WINDOW]

async def check_and_send_alerts(bot: Bot, data: dict):
    current_time = time.time()
    update_trend_history(current_time, data)
    smart_mode = shared_system_state["smart_mode"]

    try:
        pm25 = float(data.get("pm25", 0))
        voc = float(data.get("voc", 100))
        temp = float(data.get("temp", 22))
        hum = float(data.get("hum", 50))

        if smart_mode:
            oldest_pm25 = history_buffer["pm25"][0][1] if history_buffer["pm25"]
else pm25
            oldest_voc = history_buffer["voc"][0][1] if history_buffer["voc"] else
voc
            oldest_temp = history_buffer["temp"][0][1] if history_buffer["temp"]
else temp

            if (pm25 - oldest_pm25) > 10 and current_time - last_spam_time["pm25"]
> 600:
                await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"📈
<b>АНОМАЛІЯ:</b> Різкий стрибок пилу!\nЗростання до {pm25} µg/m³ за останні 5 хв.",
parse_mode="HTML")
                last_spam_time["pm25"] = current_time
```

```
            if (oldest_voc - voc) > 20 and current_time - last_spam_time["voc"] >
600:
                await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"📉
<b>АНОМАЛІЯ:</b> Різке забруднення повітря газами!\nПадіння опору до {voc} kΩ за 5
хв.", parse_mode="HTML")
                last_spam_time["voc"] = current_time

            if (temp - oldest_temp) > 3 and current_time - last_spam_time["temp"] >
600:
                await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"🔥
<b>ПОПЕРЕДЖЕННЯ:</b> Температура стрімко зростає!\n+{temp - oldest_temp:.1f} °C за
5 хв. Перевірте приміщення.", parse_mode="HTML")
                last_spam_time["temp"] = current_time

        if pm25 > 35:
            if not active_alerts["pm25_danger"] or current_time -
last_spam_time["pm25"] > COOLDOWN:
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"😷
<b>КРИТИЧНО:</b> Пил PM2.5 &gt; 35 ({pm25}).\nНегайно увімкніть очищувач!",
parse_mode="HTML")
                active_alerts["pm25_danger"] = msg.message_id
                last_spam_time["pm25"] = current_time

        elif pm25 > 15 and smart_mode:
            active_alerts["pm25_danger"] = None
            if not active_alerts["pm25_warning"]:
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"⚠️
<b>Увага:</b> Підвищений фон пилу ({pm25}).", parse_mode="HTML")
                active_alerts["pm25_warning"] = msg.message_id

        elif pm25 <= 12:
            if active_alerts["pm25_danger"] or active_alerts["pm25_warning"]:
                reply_id = active_alerts["pm25_danger"] or
active_alerts["pm25_warning"]
                await bot.send_message(chat_id=ADMIN_CHAT_ID, text="✅
<b>Стабілізація:</b> Рівень пилу повернувся в норму.", parse_mode="HTML",
reply_to_message_id=reply_id)
            active_alerts["pm25_danger"] = None
            active_alerts["pm25_warning"] = None

        if voc < 40:
            if not active_alerts["voc_danger"] or current_time -
last_spam_time["voc"] > COOLDOWN:
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"☣️
<b>КРИТИЧНО:</b> Гази/VOC на небезпечному рівні ({voc} kΩ).\nПровітріть!",
parse_mode="HTML")
                active_alerts["voc_danger"] = msg.message_id
                last_spam_time["voc"] = current_time

        elif voc <= 100 and smart_mode:
            active_alerts["voc_danger"] = None
            if not active_alerts["voc_warning"]:
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"⚠️
<b>Увага:</b> Повітря застоюється ({voc} kΩ).", parse_mode="HTML")
                active_alerts["voc_warning"] = msg.message_id

        elif voc >= 110:
```

```
            if active_alerts["voc_danger"] or active_alerts["voc_warning"]:
                reply_id = active_alerts["voc_danger"] or
active_alerts["voc_warning"]
                await bot.send_message(chat_id=ADMIN_CHAT_ID, text="✅
<b>Стабілізація:</b> Повітря свіже, рівень VOC у нормі.", parse_mode="HTML",
reply_to_message_id=reply_id)
            active_alerts["voc_danger"] = None
            active_alerts["voc_warning"] = None

