МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

СИСТЕМА МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ

Виконав: студент групи 1К-22
спеціальності
123 Комп’ютерна інженерія
Валерій САЙКО
Керівник:
Майя ЛЮТА

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп’ютерної інженерії та інформаційних технологій

Спеціальність 123 «Комп’ютерна інженерія»

Освітня програма Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувачка циклової комісії КІ та ІТ
_____ Наталя ФАЛЬЧЕНКО
(підпис)
«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сайко Валерію Валерійовичу

1. Тема кваліфікаційної роботи Система моніторингу IoT-пристроїв
Керівник роботи Люта Майя В’ячеславівна, викладач вищої категорії
затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2025 року № 84/1-у.
2. Строк подання студентом кваліфікаційної роботи 02.06.2026
3. Вихідні дані до кваліфікаційної роботи: роботи апаратна платформа периферійного вузла збору даних; прикладний протокол передачі даних – легковаговий асинхронний протокол MQTT; технічна документація щодо стандартів передачі даних; технологія розгортання та ізоляції серверних сервісів – середовище контейнеризації Docker; архітектура серверного стеку: open-source брокер повідомлень Eclipse Mosquitto, агент збору метрик Telegraf, база даних часових рядів InfluxDB, аналітична платформа візуалізації та моніторингу Grafana; середовище моделювання та віртуалізації апаратної частини – VS Code / Wokwi Gateway.
4. Зміст кваліфікаційної роботи Вступ. Розділ 1. Аналіз сучасного стану та інженерних рішень у сфері моніторингу IoT-пристроїв. Розділ 2. Проєктування архітектури, структурної схеми та мережевої топології системи. Розділ 3. Інтеграція, конфігурація серверної інфраструктури Docker та тестування системи в середовищі VS Code/Wokwi. Висновки. Список використаних джерел. Додатки (конфігураційні файли Docker Compose та лістинги коду).
5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1 Огляд існуючих IoT систем	09.12.2025	
3	Розділ 2 Проєктування системи моніторингу	10.03.2026	
4	Розділ 3 Розробка та тестування системи	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачці циклової комісії (за 7 днів до захисту)	10.06.2026	

Студент _____ Валерій САЙКО
(підпис)

Керівник роботи _____ Майя ЛЮТА
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробленню оптимізованої системи моніторингу IoT-пристроїв. Розроблено програмно-апаратний комплекс для дистанційного збору та обробки телеметричної інформації, що базується на технологіях Інтернету речей та контейнеризованій серверній інфраструктурі.

Проаналізовано сучасні протоколи обміну даними в IoT-мережах, виявлено проблему надлишкового обчислювального навантаження на автономні датчики при використанні традиційної серіалізації текстових пакетів (JSON) та обґрунтовано концепцію трансляції атомарних числових повідомлень (plain float/int).

Розроблено архітектуру розподіленої обробки даних на стороні сервера, що дозволило повністю змістити обчислювальні операції з периферійних вузлів на хост-машину. Реалізовано логіку динамічного розбору (парсингу) MQTT-топіків за допомогою сервісного агента для автоматичного структурування потоків телеметрії, ідентифікації приміщень і формування масивів часових рядів без додаткового навантаження на мікроконтролер.

Використано сучасний стек технологій: мікроконтролер ESP32, програмну платформу ESPHome, брокер повідомлень Eclipse Mosquitto, агент збору метрик Telegraf, базу даних часових рядів InfluxDB, аналітичну платформу Grafana та інструменти оркестрації Docker Compose. Проведено комплексне тестування працездатності системи в умовах емуляційного моделювання (платформа Wokwi), верифіковано стабільність передачі даних та надійність взаємодії мережевих компонентів.

Результати роботи можна використати для дистанційного контролю мікроклімату в серверних кімнатах, дата-центрах, житлових будинках, а також на промислових об'єктах (IIoT). Розроблене рішення є готовим до розгортання та легко масштабується.

Ключові слова: ІНТЕРНЕТ РЕЧЕЙ (IoT), MQTT, DOCKER, TELEGRAF, INFLUXDB, GRAFANA, МОНІТОРИНГ, ЕМУЛЯЦІЙНЕ МОДЕЛЮВАННЯ.

ABSTRACT

The qualifying paper is devoted to the development of an optimized monitoring system for IoT devices. A software and hardware complex for remote collection and processing of telemetric information has been developed, based on Internet of Things technologies and a containerized server infrastructure.

Modern data exchange protocols in IoT networks are analyzed, the problem of excessive computational load on autonomous sensors when using traditional serialization of text packets (JSON) is identified, and the concept of transmitting atomic numerical messages (plain float/int) is substantiated.

A distributed data processing architecture on the server side has been developed, allowing to completely shift computational operations from peripheral nodes to the host machine. The logic of dynamic parsing of MQTT topics using a service agent has been implemented for automatic structuring of telemetry streams, identification of premises, and formation of time-series arrays without additional load on the microcontroller.

A modern technology stack was used: ESP32 microcontroller, ESPHome software platform, Eclipse Mosquitto message broker, Telegraf metric collection agent, InfluxDB time-series database, Grafana analytics platform, and Docker Compose orchestration tools. Comprehensive performance testing of the system was conducted under emulation modeling conditions (Wokwi platform), verifying the stability of data transmission and the reliability of network component interaction.

The results of the work can be used for remote microclimate control in server rooms, data centers, residential buildings, as well as at industrial facilities (IIoT). The developed solution is ready for deployment and scales easily.

Keywords: INTERNET OF THINGS (IoT), MQTT, DOCKER, TELEGRAF, INFLUXDB, GRAFANA, MONITORING, EMULATION MODELING.

ЗМІСТ

ВСТУП

Актуальність теми пов’язана зі стрімким розвитком концепції Інтернету речей (IoT) зумовлює необхідність створення ефективних систем моніторингу, здатних стабільно працювати в умовах обмежених ресурсів кінцевих пристроїв та нестабільного мережевого з’єднання. Традиційні протоколи часто є занадто перевантаженими для автономних датчиків, що робить актуальним впровадження легковагових рішень, таких як MQTT. Також безпека передачі та візуалізація даних у реальному часі залишаються важливими завданнями для сучасних інженерних рішень, що показує необхідність комплексної розробки систем дистанційного контролю.

Мета роботи полягає у проєктуванні та практичній реалізації оптимізованої системи моніторингу IoT-пристроїв із використанням протоколу MQTT та архітектури розподіленої обробки даних, що забезпечує надійну передачу інформації, її збереження у базі даних та подальшу візуалізацію в реальному часі.

Об’єктом дослідження є система дистанційного збору та обробки даних у мережах Інтернету речей.

Предметом дослідження виступають програмно-апаратні засоби реалізації, протоколи передачі даних, методи структурування потоків числових метрик та алгоритми візуалізації в системі моніторингу IoT-пристроїв.

Завдання:

1. Провести аналіз сучасних протоколів обміну даними та методів організації IoT мереж.

2. Зробити обґрунтований вибір програмно-апаратних засобів для проєктування системи.

3. Розробити структурну схему взаємодії елементів системи моніторингу.

4. Спроєктувати мережеву топологію системи та визначити логіку процесів збору, серіалізації й транспортування телеметрії до сервера.

5. Здійснити програмне моделювання роботи системи та налаштувати віртуальну панель моніторингу.

6. Провести тестування працездатності розробленої моделі, перевірити стабільність передачі даних та оцінити надійність взаємодії мережевих компонентів.

Практичне значення отриманих результатів полягає у створенні готового до впровадження програмно-апаратного рішення для оптимізованого моніторингу IoT-пристроїв. Результати роботи можуть бути безпосередньо використані у таких сферах, як контроль клімату в серверних кімнатах та дата-центрах, промисловий Інтернет речей (IIoT), сільське господарство та екологія, системи «розумного будинку».

Апробація результатів дослідження була здійснена в межах XVIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Тенденції розвитку ІТ-технологій в Україні» [1].

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ІОТ СИСТЕМ

1.1 Архітектура та принципи функціонування мереж IoT

Основою для побудови сучасних систем Інтернету речей (IoT) є архітектурний підхід, що базується на ієрархічному розподілі функцій. Найбільш поширеною в інженерній практиці є класична трирівнева модель. Вона дозволяє систематизувати процес збору, передачі та обробки даних, забезпечуючи чітке розмежування між апаратним забезпеченням та програмними сервісами.

Трирівнева архітектура включає рівень сприйняття (Perception Layer), мережевий рівень (Network Layer) та прикладний рівень (Application Layer). У класичній теорії комп'ютерної інженерії та міжнародних стандартах IoT ця модель розглядається як базовий шаблон для проектування розподілених систем моніторингу [2, 3]. Кожен із цих рівнів виконує специфічний набір завдань та використовує відповідні технологічні стеки.

Рівень сприйняття є нижнім ярусом ієрархії та безпосередньо взаємодіє з фізичним середовищем. Його основним завданням є ідентифікація об'єктів та збір фізичної інформації. У контексті систем моніторингу на цьому рівні функціонують: датчики та сенсори (температури, вологості, тиску, освітленості тощо), які перетворюють аналогові фізичні величини у цифровий сигнал; виконавчі механізми (актуатори), що здійснюють зворотний вплив на середовище за командою системи; пристрої ідентифікації (RFID-мітки, QR-коди). На цьому рівні критично важливими є питання точності сигналу, енергоспоживання кінцевих вузлів та стабільності роботи сенсорів у різних експлуатаційних умовах. Отримані на цьому етапі дані є первинними і потребують подальшої обробки та нормалізації для коректної передачі на вищі рівні архітектури системи.

Мережевий рівень відіграє роль сполучної ланки між фізичними пристроями та обчислювальними потужностями. Його головна функція полягає у надійній та безпечній передачі даних, отриманих від рівня

сприйняття. Оскільки пристрої IoT часто мають обмежені обчислювальні ресурси, мережевий рівень базується на використанні різноманітних технологій зв’язку. Узагальнені технічні характеристики та типові сценарії використання основних технологій мережевого рівня в системах моніторингу наведено в таблиці 1.1 [4, 5].

Таблиця 1.1 – Класифікація та характеристики технологій зв’язку мережевого рівня IoT

Тип інтерфейсів	Технології / Протоколи	Радіус дії	Енергоспоживання	Типовий сценарій використання в моніторингу
Дротові	Ethernet, RS-485	Обмежений довжиною кабелю	Стаціонарне живлення	Промислові об’єкти, стаціонарні стійки обладнання з високими вимогами до завадостійкості.
Бездротові короткого радіусу	Wi-Fi, Bluetooth, ZigBee	До 10–100 метрів	Середнє / Низьке	Локальний моніторинг у межах приміщення, "розумний будинок", лабораторні стенди.
Енергоефективні дальнього радіусу (LPWAN)	LoRaWAN, NB-IoT	До 5–15 км	Дуже низьке (робота від батарей)	Розподілені автономні системи, екологічний моніторинг, збір телеметрії на великих площа

Також важливими є шлюзи (Gateways), які виконують роль посередників. Вони здійснюють агрегацію даних від декількох кінцевих вузлів, проводять попередню фільтрацію трафіку та забезпечують маршрутизацію пакетів до центральних серверів або хмарних сховищ. На цьому етапі вирішуються питання цілісності даних та інтероперабельності – здатності різних пристроїв взаємодіяти між собою в межах єдиного інформаційного простору.