        if temp > 30:
             if not active_alerts["temp_danger"] or current_time -
last_spam_time["temp"] > COOLDOWN:
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"🔥
<b>КРИТИЧНО:</b> Спека або перегрів приміщення ({temp} °C).", parse_mode="HTML")
                active_alerts["temp_danger"] = msg.message_id
                last_spam_time["temp"] = current_time

        elif temp <= 28 and active_alerts["temp_danger"]:
             await bot.send_message(chat_id=ADMIN_CHAT_ID, text="✅
<b>Стабілізація:</b> Температура знизилась до норми.", parse_mode="HTML",
reply_to_message_id=active_alerts["temp_danger"])
             active_alerts["temp_danger"] = None

        if smart_mode and (hum < 30 or hum > 70):
            if not active_alerts["hum_warning"] or current_time -
last_spam_time["hum"] > COOLDOWN:
                status = "Сухо" if hum < 30 else "Вологo"
                msg = await bot.send_message(chat_id=ADMIN_CHAT_ID, text=f"💧
<b>Дискомфорт:</b> {status} ({hum}%).", parse_mode="HTML")
                active_alerts["hum_warning"] = msg.message_id
                last_spam_time["hum"] = current_time

        elif 35 <= hum <= 65 and active_alerts["hum_warning"]:
             await bot.send_message(chat_id=ADMIN_CHAT_ID, text="✅
<b>Стабілізація:</b> Вологість повітря нормалізувалася.", parse_mode="HTML",
reply_to_message_id=active_alerts["hum_warning"])
             active_alerts["hum_warning"] = None

    except Exception as e:
        logging.error(f"Помилка аналізу алертів: {e}")

async def mqtt_listener(bot: Bot):
    while True:
        try:
            async with aiomqtt.Client(MQTT_BROKER) as client:
                await client.subscribe(MQTT_TOPIC)
                logging.info(f"✅ MQTT підключено. Слухаю топік: {MQTT_TOPIC}")

                async for message in client.messages:
                    payload = message.payload.decode()
                    new_data = json.loads(payload)

                    shared_sensor_data.update(new_data)
                    await check_and_send_alerts(bot, new_data)

        except aiomqtt.MqttError as error:
            logging.error(f"❌ Помилка MQTT: {error}. Перепідключення...")
```

```
            await asyncio.sleep(5)
        except Exception as e:
            logging.error(f"❌ Критична помилка MQTT: {e}")
            await asyncio.sleep(5)

async def main():
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
    bot = Bot(token=BOT_TOKEN)
    dp = Dispatcher()
    dp.include_router(router)

    logging.info("🚀 Запуск AeroPulse Telegram Bot...")
    task_bot = asyncio.create_task(dp.start_polling(bot))
    task_mqtt = asyncio.create_task(mqtt_listener(bot))

    await asyncio.gather(task_bot, task_mqtt)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\n🛑 Бот зупинено вручну.")
```

Лістинг Г.4 Вихідний код модуля обробки повідомлень та інтерфейсу (handlers.py)

```
import os
import aiohttp
from aiogram import Router, F
from aiogram.types import (Message, ReplyKeyboardMarkup, KeyboardButton,
                           InlineKeyboardMarkup, InlineKeyboardButton,
CallbackQuery, BufferedInputFile)
from aiogram.filters import CommandStart
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext

router = Router()

GRAFANA_INTERNAL_URL = os.getenv("GRAFANA_INTERNAL_URL", "http://grafana:3000")
GRAFANA_EXTERNAL_URL = os.getenv("GRAFANA_EXTERNAL_URL", "http://127.0.0.1:3000")
DASHBOARD_ID = os.getenv("DASHBOARD_ID")
GRAFANA_TOKEN = os.getenv("GRAFANA_TOKEN")

PANEL_IDS = {
    "temp": 1,
    "hum": 2,
    "pm25": 4,
    "voc": 6
}

class GraphStates(StatesGroup):
    waiting_for_period = State()

shared_sensor_data = {
    "temp": "N/A", "hum": "N/A", "pm25": "N/A", "voc": "N/A", "pres": "N/A",
"score": "N/A"
```

```
}

shared_system_state = {
    "smart_mode": False
}

def get_expert_analysis(pm25, voc, temp, hum):
    """Експертний висновок із безпечним HTML-форматуванням"""
    if pm25 == "N/A" or voc == "N/A" or temp == "N/A":
        return "<i>□ Дані для аналізу ще не зібрані...</i>"

    try:
        pm25_val = float(pm25)
        voc_val = float(voc)
        temp_val = float(temp)
        hum_val = float(hum)

        is_critical = pm25_val > 35 or voc_val < 40 or temp_val > 30
        is_warning = (15 < pm25_val <= 35) or (40 <= voc_val <= 100) or (hum_val <
30 or hum_val > 70)

        if is_critical:
            health_status = "● <b>ІНДЕКС ПОВІТРЯ: НЕБЕЗПЕЧНО</b>\nКритичний рівень
показників! Негайні дії!"
        elif is_warning:
            health_status = "□ <b>ІНДЕКС ПОВІТРЯ: ЗАДОВІЛЬНО</b>\nЯкість прийнятна,
але є відхилення від ідеалу."
        else:
            health_status = "□ <b>ІНДЕКС ПОВІТРЯ: ВІДМІННО</b>\nПовітря ідеальне
для дихання та роботи."

        warnings = []
        if pm25_val > 35:
            warnings.append("🏭 <b>Пил (PM2.5):</b> Небезпечний рівень (&gt;35).
Зачиніть вікна!")
        elif pm25_val > 15:
            warnings.append("⚠ <b>Пил (PM2.5):</b> Злегка підвищений фон
(&gt;15).")

        if voc_val < 40:
            warnings.append("🏭 <b>Гази (VOC):</b> Критичне забруднення (&lt;40
kΩ). Негайне провітрювання!")
        elif voc_val <= 100:
            warnings.append("⚠ <b>Гази (VOC):</b> Повітря починає застоюватись
(&lt;100 kΩ).")

        if temp_val > 30:
            warnings.append("🔥 <b>Температура:</b> Аномально спекотно!")
        elif temp_val < 18:
            warnings.append("❄ <b>Температура:</b> Холодно, є ризик
переохолодження.")

        if hum_val < 30:
            warnings.append("🏜 <b>Вологість:</b> Дуже сухе повітря, увімкніть
зволожувач.")
        elif hum_val > 70:
```

```
            warnings.append("💦 <b>Вологість:</b> Надто волого, ризик появи
цвілі.")

        details = "\n\n" + "\n".join(warnings) if warnings else "\n\n✅ Всі
показники в межах ідеальної норми."

        return f"{health_status}{details}"

    except ValueError:
        return "<i>Помилка генерації висновку.</i>"

def parse_user_time_period(text: str):
    clean_text = text.lower().replace(" ", "").replace(",", ".")
    unit = None
    num_str = ""
    if clean_text.endswith("хв"):
        unit = "m"
        num_str = clean_text.replace("хв", "")
    elif clean_text.endswith("год"):
        unit = "h"
        num_str = clean_text.replace("год", "")
    elif clean_text.endswith("д"):
        unit = "d"
        num_str = clean_text.replace("д", "")
    else:
        return None, "⚠ Неправильний формат (використовуйте хв, год, д)."

    try:
        value = float(num_str)
    except ValueError:
        return None, "⚠ Не вдалося розпізнати число."

    if value <= 0: return None, "⚠ Число має бути більшим за 0."