Прикладний рівень (Application Layer) є верхньою межею архітектури, з якою безпосередньо взаємодіє кінцевий користувач або бізнес-логіка системи. Саме тут відбувається інтерпретація отриманих даних та їх перетворення на корисну інформацію. Функціонал цього рівня охоплює: зберігання великих масивів даних у базах даних, аналітичну обробку та виявлення закономірностей у телеметричних даних, візуалізацію результатів моніторингу через графічні інтерфейси, мобільні додатки або веб-панелі, формування керуючих сигналів та автоматичне сповіщення користувача про критичні події

Прикладний рівень визначає споживчу цінність системи. Наприклад, у системі моніторингу IoT-пристроїв цей рівень дозволяє не просто бачити поточні показники сенсорів, а й аналізувати ефективність роботи обладнання за певний період, прогнозувати можливі збої та оптимізувати ресурси.

Взаємодія між рівнями у трирівневій моделі є вертикальною та послідовною. Дані піднімаються від рівня сприйняття до прикладного, а команди керування спускаються у зворотному напрямку. Такий підхід забезпечує модульність системи: розробник може змінити тип датчиків на рівні сприйняття без необхідності повної перебудови прикладного програмного забезпечення. Це робить трирівневу модель оптимальною для проєктування масштабованих систем моніторингу, де кількість підключених пристроїв може динамічно змінюватися.

1.1.1 Концепція Edge та Fog computing

Класична модель побудови IoT-систем довгий час базувалася виключно на хмарних обчисленнях, де всі дані від сенсорів передавалися до віддаленого дата-центру. Однак стрімке збільшення кількості підключених пристроїв виявило ряд критичних проблем цієї архітектури: затримки передачі пакетів (latency), надмірне навантаження на канали зв’язку та повну залежність працездатності системи від наявності зовнішнього інтернет-з’єднання. Для вирішення цих питань у комп’ютерній інженерії було впроваджено парадигми

периферійних (Edge) та туманних (Fog) обчислень, які передбачають наближення обчислювальних потужностей безпосередньо до джерел даних. [6]

Концепція Edge Computing (периферійні обчислення) полягає у виконанні операцій обробки даних на кінцевих вузлах мережі, тобто на самих мікроконтролерах або інтелектуальних датчиках. У системі моніторингу це дозволяє реалізувати первинну фільтрацію сигналів, виявлення аномалій та стиснення інформації перед відправкою. Такий підхід суттєво зменшує обсяг трафіку, оскільки замість безперервного потоку сирих даних пристрій передає лише значущі зміни станів або агреговані показники за певний період. Окрім економії мережевих ресурсів, периферійні обчислення забезпечують миттєву реакцію на критичні події, оскільки логіка прийняття рішень спрацьовує локально, не очікуючи підтвердження від віддаленого сервера.

Fog Computing (туманні обчислення) розглядається як проміжний архітектурний рівень, що об’єднує ресурси локальної мережі [7]. Роль туманних вузлів зазвичай виконують локальні шлюзи (IoT Gateways) або сервери з вищою обчислювальною потужністю, ніж у датчиків. На цьому рівні здійснюється координація між декількома периферійними пристроями, локальне зберігання історії логів та глибша аналітика, яка є занадто ресурсомісткою для окремих мікроконтролерів. Використання цього рівня дозволяє побудувати відмовостійку систему, здатну зберігати працездатність та накопичувати дані навіть у разі обриву зв’язку з центральним сервером.

Інтеграція цих концепцій у систему моніторингу дозволяє збалансувати навантаження між апаратними компонентами. Роботу пристрою на базі Edge-обчислень можна описати як цикл «збір-фільтрація-реакція», тоді як Fog-вузли забезпечують стратегічне управління сегментом мережі. Такий розподіл функцій є особливо актуальним для автономних систем, де енергоефективність вузлів напряму залежить від частоти та тривалості роботи радіомодулів. Отже, перехід від централізованої хмарної моделі до розподіленої ієрархії Edge-Fog-Cloud є необхідною умовою для створення сучасних, надійних та масштабованих систем моніторингу IoT.

1.1.2 Взаємодія між компонентами системи

Концепція Machine-to-Machine (M2M) забезпечує прямий обмін даними між пристроями без обов’язкової участі людини [8]. У системі моніторингу IoT-пристроїв M2M-комунікації відповідають за те, як саме кінцеві вузли (сенсори) передають інформацію на шлюзи, а ті – у хмарні сервіси або інші виконавчі пристрої.

Взаємодія компонентів у таких мережах базується на трьох основних етапах:

1. Генерація та збір даних – периферійний пристрій зчитує показники фізичного світу.
2. Передача – використання дротових або бездротових каналів зв’язку для транспортування пакетів даних.
3. Обробка та реакція – отримання даних іншим пристроєм або сервером для виконання певного алгоритму (наприклад, увімкнення системи охолодження при перевищенні температури).

Для забезпечення ефективної M2M-взаємодії критичним є вибір моделі комунікації. Найбільш розповсюдженими є:

Запит-відповідь (Request-Response) – класична модель, де клієнт запитує дані, а сервер їх надає. Вона зручна для адміністрування, але створює додаткове навантаження на мережу через постійне опитування (polling).

Видавець-підписник (Publish-Subscribe) – найбільш адаптована модель для IoT. Пристрої відправляють дані брокеру, а всі зацікавлені компоненти отримують їх автоматично. Це дозволяє реалізувати асинхронний обмін та значно економити енергію кінцевих вузлів.

Особливістю взаємодії в сучасних системах моніторингу є необхідність забезпечення інтероперабельності. Це означає, що пристрої від різних виробників повинні "розуміти" один одного на рівні форматів даних. У роботі встановлено, що використання стандартизованих протоколів взаємодії дозволяє зменшити об’єм службової інформації в пакетах, що критично для каналів з низькою пропускною здатністю.

M2M-комунікації перетворюють розрізнені датчики на цілісну екосистему, де швидкість і надійність обміну даними безпосередньо впливають на точність моніторингу.

1.2 Типи IoT-пристроїв та їх особливості

Функціонування будь-якої сучасної системи моніторингу базується на отриманні та обробці первинної інформації про стан об’єкта або навколишнього середовища. В архітектурі Інтернету речей ці пристрої класифікуються за функціональним призначенням та роллю в ієрархії мережі. Базовий рівень утворюють сенсорні вузли (End devices), які як кінцеві пристрої здійснюють безпосередній збір даних, та виконавчих вузлів (актуаторів), що забезпечують зворотний фізичний вплив на середовище. Наступну ланку формують Edge-пристрої, призначені для локальної обробки й фільтрації отриманих сигналів з метою зменшення навантаження на комунікації, та шлюзи (Gateways), які виступають проміжними вузлами для агрегації даних і їх трансляції між різними типами мереж. Фінальним елементом цієї ієрархії є хмарні вузли, що представляють собою серверну інфраструктуру для довготривалого зберігання телеметрії та проведення глибокого ретроспективного аналізу всієї інформації.

Компонентами, що виконують завдання збору даних, є датчики (сенсори). Вони відіграють роль первинних перетворювачів, які трансформують фізичні параметри – такі як температура, вологість, атмосферний тиск або концентрація газів – у електричні сигнали. За принципом формування вихідного сигналу датчики поділяють на аналогові та цифрові.

Аналогові датчики генерують безперервний сигнал у вигляді напруги або струму, амплітуда якого пропорційна вимірюваній величині. Для їх інтеграції обов’язковим є використання аналого-цифрового перетворювача (АЦП). До цієї категорії належать термістори та фоторезистори. Головним

недоліком аналогових рішень є висока чутливість до електромагнітних перешкод, що вимагає апаратного фільтрування.

Цифрові датчики (наприклад, DHT22 для вологості або BMP280 для тиску) оснащені вбудованою логікою та передають дані у вигляді послідовності бітів за протоколами I2C, SPI, One-Wire. Використання таких компонентів дозволяє отримувати вже відкалібровані значення [4], мінімізуючи вплив зовнішніх шумів на результати моніторингу та спрощує розробку програмного забезпечення.

Важливою частиною систем є актуатори – виконавчі пристрої (реле, сервоприводи), що забезпечують зворотний зв’язок. Поєднання датчиків та актуаторів дозволяє реалізувати замкнений контур керування.

При виборі типу пристрою виникає ряд інженерних компромісів. Збільшення частоти передачі даних призводить до зростання енергоспоживання, а підвищення обчислювальної потужності вузла (для реалізації Edge computing) збільшує вартість кінцевого виробу. Наприклад, у системах моніторингу навколишнього середовища сенсорні вузли на базі ESP32 збирають дані про температуру та передають їх через Wi-Fi на шлюз, який виконує попередню обробку та трансляцію у хмару [2].

Платформи на базі 8-бітних AVR-мікроконтролерів Arduino є застарілими для таких задач через відсутність вбудованого зв’язку. Одноплатні комп’ютери Raspberry Pi мають надлишкове споживання для автономних вузлів. Найбільш збалансованим рішенням є ESP32, що підтримує режим глибокого сну.

Побудова оптимальної архітектури системи моніторингу вимагає використання цифрових сенсорів та енергоефективних мікроконтролерів із підтримкою сучасних бездротових протоколів, що дозволяє досягти необхідного рівня автономності та надійності передачі телеметрії.

Важливою особливістю IoT-пристроїв на базі мікроконтролерів є підхід до формування їхнього програмного забезпечення, де зазвичай виділяють два основні напрямки. Перший напрямок базується на класичному

низькорівневому програмуванні в середовищах типу Arduino IDE або ESP-IDF з використанням мов C та C++. Цей підхід забезпечує максимальну гнучкість і контроль над ресурсами, проте вимагає значних часових витрат на написання коду, керування мережевими стеками та обробки винятків.

Другий напрямок представляє сучасну концепцію декларативного конфігурування, прикладом якої є платформа з відкритим вихідним кодом ESPHome. Цей підхід передбачає опис апаратної структури пристрою у файлах конфігурації YAML замість класичного програмування. Платформа автоматично генерує оптимізовану та стабільну прошивку з підтримкою потрібних датчиків та протоколів. Використання таких систем суттєво прискорює проектування розподілених мереж, мінімізує людські помилки у коді та спрощує подальше адміністрування системи [9].

1.3 Аналіз протоколів прикладного рівня для обміну даними

Проєктування ефективної та надійної системи моніторингу потребує ретельного аналізу мережевих протоколів прикладного рівня. Оскільки кінцеві вузли (мікроконтролери) часто функціонують в умовах обмежених апаратних ресурсів та нестабільних каналів зв’язку, класичні підходи до передачі даних потребують адаптації.

Одним із базових стандартів для взаємодії є протокол MQTT (Message Queuing Telemetry Transport). Це мережевий протокол, що функціонує поверх стека TCP/IP і оптимізований для пристроїв із низькою обчислювальною потужністю та мереж із низькою пропускною здатністю. На відміну від традиційної синхронної моделі «клієнт-сервер», як у HTTP, MQTT використовує асинхронну архітектуру «видавець-підписник» (Publish/Subscribe). У такій системі кінцеві пристрої не встановлюють прямих з’єднань один з одним. Замість цього інформаційні потоки маршрутизуються через центральний керуючий вузол – брокер [10].

Архітектура протоколу базується на наступних ключових компонентах:

– Видавець (Publisher) – сенсорний вузол, який генерує телеметричні дані та ініціює їх відправку на сервер.

– Підписник (Subscriber) – програмний або апаратний клієнт, який реєструє свій інтерес до отримання даних за певними критеріями.

– Брокер (Broker) – серверне програмне забезпечення, що виконує функції отримання повідомлень від видавців, їх фільтрації та подальшої ретрансляції авторизованим підписникам.

– Тема (Topic) – ієрархічний текстовий ідентифікатор, виконує роль логічного каналу зв'язку або адреси для маршрутизації повідомлень.

Важливою характеристикою протоколу MQTT є підтримка рівнів якості обслуговування QoS (Quality of Service). Це механізм, який дозволяє налаштовувати баланс між надійністю доставки телеметрії та навантаженням на канали зв'язку за допомогою трьох рівнів специфікації.

На найнижчому рівні QoS 0 (At most once / «Найбільше один раз») повідомлення відправляється без підтвердження отримання, що мінімізує трафік, але не гарантує доставку в умовах нестабільного з'єднання. Рівень QoS 1 (At least once / «Щонайменше один раз») гарантує обов'язкову доставку повідомлення одержувачу завдяки використанню пакетів підтвердження, проте це може призвести до появи дублікатів даних при повторних відправках. Максимальну надійність забезпечує рівень QoS 2 (Exactly once / «Рівно один раз»), який гарантує доставку суворо одного примірника повідомлення без дублікатів за допомогою чотиристороннього рукостискання. У системах моніторингу вибір конкретного рівня QoS залежить від критичності даних: для регулярної поточної телеметрії зазвичай обирають QoS 0 або 1, тоді як для аварійних сповіщень – QoS 2 [10].

Незважаючи на ефективність MQTT на рівні сенсорних мереж, технологія REST API на базі HTTP є ключовим інструментом для інтеграції пристроїв з вебсервісами. Взаємодія в межах REST-архітектури реалізується через стандартні HTTP-запити до унікальних URI-адрес ресурсів [11].

Використання HTTP/HTTPS має значні переваги, серед яких сумісність з наявними вебтехнологіями та безперешкодне проходження через корпоративні мережеві екрани. Проте порівняно з MQTT, даний підхід має обмеження для бюджетних мікроконтролерів.

Для комплексного розуміння мережевого стека IoT у роботі також розглянуто альтернативні протоколи [12]. Протокол CoAP (Constrained Application Protocol) розроблений для пристроїв із критичними обмеженнями ресурсів, функціонує поверх UDP і є оптимальним для мереж із високим рівнем втрати пакетів. AMQP забезпечує корпоративний рівень черг повідомлень та високу надійність, проте є ресурсоємним для кінцевих мікроконтролерів. WebSocket забезпечує двосторонній зв’язок поверх єдиного TCP-з’єднання, що робить його ефективним вибором для веб-панелей моніторингу реального часу.

Важливим аспектом проєктування є забезпечення безпеки передачі даних. Більшість сучасних протоколів підтримують шифрування за допомогою TLS/SSL, а також механізми автентифікації клієнтів, що мінімізує ризики перехоплення телеметрії у відкритих мережах.

У типовій IoT-системі сенсорні вузли передають дані на проміжний сервер або брокер, який виконує маршрутизацію повідомлень до споживачів (бази даних, аналітичні сервіси). Для взаємодії з веб-інтерфейсом можуть використовуватися HTTP або WebSocket залежно від вимог до затримки та режиму роботи.

Можна виділити основні критерії вибору протоколу для IoT-систем:

– енергоефективність (мінімізація обсягу переданих даних);
– затримка передачі (latency);
– надійність доставки (QoS);
– складність реалізації на мікроконтролерах;
– вимоги до пропускної здатності мережі;
– підтримка асинхронної взаємодії.

На основі аналізу міжнародних інженерних стандартів та порівняльних досліджень архітектури IoT-мереж, наведених у науковій праці [13], було сформовано порівняльну характеристику ключових протоколів прикладного рівня. Об’єктивні технічні параметри, що впливають на вибір технологічного стека системи, зведено в таблиці 1.2.

Таблиця 1.2 – Технічні характеристики протоколів прикладного рівня мережі IoT

Параметр протоколу	MQTT	HTTP (REST)	CoAP	WebSocket
Транспортний рівень (L4)	TCP	TCP	UDP	TCP
Мінімальний розмір заголовка	2 байти	~200–800 байт	4 байти	2–14 байт
Формат кодування пакетів	Бінарний	Текстовий (ASCII)	Бінарний	Бінарний / Текстовий
Гарантія доставки (QoS)	3 рівні (QoS 0, 1, 2)	Модель «запит-відповідь»	2 типи повідомлень	Відсутня

1.4 Основні проблеми IoT-систем

Основні проблеми IoT-систем можна поділити на кілька груп: безпека, масштабованість і сумісність, мережеві обмеження, енергоефективність та надійність функціонування. Вони мають як апаратний, так і програмний характер, безпосередньо впливаючи на стабільність та економічну доцільність впровадження системи.

Питання безпеки є дуже важливим аспектом в IoT. Більшість кінцевих пристроїв мають обмежені обчислювальні ресурси, що ускладнює використання алгоритмів асиметричного шифрування. До основних загроз цілісності та конфіденційності даних у системах моніторингу належать перехоплення незашифрованого трафіку зловмисником у межах локальної або глобальної мережі, а також підміна вузла (Spoofing), що створює можливість підключення до системи несанкціонованого пристрою, який транслює хибну телеметрію від імені легітимного датчика. Окрему небезпеку становлять атаки

типу «відмова в обслуговуванні» (DoS), які полягають у навмисному перевантаженні мережевого шлюзу або брокера повідомлень великою кількістю запитів для повного блокування працездатності системи моніторингу.

Для мінімізації зазначених загроз у сучасних системах застосовуються методи шифрування даних (TLS/SSL), дворівнева автентифікація клієнтів, контроль доступу на рівні брокера (ACL) та використання ізольованих каналів зв’язку (VLAN або VPN).

Також важливою проблемою є забезпечення надійності функціонування. У реальних умовах IoT-вузли можуть втрачати з’єднання через нестабільність бездротового покриття або апаратні збої. Це призводить до втрати даних або порушення безперервності моніторингу об’єкта. Для підвищення відмовостійкості доцільно впроваджувати механізми автоматичної повторної передачі повідомлень, локальної буферизації даних на стороні вузла та використання протоколів із обов’язковим підтвердженням доставки на прикладному рівні.

Проблема масштабованості та сумісності виникає при інтеграції обладнання різних виробників, що використовують закриті пропрієтарні протоколи. При збільшенні кількості вузлів з десятків до тисяч спостерігається лінійне зростання кількості одночасних з’єднань та обсягу трафіку, яке перевантажує центральний сервер. Для вирішення цієї проблеми архітектура системи має передбачати перехід на відкриті стандарти взаємодії, впровадження методів балансування навантаження, кластеризацію серверних потужностей та децентралізацію обробки даних на проміжних рівнях мережі.

При розгортанні територіально розподіленої мережі необхідно забезпечити можливість дистанційного оновлення вбудованого програмного забезпечення (Over-the-Air, OTA), моніторинг апаратного стану вузлів та їх централізовану конфігурацію. Відсутність таких механізмів ускладнює обслуговування системи при її розширенні.

У бездротових мережах передачі даних існує жорстка залежність між дальністю зв’язку, швидкістю трансляції та енерговитратами. Постійна підтримка активного радіозв’язку з метою мінімізації затримок (latency) призводить до швидкого розряду акумуляторів автономних вузлів. Загальне енергоспоживання пристрою напряму пропорційне потужності радіомодуля та часу його перебування в активному стані. Це зумовлює інженерну необхідність скорочення тривалості сеансів зв’язку та переведення мікроконтролера в режими глибокого сну для збереження ресурсу батареї. Таким чином, при проєктуванні сучасних IoT-систем необхідно враховувати складний компроміс між затримкою передачі даних, рівнем безпеки, енергоспоживанням та загальною стабільністю функціонування мережі.

У першому розділі проведено комплексний теоретичний аналіз архітектурних принципів та комунікаційних моделей сучасних систем Інтернету речей. Обґрунтовано доцільність використання багаторівневої структури мережі з розподіленими обчисленнями для мінімізації затримок, зниження навантаження на канали зв’язку та забезпечення енергоефективності й безпеки передачі телеметрії. Отримані результати дозволяють сформувати вихідні технічні вимоги до компонентів системи моніторингу, конкретний вибір та обґрунтування яких буде здійснено у наступному розділі роботи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ

2.1 Аналіз існуючих систем

Розвиток індустрії Інтернету речей призвів до появи значної кількості платформ, що дозволяють збирати та візуалізувати дані з датчиків за допомогою готових інструментів. Для розробки ефективної системи моніторингу необхідно проаналізувати переваги та недоліки найбільш розповсюджених хмарних рішень.

Одним із найвідоміших сервісів є Blynk. Дана платформа орієнтована на швидке створення мобільних застосунків для керування мікроконтролерами. Основними перевагами Blynk є простий інтерфейс для створення дашбордів та широка підтримка різних типів зв’язку (Wi-Fi, Bluetooth, Ethernet). Проте для професійних або розгалужених систем моніторингу Blynk має суттєві недоліки: закрита екосистема, обмеження безкоштовного плану та відсутність гнучких інструментів для глибокого аналізу історичних даних.

Платформа ThingsBoard представляє собою більш потужне рішення корпоративного рівня. Вона підтримує протоколи MQTT, CoAP та HTTP і дозволяє створювати складні сценарії обробки даних за допомогою вбудованого Rule Engine. ThingsBoard забезпечує високу масштабованість, але є ресурсомісткою. Розгортання власного сервера потребує значних потужностей, а використання хмарної версії є дороговартісним для некомерційних або освітніх проєктів.

Adafruit IO є популярним вибором серед розробників прототипів завдяки простоті використання та вичерпної документації. Платформа фокусується на принципі «feeds» (потоки даних), що полегшує роботу з часовими рядами. Однак обмеження безкоштовного тарифу (наприклад, не більше 30 точок даних на хвилину) роблять її непридатною для систем, де потрібен моніторинг параметрів у реальному часі з високою частотою оновлення.

Альтернативою закритим хмарним сервісам є використання платформ із відкритим вихідним кодом (Open-source). Такий підхід дозволяє розгорнути

серверну частину на власних потужностях розробника, наприклад, на персональному комп’ютері, VPS-сервері або одноплатному комп’ютері типу Raspberry Pi. Це забезпечує повний контроль над даними та відсутність залежності від платної підписки сторонніх компаній.

Серед найбільш розповсюджених систем з відкритим кодом для задач моніторингу виділяють Home Assistant, OpenHAB та інструмент Node-RED.

Home Assistant – це популярна платформа для автоматизації. Вона підтримує велику кількість пристроїв і протокол MQTT. Головною перевагою є можливість роботи у локальній мережі без доступу до інтернету. Проте для невеликої системи моніторингу ця платформа може бути занадто складною в налаштуванні, оскільки вона перевантажена функціями для «розумного будинку», які не є обов’язковими для збору технічної телеметрії.

OpenHAB має схожі можливості, але побудований на мові програмування Java. Платформа пропонує гнучкі налаштування через конфігураційні файли. Основним недоліком OpenHAB є високий поріг входження для розробника-початківця та складний процес створення графічного інтерфейсу, що потребує багато часу.

Node-RED використовує принцип візуального програмування потоків даних. Користувач створює логіку роботи системи, з’єднуючи готові вузли (nodes) між собою. Node-RED працює на базі Node.js і споживає мінімум системних ресурсів. Це ідеальний інструмент для обробки MQTT-повідомлень завдяки своїй гнучкості. Проте для побудови професійних графіків та тривалого зберігання історії даних Node-RED зазвичай потребує інтеграції з зовнішніми базами даних та спеціалізованими панелями візуалізації, такими як Grafana.