    total_minutes = 0
    if unit == "m":
        total_minutes = int(value)
        grafana_str = f"{total_minutes}m"
    elif unit == "h":
        total_minutes = int(value * 60)
        grafana_str = f"{int(value)}h" if value.is_integer() else
f"{total_minutes}m"
    elif unit == "d":
        total_minutes = int(value * 24 * 60)
        grafana_str = f"{int(value)}d" if value.is_integer() else f"{int(value *
24)}h"

    if total_minutes < 5: return None, "❌ Мінімум 5 хв."
    if total_minutes > 43200: return None, "❌ Максимум 30 д."
    return grafana_str, None

def get_reply_keyboard():
    smart_status = "УВІМК ✅" if shared_system_state["smart_mode"] else "ВИМК ❌"

    return ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text="📊 Поточний стан")],
```

```
            [KeyboardButton(text="📈 Графіки та Аналітика")],
            [KeyboardButton(text=f"🤖 Розумні сповіщення: {smart_status}")]
        ],
        resize_keyboard=True,
        input_field_placeholder="Оберіть дію..."
    )

def get_metrics_keyboard():
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="🌡 Температура",
callback_data="graph_metric_temp"),
         InlineKeyboardButton(text="💧 Вологість",
callback_data="graph_metric_hum")],
        [InlineKeyboardButton(text="🌫 Пил PM2.5",
callback_data="graph_metric_pm25"),
         InlineKeyboardButton(text="☁ VOC (Гази)",
callback_data="graph_metric_voc")]
    ])

def get_time_keyboard(metric: str):
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="1 год",
callback_data=f"graph_render_{metric}_1h"),
         InlineKeyboardButton(text="6 год",
callback_data=f"graph_render_{metric}_6h"),
         InlineKeyboardButton(text="24 год",
callback_data=f"graph_render_{metric}_24h")],
        [InlineKeyboardButton(text="🔧 Налаштувати свій період",
callback_data=f"graph_input_{metric}")],
        [InlineKeyboardButton(text="⬅ Назад",
callback_data="graph_back_to_metrics")]
    ])

def get_grafana_link_keyboard():
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="🌐 Відкрити Live-Дашборд",
url=f"{GRAFANA_EXTERNAL_URL}/d/{DASHBOARD_ID}")]
    ])

@router.message(CommandStart())
async def cmd_start(message: Message):
    text = "👋 <b>Привіт! Я система AeroPulse.</b>\nВикористовуйте меню внизу
екрана:"
    await message.answer(text, parse_mode="HTML",
reply_markup=get_reply_keyboard())

@router.message(F.text.startswith("🤖 Розумні сповіщення"))
async def toggle_smart_mode(message: Message):
    shared_system_state["smart_mode"] = not shared_system_state["smart_mode"]

    if shared_system_state["smart_mode"]:
        text = "🤖 <b>Розумний режим УВІМКНЕНО</b>.\nТепер я слідкую за трендами,
різкими стрибками та відхиленнями."
    else:
        text = "⚙ <b>Розумний режим ВИМКНЕНО</b>.\nТепер я повідомлю вас ТІЛЬКИ
про критичну небезпеку."
```

```
    await message.answer(text, parse_mode="HTML",
reply_markup=get_reply_keyboard())

@router.message(F.text == "📊 Поточний стан")
async def process_status_btn(message: Message):
    analysis_text = get_expert_analysis(
        shared_sensor_data['pm25'],
        shared_sensor_data['voc'],
        shared_sensor_data['temp'],
        shared_sensor_data['hum']
    )
    text = (
        "📊 <b>ДЕТАЛІЗАЦІЯ СЕНСОРІВ:</b>\n"
        "〰〰〰〰〰〰〰〰〰〰〰〰\n"
        f"🌡 Температура: <b>{shared_sensor_data['temp']} °C</b>\n"
        f"💧 Вологість: <b>{shared_sensor_data['hum']} %</b>\n"
        f"💨 Пил PM2.5: <b>{shared_sensor_data['pm25']} µg/m³</b>\n"
        f"🧪 Опір VOC: <b>{shared_sensor_data['voc']} kΩ</b>\n"
        f"🎈 Тиск: <b>{shared_sensor_data['pres']} hPa</b>\n"
        f"{analysis_text}"
    )
    await message.answer(text, parse_mode="HTML")