Узагальнені технічні характеристики та порівняльний аналіз архітектурних обмежень досліджуваних платформ на основі аналізу їхніх специфікацій [12] зведено в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика систем моніторингу IoT

Платформа	Тип архітектури	Безкоштовний ліміт даних	Локальне збереження даних	Мін. вимоги до RAM сервера	Ключове інженерне обмеження
Blynk	Хмарна (Cloud)	До 3 пристроїв / 10 датчиків	Ні (Хмара Blynk)	Не застосовується (SaaS)	Жорстка прив’язка до пропрієтарного мобільного додатка
ThingsBoard	Гібридна (On-premise / Cloud)	Безлімітно (Community Edition)	Так (PostgreSQL / Cassandra)	1–2 Гб	Висока обчислювальна складність для мікрокомп’ютерів
Adafruit IO	Хмарна (Cloud)	30 повідомлень / хв (до 30 днів)	Ні (Хмара Adafruit)	Не застосовується (SaaS)	Обмеження на об’єм ретроспективних даних (30 днів)
Home Assistant	Локальна (On-premise)	Безлімітно (Open Source)	Так (SQLite / MariaDB)	1–2 Гб	Архітектурна орієнтація на побутову автоматизацію (Smart Home)
OpenHAB	Локальна (On-premise)	Безлімітно (Open Source)	Так (вбудована СУБД / rrd4j)	1 Гб (через Java JVM)	Високе споживання ресурсів RAM та складність конфігурації
Node-RED	Локальна (On-premise)	Безлімітно (Open Source)	Потребує сторонньої БД	256–512 Мб	Відсутність вбудованої СУБД для збереження телеметрії

Аналіз показує, що для реалізації системи моніторингу найкраще підходить модульний підхід. Він передбачає поєднання легкого брокера повідомлень для обміну даними та окремого інструменту для професійної візуалізації результатів. Базовим напрямком для практичної реалізації системи обрано моніторинг параметрів мікроклімату. За даними сучасних досліджень [6], контроль температури та вологості за допомогою IoT-сенсорів є одним із найпопулярніших рішень для технічних приміщень, складів та серверних кімнат, оскільки ці показники є критичними для стабільної роботи комп’ютерного та виробничого обладнання.

При цьому розроблена архітектура системи є універсальною. Поточна практична конфігурація налаштована суто під збір кліматичних метрик, але сама побудова мережі та серверної частини дозволяє адаптувати комплекс під

будь-які інші задачі Інтернету речей (наприклад, моніторинг вібрації чи тиску). Для переорієнтації системи на новий напрямок достатньо змінити склад сенсорів на клієнтському вузлі та налаштування віджетів у Grafana, тоді як базове ядро інфраструктури та мережева топологія залишаються незмінними

2.2 Вибір технологій для реалізації системи

Ефективність системи моніторингу значною мірою залежить від правильного вибору апаратної платформи. Ключовими критеріями для вибору мікроконтролера в задачах IoT є наявність вбудованих мережевих інтерфейсів, обчислювальна потужність, енергоспоживання та обсяг оперативної пам’яті.

Для реалізації даної системи було розглянуто три найбільш розповсюджені платформи: Arduino Uno, STM32 та ESP32.

Arduino (Uno/Nano) є класичним рішенням для навчання, проте вона має суттєві обмеження для сучасних систем моніторингу. Головним недоліком є відсутність вбудованих інтерфейсів Wi-Fi або Ethernet, що потребує використання додаткових модулів (наприклад, ESP8266 або W5500). Крім того, малий обсяг оперативної пам’яті (всього 2 КБ) ускладнює реалізацію надійних протоколів шифрування даних.

STM32 – це потужні 32-бітні контролери з широким набором периферії. Вони демонструють високу швидкість роботи та низьке енергоспоживання. Проте більшість моделей також не мають інтегрованого Wi-Fi стеку, що ускладнює конструкцію пристрою та підвищує його вартість для простих вузлів моніторингу.

ESP32 від компанії Espressif Systems є найбільш збалансованим рішенням. Цей контролер має двоядерний процесор із частотою до 240 МГц та, найголовніше, вбудовані модулі Wi-Fi та Bluetooth. Наявність 520 КБ оперативної пам’яті дозволяє без проблем використовувати стек протоколів TCP/IP та бібліотеки для роботи з MQTT. Також ESP32 підтримує режими глибокого сну (deep sleep), що критично для автономних датчиків.

Порівняльні характеристики платформ наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння апаратних платформ для IoT-вузла

Характеристика	Arduino Uno	STM32F103C8	ESP32
Процесор	8-біт	32-біт	32-біт
Частота	16 МГц	72 МГц	240 МГц
RAM	2 КБ	20 КБ	520 КБ
Wi-Fi / BT	Відсутні	Відсутні	Вбудовані
Ціна/Ефективність	Низька	Середня	Висока

Також необхідно визначити відповідне програмне забезпечення для розгортання внутрішньої логіки пристрою та обробки даних на серверній стороні. Для реалізації програмного забезпечення клієнтського вузла на базі ESP32, замість традиційного написання низькорівневого коду в середовищі Arduino IDE або використання інтерпретованої мови MicroPython, у даному проєкті обрано систему ESPHome.

На основі проведеного в першому розділі аналізу, саме декларативне конфігурування дозволяє повністю відмовитися від ручного програмування мережевих сокетів, забезпечуючи стабільну роботу Wi-Fi стека та швидку інтеграцію з MQTT-інфраструктурою через файли конфігурації YAML.

Для збереження історії показників (температури, вологості тощо) доцільно використовувати базу даних часових рядів (Time Series Database), таку як InfluxDB, оскільки вона має перевагу над звичайними базами даних у швидкості запису та зчитування даних, які прив’язані до конкретної мітки часу.

Щодо серверної частини, для збору та проміжної обробки даних було обрано агента збору метрик Telegraf з відкритим вихідним кодом. Вибір зумовлений наявністю вбудованої комунікації з MQTT-брокером та автоматичної трансляції отриманих метрик телеметрії до бази даних InfluxDB.

Отже, програмний стек проектованої системи базуватиметься на декларативних інструментах ESPHome для клієнтського рівня та сервісах Telegraf й InfluxDB для серверної обробки інформації.

Ключовим елементом архітектури IoT-системи є брокер повідомлень, який виступає посередником між пристроями та сервером. Для реалізації

обміну даними за протоколом MQTT було розглянуто кілька рішень: Mosquitto, EMQX та HiveMQ.

Брокер Mosquitto є найбільш розповсюдженим рішенням з відкритим кодом. Він вирізняється надзвичайною легкістю, низькими вимогами до системних ресурсів та повною підтримкою стандартів MQTT v3.1/v5.0. Завдяки своїй компактності, Mosquitto ідеально підходить для розгортання на невеликих серверах або локальних машинах розробника.

EMQX та HiveMQ є потужними брокерами промислового рівня, що підтримують мільйони одночасних підключень та мають вбудовані графічні інтерфейси для моніторингу черг. Проте їхня архітектура є надто складною для системи моніторингу обмеженої кількості пристроїв, а споживання оперативної пам’яті значно перевищує показники Mosquitto. Тому для даного проєкту було обрано Eclipse Mosquitto.

Для візуалізації отриманої телеметрії було проведено аналіз між використанням Node-RED Dashboard та Grafana.

Node-RED Dashboard дозволяє швидко створити простий інтерфейс користувача, але має обмежені можливості для побудови складної аналітики та роботи з історичними даними.

Grafana є професійним інструментом для аналізу та візуалізації метрик. Вона підтримує інтеграцію з більшістю сучасних баз даних, дозволяє створювати динамічні дашборди з гнучкими налаштуваннями графіків, таблиць та систем сповіщень (Alerting). Використання Grafana забезпечує високу якість представлення даних для кінцевого користувача та зручність аналізу ефективності системи.

2.2.1 Обґрунтування ефективності передачі даних

Ключовим інженерним критерієм при виборі фінального протоколу взаємодії (MQTT) є мінімізація об’єму службової інформації в бездротовому каналі зв’язку. Для кількісної оцінки ефективності передачі даних

використовується відома в телекомунікаціях метрика накладних витрат (Overhead), яка розраховується за формулою:

$$Overhead = \frac{D_{header}}{D_{payload} + D_{header}} * 100\%, \quad (2.1)$$

де D_{header} – об'єм службових заголовків пакета (Байт), а $D_{payload}$ – розмір корисного навантаження телеметрії (Байт).

Для практичної оцінки розглянемо випадок передачі типового пакета телеметрії (значення температури та вологості об'єкта у форматі JSON), об'єм якого становить $D_{payload} = 24$ Байти.

При використанні базового запиту HTTP POST мінімальний розмір заголовків становить близько $D_{header_{HTTP}} = 200$ Байт Проведемо розрахунок накладних витрат за формулою (2.1):

$$Overhead_{HTTP} = \frac{200}{24 + 200} * 100\% = \frac{200}{224} * 100\% \approx 89.3\%$$

При використанні обраного протоколу MQTT фіксований заголовок пакета PUBLISH становить лише $D_{header\_MQTT} = 2$ Байти. Розрахунок накладних витрат для цього випадку матиме вигляд:

$$Overhead_{MQTT} = \frac{2}{24 + 2} * 100\% = \frac{2}{26} * 100\% \approx 7.7\%$$

Порівняльний аналіз результатів показує, що при використанні HTTP накладні витрати становлять 89.3%, тобто майже весь трафік витрачається на службову інформацію. Натомість протокол MQTT демонструє показник витрат лише 7.7%, що дозволяє кардинально знизити навантаження на мережу, зменшити час роботи радіомодуля в активному режимі та забезпечити високу енергоефективність автономного сенсорного вузла.

Отже, остаточний технологічний стек системи включає: мікроконтролер ESP32, протокол MQTT (брокер Mosquitto), агент збору метрик Telegraf, базу даних InfluxDB та платформу візуалізації Grafana.

2.3 Розробка структурної та функціональної схем системи

На основі проведеного аналізу архітектурних рішень у сфері Інтернету речей (IoT) було спроєктовано та деталізовано структурну схему комп'ютерної системи моніторингу мікроклімату, яка базується на класичній трирівневій архітектурній моделі (Perception, Network, Application Layers). Запропонована архітектурна модель базується на чіткому розділенні функціональних обов'язків між апаратними модулями, транспортною системою та серверною інфраструктурою. Такий підхід забезпечує високий рівень масштабованості комп'ютерної системи, гнучкість при інтеграції нових вузлів та відмовостійкість інфраструктури в цілому.

Для забезпечення деталізації практичної реалізації системи, на схемі базовий прикладний рівень (Application Layer) розділено на два функціональні блоки: рівень обробки/збереження даних та інтерфейс оператора.

Загальну логіку взаємодії та ієрархічну побудову розробленої системи представлено на структурній схемі (рис. 2.1).

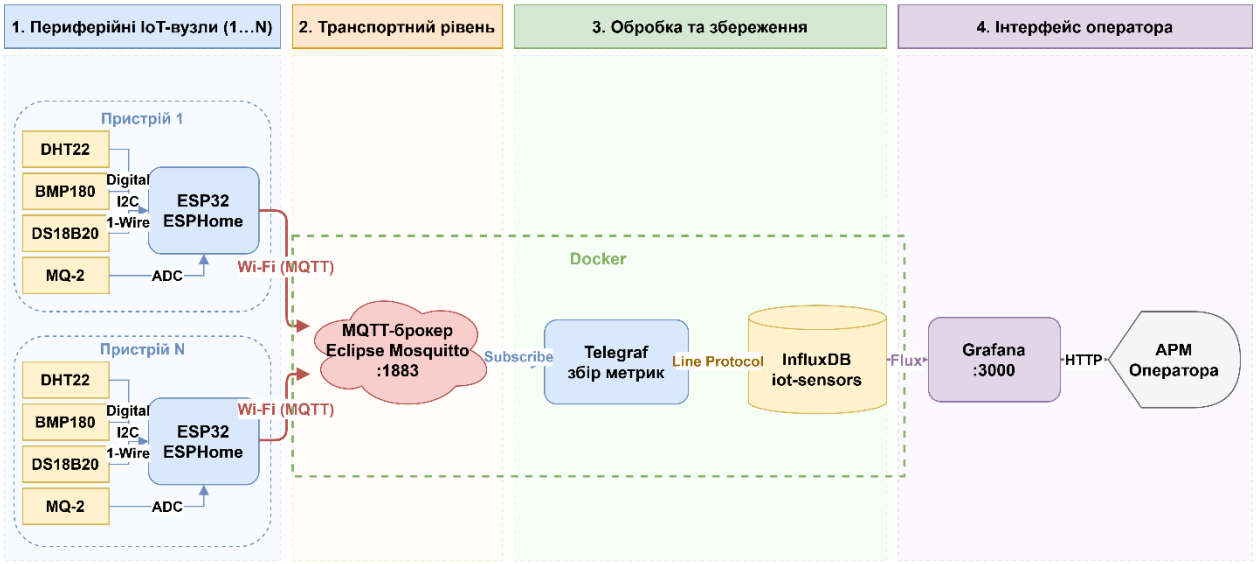

Рисунок 2.1 – Структурна схема комп'ютерної системи моніторингу мікроклімату

Відповідно до розробленої схеми, комп'ютерна система функціонує на базі чотирьох основних технологічних рівнів:

1. Периферійні IoT-вузли (Perception Layer) – апаратна основа системи, що взаємодіє з фізичним середовищем об’єкта моніторингу. Він складається з двох функціональних підрівнів:

– Сенсорний підрівень – містить датчики для реєстрації параметрів навколишнього середовища. До них відносяться: цифровий датчик температури та вологості DHT22 (інтерфейс GPIO), датчик атмосферного тиску та температури BMP180 (шина I2C), прецизійний напівпровідниковий датчик температури DS18B20 (протокол One-Wire) та аналого-цифровий датчик концентрації газів і диму MQ-2 (підключений до ADC). Така конфігурація демонструє універсальність системи щодо підтримки різних фізичних інтерфейсів збору даних.

– Підрівень мікроконтролера – базується на платах ESP32 з прошивкою ESPHome, що функціонують паралельно у різних приміщеннях об’єкта. Архітектура системи передбачає можливість незалежного підключення N-кількості таких пристроїв до єдиного центрального сервера. Кожен мікроконтролер здійснює циклічне опитування власної підключеної периферії. Також передбачене автоматичне відправлення діагностичних міток стану самого заліза, зокрема рівня сигналу бездротової мережі RSSI та часу безперервної роботи пристрою Uptime. Програмний MQTT-клієнт відповідає за нормалізацію отриманих метрик, приведення їх до числового типу та ініціалізацію сесії передачі повідомлень до спільного брокера.

2. Транспортний рівень (Network Layer) – виконує роль середовища доставки телеметричної інформації від периферійних пристроїв досерверної частини. Взаємодія компонентів організована в межах локальної комп’ютерної мережі (LAN) за допомогою стандартного стеку TCP/IP.

Прикладним протоколом передачі даних обрано легковаговий асинхронний протокол MQTT, який функціонує через виділений системний порт 1883 (TCP). Використання операції публікації (publish) дозволяє IoT-вузлу відправляти пакети даних миттєво по мірі їх зчитування, мінімізуючи час утримання мережевого підключення та обсяг службового трафіку.

3. Рівень обробки та збереження даних (Application Layer) – представляє собою серверну інфраструктуру, розгорнуту за допомогою контейнеризації на базі Docker з ОС Linux. Використання ізольованих контейнерів дозволяє незалежно адмініструвати, конфігурувати та масштабувати кожен програмний сервіс:

– MQTT-брокер (Контейнер: mqtt-broker) – функціонує на базі Eclipse Mosquitto 2.x, виступає центральним комутаційним вузлом, який приймає вхідні TCP-пакети від клієнтів.

– Агент збору метрик (Контейнер: telegraf) – утиліта Telegraf 1.30 налаштована в режимі постійної підписки (subscribe) на відповідні MQTT-топіки брокера. Агент виконує роль шлюзу: перехоплює повідомлення, здійснює синтаксичний розбір структури топіків для ідентифікації метрик та виконує трансляцію числових даних (операція write) до бази даних.

– База даних часових рядів (Контейнер: influxdb) – СУБД InfluxDB 2.7 (TSDB), призначена для збереження потокових метрик із прив’язкою до часових міток, що забезпечує швидку обробку запитів та ефективне збереження історії вимірювань.

4. Рівень інтерфейсу оператора (Application Layer) – верхній прикладний рівень взаємодії користувача з системою. Він базується на аналітичній платформі Grafana, яка через внутрішню Docker-мережу надсилає Flux-запити до InfluxDB і динамічно будує інтерактивні графіки. Кінцевий користувач (оператор) отримує доступ до графічного дашборду через веб-браузер за протоколом HTTP (порт 3000), що виключає необхідність встановлення додаткового клієнтського програмного забезпечення.

Для деталізації внутрішніх процесів перетворення, обробки, десеріалізації та передачі інформаційних сигналів у межах визначених технологічних рівнів було розроблено функціональну схему комп’ютерної системи (рис. 2.2). Якщо структурна модель відображає архітектурний склад системи, то функціональна схема демонструє динаміку проходження та

трансформації даних від моменту реєстрації фізичної величини до її фінального відображення на моніторі оператора.

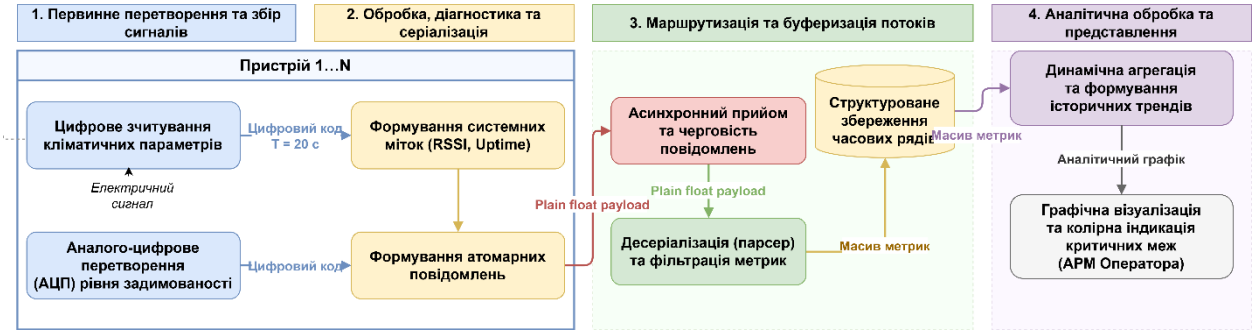

Рисунок 2.2 – Функціональна схема процесів перетворення даних

Загальну логіку функціональних перетворень у системі розподілено за такими послідовними етапами:

1. Первинне перетворення та збір сигналів – реєстрація фізичних параметрів середовища чутливими елементами датчиків. Для цифрових сенсорів (DHT22, BMP180, DS18B20) трансляція даних здійснюється у вигляді готового цифрового коду через відповідні інтерфейси (I2C, One-Wire, GPIO). Для аналогового датчика газу MQ-2 електричний сигнал напруги транслюється на інтегрований АЦП мікроконтролера, де конвертується в цифровий вигляд.

2. Обробка, діагностика та серіалізація – виконання локальних обчислювальних та аналітичних операцій на базі обчислювального модуля ESP32 під керуванням операційного середовища ESPHome. Система здійснює генерацію телеметричних та діагностичних міток стану самого пристрою (рівень сигналу бездротової мережі RSSI та час безперервної роботи пристрою Uptime). Отримані цифрові коди калібруються, переводяться у змінні з плаваючою комою (тип даних float) та розподіляються на окремі, незалежні повідомлення для кожного типу вимірювання. Сформоване корисне навантаження передається мережевому клієнту.

3. Маршрутизація та буферизація потоків – цей етап відповідає за транспортування, черговість та проміжне структурування даних на стороні серверної інфраструктури Docker. Серверний брокер Mosquitto забезпечує

асинхронний прийом та розподіл повідомлень за топіками. Далі системний агент Telegraf перехоплює ці дані, здійснює автоматичний розбір структури самого MQTT-топіка для фільтрації метрик і виділення назви приміщення, після чого трансформує чисті числові значення у структурований масив для передачі в базу даних InfluxDB, де вони зберігаються разом із часовими мітками (timestamps).

4. Аналітична обробка та представлення - верхній прикладний рівень взаємодії кінцевого користувача (оператора) з системою на базі платформи Grafana. Аналітичне ядро періодично робить вибірку історичних масивів метрик із бази даних, надсилаючи Flux-запити, здійснює динамічну агрегацію та формування історичних трендів. На автоматизованому робочому місці (АРМ) оператора виконується графічна візуалізація поточних параметрів, побудова часових залежностей та колірна індикація критичних меж для оперативного контролю стану.

2.4 Проєктування мережевої топології та функціональної структури інформаційного обміну

Для забезпечення надійної, безпечної та відмовостійкої взаємодії між усіма апаратними та програмними компонентами комп’ютерної системи було спроєктовано мережеву топологію інфраструктури збору телеметрії. За базову топологію мережі на концептуальному рівні обрано схему типу «Зірка» (рис. 2.3), де центральним комутаційним вузлом і координатором інформаційних потоків виступає брокер повідомлень, а периферійні пристрої та прикладні сервіси функціонують як клієнти.

Загальну схему мережевої топології розробленої системи з урахуванням конфігурації сокетів та віртуальних шлюзів представлено на рисунку 2.3.