@router.message(F.text == "📈 Графіки та Аналітика")
async def process_charts_btn(message: Message):
    await message.answer("📊 <b>Генерація графіка</b>\n\nОберіть параметр, який
хочете переглянути:", parse_mode="HTML", reply_markup=get_metrics_keyboard())

@router.callback_query(F.data == "graph_back_to_metrics")
async def back_to_metrics(callback: CallbackQuery):
    await callback.message.edit_text("📊 <b>Генерація графіка</b>\n\nОберіть
параметр, який хочете переглянути:", parse_mode="HTML",
reply_markup=get_metrics_keyboard())
    await callback.answer()

@router.callback_query(F.data.startswith("graph_metric_"))
async def choose_time_range(callback: CallbackQuery):
    metric = callback.data.split("_")[2]
    metric_names = {"temp": "Температура", "hum": "Вологість", "pm25": "Пил PM2.5",
"voc": "VOC"}
    await callback.message.edit_text(f"🕒 Оберіть часовий діапазон для графіка
<b>{metric_names.get(metric, 'Обраний параметр')}</b>:", parse_mode="HTML",
reply_markup=get_time_keyboard(metric))
    await callback.answer()

async def send_rendered_graph(message_to_edit: Message, metric: str,
time_range_str: str):
    panel_id = PANEL_IDS.get(metric, 1)

    render_url = f"{GRAFANA_INTERNAL_URL}/render/d-
solo/{DASHBOARD_ID}/aeropulse?orgId=1&panelId={panel_id}&from=now-
{time_range_str}&to=now&width=1000&height=500&theme=dark&tz=Europe/Kyiv"
    headers = {"Authorization": f"Bearer {GRAFANA_TOKEN}"} if GRAFANA_TOKEN else {}
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(render_url, headers=headers) as response:
```

```
                if response.status == 200 and 'image' in
response.headers.get('Content-Type', ''):
                    image_bytes = await response.read()
                    photo = BufferedInputFile(image_bytes,
filename=f"{metric}_{time_range_str}.png")
                    await message_to_edit.answer_photo(photo=photo, caption=f"
Графік за період: {time_range_str}.", reply_markup=get_grafana_link_keyboard())
                    await message_to_edit.delete()
                else:
                    await message_to_edit.edit_text("
Grafana.")
    except Exception as e:
        await message_to_edit.edit_text(f"
@router.callback_query(F.data.startswith("graph_render_"))
async def render_standard_graph(callback: CallbackQuery):
    parts = callback.data.split("_")
    await callback.message.edit_text("
parse_mode="HTML")
    await send_rendered_graph(callback.message, parts[2], parts[3])
    await callback.answer()

@router.callback_query(F.data.startswith("graph_input_"))
async def start_custom_time_fsm(callback: CallbackQuery, state: FSMContext):
    await state.update_data(chosen_metric=callback.data.split("_")[2])
    await state.set_state(GraphStates.waiting_for_period)
    await callback.message.edit_text("
періоду:</b>\nВведіть значення (напр: <code>45 хв</code>, <code>1.5 год</code>):",
parse_mode="HTML")
    await callback.answer()

@router.message(GraphStates.waiting_for_period)
async def process_custom_period_input(message: Message, state: FSMContext):
    grafana_time_str, error_msg = parse_user_time_period(message.text.strip())
    if error_msg:
        await message.answer(error_msg, parse_mode="HTML")
        return
    data = await state.get_data()
    await state.clear()
    status_msg = await message.answer("
parse_mode="HTML")
    await send_rendered_graph(status_msg, data.get("chosen_metric"),
grafana_time_str)
```

Лістинг Г.5 Конфігураційні файли середовища розгортання бота (Dockerfile, requirements.txt)

Dockerfile:

```
# Офіційний легкий образ Python
FROM python:3.11-slim

# Робоча директорія всередині контейнера
WORKDIR /app

# Копіювання файлу із залежностями та їхнє встановлення
```

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Копіювєвання іншого коду бота
COPY . .

# Команда для запуску бота
CMD ["python", "bot.py"]
```

requirements.txt:

```
aiogram>=3.4.0
aiomqtt>=2.0.0
python-dotenv>=1.0.0
```