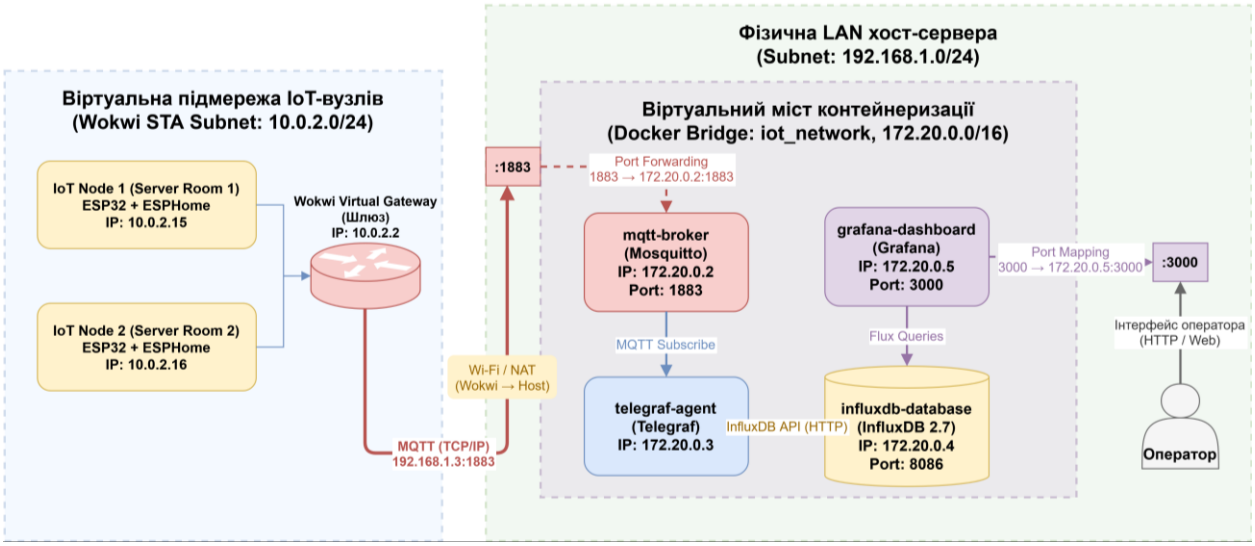

Рисунок 2.3 – Схема мережевої топології

Мережева архітектура проєкту розділена на два основні сегменти: клієнтську сторону та серверну сторону.

1. Клієнтський сегмент мережі – складається з двох IoT-пристроїв, розгорнутих у різних технологічних зонах (Server Room 1 та Server Room 2). Під час моделювання вони функціонують всередині ізольованого симулятора, де кожному вузлу динамічно виділяється власна внутрішня IP-адреса (10.0.2.15 для першого вузла та 10.0.2.16 для другого вузла). Зв’язок із зовнішнім світом та локальною мережею хост-комп’ютера здійснюється через спеціалізований віртуальний комутатор – Wokwi Gateway. На рівні прошивок обох мікроконтролерів маршрутом за замовчуванням визначено IP-адресу шлюзу 10.0.2.2. Усі вихідні TCP-пакети від обох кімнат передаються через Wi-Fi шлюз і за допомогою технології NAT транслюються у фізичну мережу обчислювального сервера. Після цього MQTT-брокер приймає ці дані та розподіляє їх за відповідними топіками.

2. Серверна частина системи розгорнута у віртуальній підмережі Docker типу bridge (міст). Це забезпечує ізоляцію сервісів та їхню безпечну комунікацію через внутрішні DNS-імена без потреби відкриття всіх портів у зовнішню мережу. Взаємодія та логіка адресації сокетів (IP:Port) на серверному рівні спроєктована за такою схемою:

Сокет MQTT-брокера – контейнер Eclipse Mosquitto прокидає (експортує) свій внутрішній порт назовні, роблячи його доступним для шлюзу Wokwi через сокет 10.0.2.2:1883. IoT-пристрій встановлює TCP-з'єднання саме з цим портом для публікації пакетів телеметрії.

Внутрішній сокет збору метрик: Контейнер Telegraf підключається до брокера всередині Docker-мережі (порт 1883), трансформує отримані пакети і надсилає їх у базу даних на сокет influxdb:8086. Для зовнішньої мережі порт 8086 залишається закритим, що мінімізує ризики несанкціонованого доступу.

Сокет підсистеми візуалізації – контейнер Grafana звертається до контейнера influxdb:8086 для виконання аналітичних запитів. Для того, щоб кінцевий користувач або системний адміністратор міг здійснювати візуальний моніторинг, внутрішній порт Grafana прокинуто на фізичний інтерфейс хост-комп'ютера. Таким чином, доступ до графічного інтерфейсу керування здійснюється через сокет localhost:3000 (або 127.0.0.1:3000) за протоколом HTTP.

Запропонована топологія мережі забезпечує сувору ізоляцію бази даних, ефективний розподіл трафіку через брокер повідомлень та гнучкість налаштування мережевих екранів (firewalls), оскільки для роботи всієї системи назовні відкритими залишаються лише два порти: 1883 – для прийому даних від IoT-пристроїв та 3000 – для веб-інтерфейсу моніторингу оператора.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Конфігурація IoT-вузлів у середовищі віртуалізації

Практична реалізація системи розпочинається з етапу конфігурації мережевих параметрів периферійних IoT-вузлів та верифікації їхньої взаємодії з транспортною інфраструктурою. Оскільки розгортання великої кількості фізичних пристроїв на етапі створення прототипу є економічно недоцільним, для моделювання та інженерної перевірки мережевого стеку було використано середовище Visual Studio Code з інтегрованим модулем апаратної віртуалізації Wokwi. Робота на цьому етапі полягала у створенні уніфікованого декларативного профілю для платформи ESPHome, налаштуванні низькорівневих апаратних інтерфейсів мікроконтролера ESP32, ініціалізації бездротового модуля зв’язку та заданні параметрів адресації сокетів для передачі даних за протоколом MQTT.

Програмно-апаратне налаштування клієнтських вузлів виконувалось шляхом формування конфігураційного файлу у форматі YAML, повний текст якого представлено в Додатку А, що дозволило повністю виключити необхідність написання коду та рутинної обробки помилок. На першому етапі конфігурувалися параметри бездротової мережі, де цільова точка доступу вказувалася як віртуальна мережа симулятора Wokwi-GUEST. Для забезпечення транспортування даних на сервер у блоці налаштувань MQTT-клієнта було визначено мережевий сокет брокера Mosquitto з локальною IP-адресою 192.168.1.3 та виділеним стандартним портом 1883. Завдяки вбудованим механізмам маршрутизації Wokwi Gateway, віртуальний пристрій успішно транслює пакети через NAT-шлюз безпосередньо до розгорнутого локального сервера інфраструктури.

Наступний етап конфігурування охоплював декларативний опис підключеної периферії та визначення режимів збору даних. Замість використання стандартних блокуючих затримок процесора, в ESPHome було застосовано автоматичне планування інтервалів опитування через параметр

update_interval, що дорівнює 20 секундам для основних сенсорів. Прошивка на основі конфігурації самостійно ініціалізує цифрову шину GPIO для датчика мікроклімату DHT22, інтерфейс I2C для датчика атмосферного тиску BMP180, протокол One-Wire для термометра DS18B20 та аналоговий вхід ADC для датчика газу MQ-2. Внутрішнє ядро системи самостійно розподіляє обчислювальні потоки, що гарантує стабільну роботу мережевого стеку та унеможливлює випадкові розриви TCP-сесії з брокером повідомлень.

Важливим елементом конфігурації, що забезпечує виконання завдань самодіагностики системи, стало інтегрування вбудованих компонентів моніторингу апаратного стану самого вузла. До загального профілю пристрою було додано системні платформи wifi_signal та uptime. Перша дозволяє безперервно зчитувати та транслювати рівень сигналу Wi-Fi мережі (RSSI) у децибелах, що необхідно для вимірювання якості покриття. Друга платформа фіксує час безперервної роботи мікроконтролера з моменту його останнього запуску. Ці діагностичні параметри автоматично агрегуються системою та збираються разом із кліматичними показниками в єдине структуроване повідомлення. Пристрої публікують сформовані дані у відповідні структуровані MQTT-топіки брокера із префіксами room1/ та room2/ відповідно до ідентифікаторів кімнат, що забезпечує чітку ізоляцію та зручну маршрутизацію телеметрії на серверній стороні.

Для верифікації коректності спроєктованої архітектури, перевірки працездатності логіки та цілісності зв’язку було запущено сесію віртуалізації. Загальний вигляд інтеграції розробленого IoT-вузла у середовищі симулятора Wokwi представлено на рисунку 3.1.

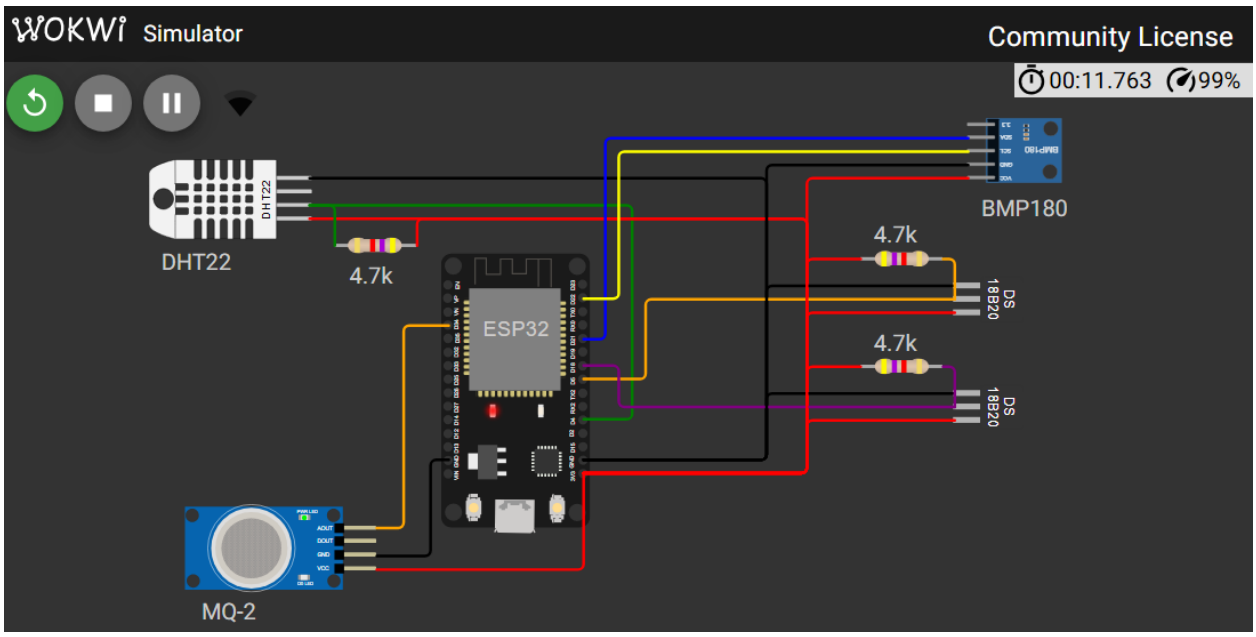

Рисунок 3.1 – Моделювання та верифікація роботи IoT-вузла у середовищі VS Code / Wokwi

3.2 Контейнеризація та розгортання серверної інфраструктури в середовищі Docker

Наступним етапом реалізації комп'ютерної системи є розгортання та конфігурація серверної інфраструктури, призначеної для прийому, маршрутизації та збереження потоків телеметричної інформації. Локальне встановлення кожного системного сервісу безпосередньо в операційну систему хост-машини є неефективним з інженерного погляду через конфлікти залежностей, складність масштабування та низьку ізоляцію процесів.

Для вирішення цих системних задач у роботі застосовано технологію контейнеризації на базі інструменту Docker Compose. Це дозволяє реалізувати концепцію інфраструктури як коду (Infrastructure as Code), описати весь серверний стек у єдиному маніфесті, ізолювати програмні компоненти на рівні просторів імен (namespaces) та автоматизувати їхню інтеграцію у віртуальну мережу типу bridge.

Для автоматизації розгортання та забезпечення взаємодії сервісів було розроблено конфігураційний маніфест docker-compose.yml, представлений у Додатку Б.

Адміністративна та системна робота під час налаштування та адаптації цього стеку під задачі моніторингу мікроклімату полягала в конфігурації зв’язків між контейнерами та сокетів безпеки:

1. Під час конфігурації Eclipse Mosquitto було налаштовано файл параметрів mosquitto.conf (Додаток В) з визначенням інтерфейсів прослуховування мережі та активацією доступу для клієнтських вузлів у межах локальної мережі, а порт 1883 винесено на фізичний інтерфейс хост-машини для прийому трафіку від інфраструктури Wokwi Gateway.

2. Налаштування утиліти збору метрик Telegraf через файл конфігурації telegraf.conf, (Додаток Г), де у відповідній секції визначено параметри підписки на брокер повідомлень із використанням внутрішнього DNS-імені контейнера mqtt-broker:1883, інтегровано шаблони динамічного парсингу топіків для структурування вхідних потоків даних, а також прописано реквізити авторизації в базі даних.

3. Ініціалізація СУБД часових рядів InfluxDB, яка полягала в заданні ідентифікаторів організації, назви логічного сховища (bucket) для накопичення масивів інформації, а також у визначенні політик очищення застарілих даних (Retention Policies) з метою оптимізації використання дискового простору сервера.

Запуск усього серверного комплексу виконується через системний термінал командою docker-compose up -d. Системний статус успішно ініціалізованих, запущених та мережево ізольованих контейнерів відображено на рисунку 3.2.

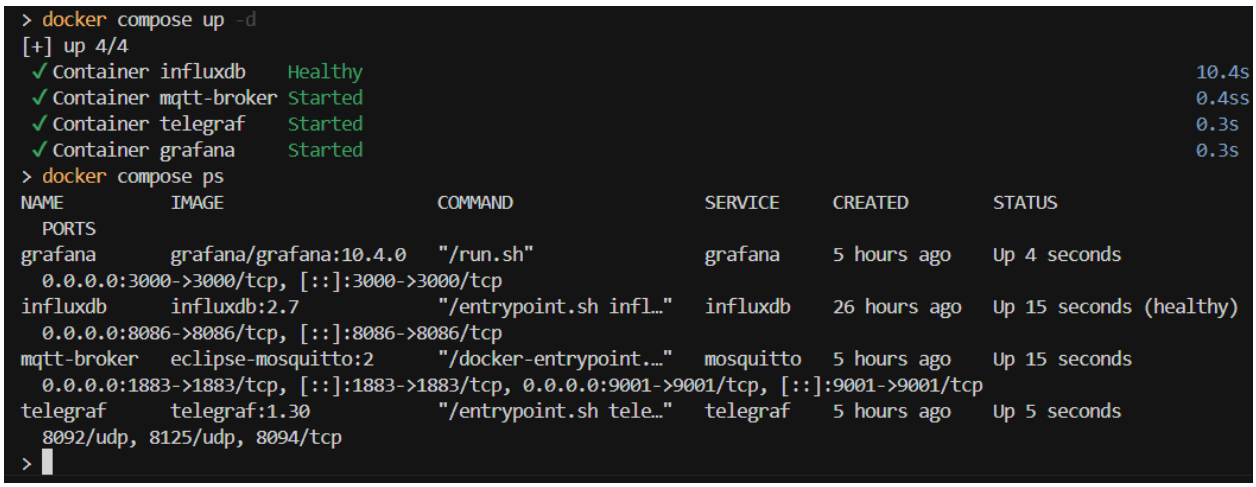

Рисунок 3.2 – Статус запущених контейнерів серверної інфраструктури в Docker Desktop

Аналіз стану віртуального середовища підтверджує, що всі сервіси функціонують стабільно (Status: Up), виділені мережеві порти (1883, 8086, 3000) успішно прив’язані до інтерфейсів, а обмін даними між контейнерами здійснюється всередині віртуального мосту iot_network без конфліктів чи затримок у передачі даних.

3.3 Налаштування сервісу візуалізації Grafana та аналіз моніторингу телеметрії

Фінальним етапом розгортання комп’ютерної системи є конфігурація прикладного рівня представлення даних та верифікація розгорнутої інфраструктури. Основне інженерне завдання на цьому етапі полягало в інтеграції платформи візуалізації Grafana з базою даних InfluxDB всередині Docker-мережі та створенні операторського інтерфейсу (дашборду).

Системне налаштування підсистеми візуалізації розпочиналося з етапу конфігурації джерела даних (Data Source). У веб-інтерфейсі Grafana було ініціалізовано підключення до бази даних, де цільовою URL-адресою вказано внутрішній сокет Docker мережі http://influxdb:8086. Безпеку та авторизацію аналітичних запитів на вибірку телеметрії було забезпечено за допомогою

унікального автентифікаційного токена, згенерованого на етапі ініціалізації СУБД.

Наступний крок охоплював безпосередню розробку та композицію операторських панелей (Widgets/Panels). Для забезпечення оперативності контролю було реалізовано паралельне виведення метрик з обох технологічних зон (Server Room 1 та Server Room 2) на єдиному робочому просторі екрана без необхідності ручного перемикання каналів. Вибірка інформації здійснювалася шляхом написання спеціалізованих запитів до відповідних таблиць і полів бази даних, які наповнюються агентом Telegraf від кожного IoT-вузла.

При конфігурації графічних елементів дашборду для поточних значень температури, вологості та концентрації газів було використано блоки типу Stat та Gauge з налаштованим колірним зонуванням критичних меж, що дозволяє фіксувати відхилення від норми. Для відстеження ретроспективної динаміки змін клімату в часі застосовано інтерактивні лінійні тренди (Time series).

Було розміщено панелі системної діагностики обчислювальних вузлів, які відображають коливання рівня бездротового сигналу RSSI та монотонне зростання метрики Uptime. Це дозволяє здійснювати комплексний технічний аудит системи в реальному часі, оцінюючи не лише параметри середовища, а й апаратну стабільність усього розподіленого комплексу.

Візуальне представлення налаштованого інтерфейсу моніторингу оператора в момент отримання реальних потокових даних від віртуального IoT-вузла наведено на рисунку 3.3.

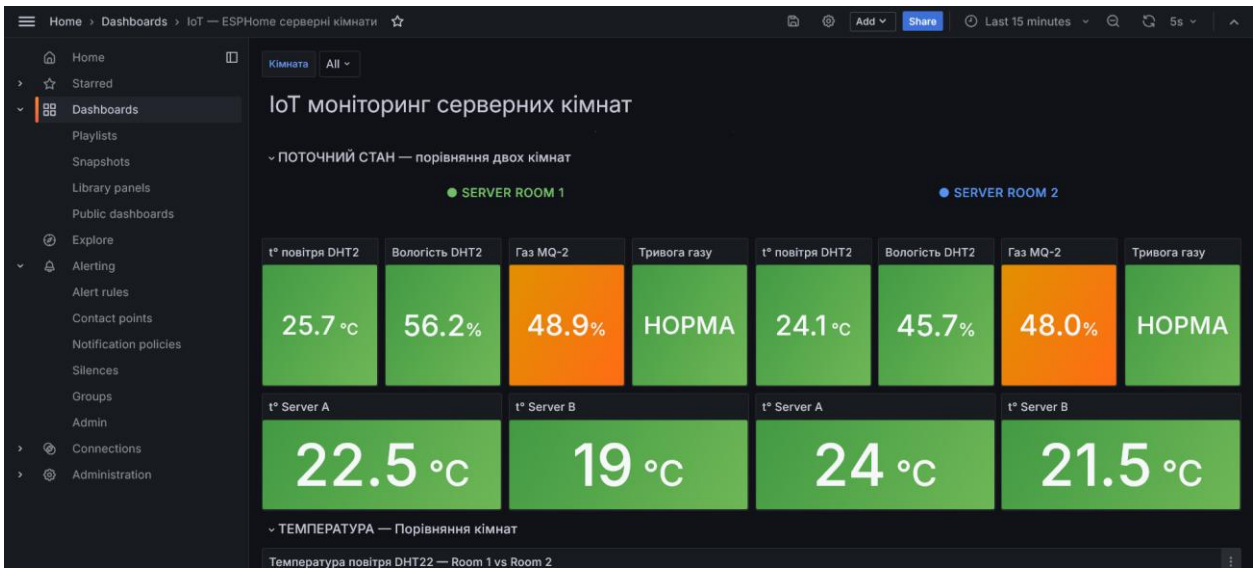

Рисунок 3.3 – Панель моніторингу параметрів мікроклімату в середовищі Grafana

Таким чином, після завершення етапів конфігурування периферійних пристроїв, контейнеризованих серверних служб та панелей візуалізації, було сформовано остаточну архітектуру проєкту. Повну ієрархічну структуру всіх створених файлів та каталогів розгорнутої системи наведено у Додатку Д.

3.4 Тестування працездатності та діагностичних можливостей системи

Для остаточного підтвердження надійності, відмовостійкості та точності функціонування розробленої комп'ютерної інфраструктури було проведено комплексне навантажувально-діагностичне тестування.

На першому етапі випробувань сценарій передбачав оцінку стабільності безперервного збору даних за умов тривалої автономної роботи. Для цього було здійснено одночасний запуск двох уніфікованих периферійних IoT-вузлів у середовищі віртуалізації, налаштованих на циклічне опитування сенсорів з інтервалом у 20 секунд. Протягом тестового періоду пристрої генерували безперервний потік телеметрії, імітуючи роботу в умовах двох окремих технологічних приміщень. Результат показав, що серверний стек у складі брокера Mosquitto та агента Telegraf успішно перехопив, десеріалізував та

зафіксував у базі даних InfluxDB всі надіслані пакети телеметрії без жодних втрат трафіку або порушення часових міток, а платформа Grafana забезпечила їх коректне паралельне відображення на єдиному дашборді оператора (рис. 3.4).

Рисунок 3.4 – Відображення параметрів телеметрії на дашборді Grafana після 30 хвилин стабільної роботи системи

Другий, важливий етап тестування полягав у верифікації аналітичних та діагностичних можливостей системи у випадку виникнення апаратних збоїв або повного знеструмлення обладнання. Для реалізації цього інженерного експерименту було здійснено перезавантаження одного з IoT-вузлів (Server Room 1). У процесі аналізу отриманих результатів на графіках Grafana було зафіксовано миттєве скидання системної метрики Uptime у нульове значення в момент відновлення зв’язку, після чого розпочалося її нове монотонне зростання (рис 3.5).

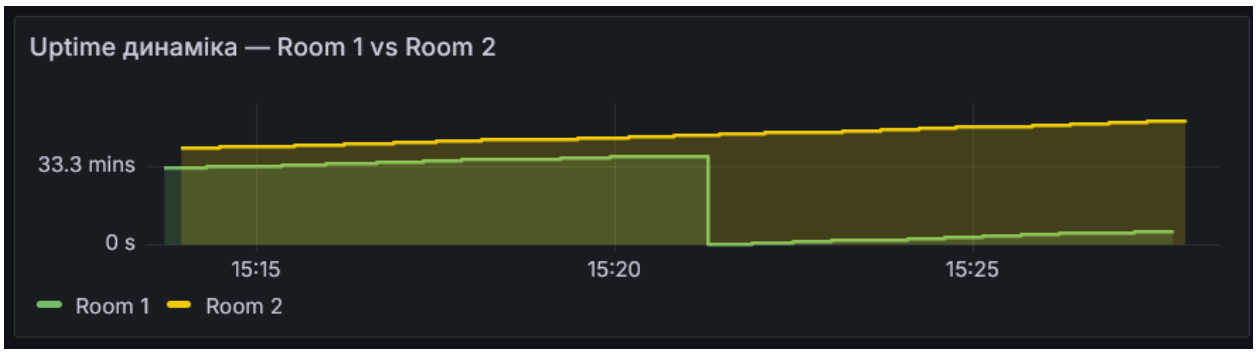

Рисунок 3.5 – Графік часового ряду метрики Uptime в момент імітації перезавантаження IoT-вузла

Проведений експеримент показує, що розроблена комп'ютерна система дозволяє оперативно здійснювати дистанційний технічний аудит заліза без необхідності фізичного огляду об'єкта. Фіксація скидання лічильника часу безперервної роботи в поєднанні з аналізом графіків Wi-Fi покриття дає змогу ідентифікувати несправності мікроконтролерів, збої у лініях електроживлення або програмних стеках прошивки ESPHome. Це підтверджує високу ефективність, практичну цінність та готовність спроєктованого комплексу до експлуатації в реальних умовах розподілених комп'ютерних мереж.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне інженерно-практичне завдання з аналізу, проєктування, реалізації та експериментального тестування комплексної системи моніторингу IoT-пристроїв.

Досліджено теоретичні засади та архітектурні принципи побудови IoT-мереж. На основі аналізу класичної трирівневої моделі обґрунтовано доцільність розподілу обчислювального навантаження між периферійними пристроями (Edge) та проміжними шлюзами (Fog). Визначено, що для систем інтенсивного збору телеметрії асинхронна модель взаємодії «видавець-підписник» є значно ефективнішою за класичну схему «запит-відповідь».

Обґрунтовано та сформовано оптимальний технологічний стек системи. Шляхом порівняльного аналізу апаратних платформ обрано мікроконтролер ESP32 завдяки наявності вбудованого Wi-Fi модуля та підтримці енергоефективних режимів роботи. Для серверної частини обрано модульний підхід: базу даних часових рядів InfluxDB, агент збору метрик Telegraf та платформу візуалізації Grafana. Кількісна оцінка накладних витрат у бездротовому каналі зв’язку підтвердила, що протокол MQTT є в ~11 разів ефективнішим за HTTP, оскільки службові заголовки пакета PUBLISH становлять лише 7.7% від загального обсягу трафіку проти 89.3% у HTTP.

Спроєктовано структурну, функціональну та мережеву архітектуру системи. Було обрано мережеву топологію типу «Зірка» навколо центрального MQTT-брокера Mosquitto. Завдяки технології контейнеризації Docker всі серверні сервіси ізольовано всередині віртуальної підмережі, що дозволило чітко розмежувати інформаційні потоки: пристрої передають дані виключно на брокер, а користувач отримує доступ до аналітики через окремий веб-інтерфейс.

Реалізовано програмне забезпечення та розгорнуто інфраструктуру моніторингу. На базі екосистеми ESPHome сформовано декларативні конфігураційні маніфести, працездатність яких верифіковано в середовищі

віртуалізації Wokwi для двох окремих технологічних зон (Server Room 1 та Server Room 2). Усі серверні компоненти автоматизовано та об’єднано в єдиний працездатний комплекс за допомогою інструменту Docker Compose, а для оператора створено інтерактивний графічний дашборд у Grafana, що забезпечує паралельне виведення кліматичних та системних метрик.

Проведено експериментальну перевірку працездатності та діагностичних можливостей розробленого рішення. Навантажувальний тест системи з інтервалом опитування датчиків у 20 секунд підтвердив її високу надійність – протягом 30 хвилин безперервної експлуатації не зафіксовано втрат пакетів телеметрії. Шляхом імітації апаратного збою та штучного перезавантаження периферійного вузла експериментально доведено ефективність дистанційного аудиту заліза, що підтверджується миттєвою фіксацією скидання лічильника Uptime у нульове значення та відстеженням динаміки бездротового сигналу RSSI на аналітичних панелях сервера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сайко В. В., Люта М. В. Порівняльний аналіз ефективності протоколів MQTT та HTTP у IoT-системах. Матеріали XVIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених за тематикою «Тенденції розвитку ІТ-технологій в Україні». 23-24 квітня 2026 р., Черкаси, Україна. Черкаси: Черкаський державний фаховий бізнес-коледж, 2026. С. 81-83.
2. IoT-Enabled Shipping Container with Environmental Monitoring and Location Tracking / K. Salah et al. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/9045495 (дата звернення: 17.05.2026).
3. Development of a Unified IoT Platform for Assessing Meteorological and Air Quality Data in a Tropical Environment / D. Kairuz et al. MDPI. URL: https://www.mdpi.com/1424-8220/24/9/2729 (дата звернення: 17.05.2026).
4. Implementation of an Internet of Things Architecture to Monitor Indoor Air Quality: A Case Study During Sleep Periods / A. Mota et al. MDPI. URL: https://www.mdpi.com/1424-8220/25/6/1683. (дата звернення: 21.05.2026).
5. From Data to Decisions: A Smart IoT and Cloud Approach to Environmental Monitoring | E3S Web of Conferences / M. Guerbaoui et al. E3S Conferences. URL: https://doi.org/10.1051/e3sconf/202560100008 (дата звернення: 21.05.2026).
6. Edge Computing: Vision and Challenges / W. Shi et al. IEEE. URL: https://ieeexplore.ieee.org/document/7488250 (дата звернення: 21.05.2026).
7. Fog computing and its role in the internet of things / F. Bonomi et al. URL: https://dl.acm.org/doi/10.1145/2342509.2342513 (дата звернення: 25.05.2026).
8. Boswarthick D., Elloumi O., Hersent O. Introduction to M2M. IEEE. URL: https://ieeexplore.ieee.org/document/8045212 (дата звернення: 25.05.2026).
9. ESPHome Architecture and Core Components Reference Manual. ESPHome Open Source Project, 2024. URL: https://esphome.io/components/esphome.html (дата звернення: 25.05.2026).

10. Environmental Monitoring System Based on EMQX Cloud Platform / Y. Wang et al. E3S Web Conferences. URL: https://www.e3s-conferences.org/articles/e3sconf/pdf/2025/30/e3sconf_epemr2025_01019.pdf (дата звернення: 28.05.2026).
11. Fielding R., Reschke J. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. IETF. URL: https://datatracker.ietf.org/doc/html/rfc7231 (дата звернення: 28.05.2026).
12. Khomenko Y., Babichev S. Modular IoT Architecture for Monitoring and Control of Office Environments Based on Home Assistant. MDPI. URL: https://www.mdpi.com/2624-831X/6/4/69 (дата звернення: 28.05.2026).
13. Suleman D., Shibi R., Ansari K. Investigation of Data Quality Assurance across IoT Protocol Stack for V2I Interactions. MDPI. URL: https://www.mdpi.com/2624-6511/6/5/121 (дата звернення: 01.06.2026).

ДОДАТКИ

Додаток А – Файл конфігурації периферійного IoT-вузла

У даному додатку наведено файл конфігурації мікроконтролера ESP32 для операційного середовища ESPHome. Конфігурація містить параметри ініціалізації апаратних інтерфейсів, бездротового зв'язку Wi-Fi та правила опитування підключених цифрових і аналогових датчиків.

Лістинг А.1 – Текст файлу програмної конфігурації server-room-1.yaml

```
esphome:
  name: server-room-1
  friendly_name: "Server Room 1"

esp32:
  board: esp32dev
  framework:
    type: arduino

# WiFi — Wokwi симулятор
wifi:
  ssid: "Wokwi-GUEST"
  password: ""
  ap:
    ssid: "Room1 Fallback"
    password: ""

# OTA оновлення
ota:
  - platform: esphome
    password: ""

# Логи в Serial (Wokwi покаже їх в консолі)
logger:
  level: INFO

# Налаштування шини I2C (для BMP180 — піни D21 та D22)
i2c:
  sda: GPIO21
  scl: GPIO22
  scan: true

# MQTT брокер
mqtt:
  broker: 192.168.1.3
  port: 1883
  client_id: esphome_room1
  topic_prefix: esphome/server_room_1
  birth_message:
    topic: esphome/server_room_1/status
```

```
    payload: online
  will_message:
    topic: esphome/server_room_1/status
    payload: offline

# Вихід для світлодіода (Пін D2)
output:
  - platform: gpio
    pin: GPIO2
    id: led_output

# Створюємо One-Wire шини для кожного датчика (Піни D5 та D18)
one_wire:
  - platform: gpio
    pin: GPIO5
    id: ow_bus_5
  - platform: gpio
    pin: GPIO18
    id: ow_bus_18

# ── Сенсори ──────────────────────────────────────────────────────────
sensor:
  # DHT22 — температура і вологість (Пін D4)
  - platform: dht
    pin: GPIO4
    model: DHT22
    update_interval: 10s
    temperature:
      name: "DHT22 Temperature"
      id: dht22_temp
      unit_of_measurement: "°C"
      state_topic: esphome/server_room_1/dht22/temperature
      accuracy_decimals: 1
    humidity:
      name: "DHT22 Humidity"
      id: dht22_hum
      unit_of_measurement: "%"
      state_topic: esphome/server_room_1/dht22/humidity
      accuracy_decimals: 1

  # BMP180 — тиск і температура (Працює через I2C)
  - platform: bmp085
    address: 0x77
    update_interval: 15s
    temperature:
      name: "BMP180 Temperature"
      unit_of_measurement: "°C"
      state_topic: esphome/server_room_1/bmp180/temperature
      accuracy_decimals: 1
    pressure:
      name: "BMP180 Pressure"
      unit_of_measurement: "hPa"
```

```
    state_topic: esphome/server_room_1/bmp180/pressure
    accuracy_decimals: 1

# DS18B20 #1 — температура сервера A (Шина на D5)
- platform: dallas_temp
  one_wire_id: ow_bus_5
  name: "DS18B20 Server A"
  unit_of_measurement: "°C"
  state_topic: esphome/server_room_1/ds18b20/server_a
  update_interval: 10s
  accuracy_decimals: 1

# DS18B20 #2 — температура сервера B (Шина на D18)
- platform: dallas_temp
  one_wire_id: ow_bus_18
  name: "DS18B20 Server B"
  unit_of_measurement: "°C"
  state_topic: esphome/server_room_1/ds18b20/server_b
  update_interval: 10s
  accuracy_decimals: 1

# MQ-2 — аналогова напруга (Пін D34)
- platform: adc
  pin: GPIO34
  name: "MQ2 Raw Voltage"
  id: mq2_voltage
  attenuation: 12db
  unit_of_measurement: "V"
  state_topic: esphome/server_room_1/mq2/voltage
  update_interval: 5s
  accuracy_decimals: 3

# Статус сигналу WiFi
- platform: wifi_signal
  name: "WiFi RSSI"
  state_topic: esphome/server_room_1/device/rssi
  update_interval: 30s

# Час роботи пристрою
- platform: uptime
  name: "Uptime"
  state_topic: esphome/server_room_1/device/uptime
  update_interval: 30s
```

Додаток Б – Файл конфігурації середовища розгортання серверного стеку

У даному додатку наведено файл конфігурації середовища контейнеризації Docker Compose. Конфігурація визначає параметри розгортання, мережевої інтеграції, механізмів відмовостійкості та тривалого збереження даних (volumes) для чотирьох взаємопов'язаних сервісів: MQTT-брокера Mosquitto, бази даних часових рядів InfluxDB, агента збору телеметрії Telegraf та аналітичної платформи візуалізації Grafana.

Лістинг Б.1 – Текст конфігураційного файлу docker-compose.yml

```
services:
  # ── MQTT Broker ──────────────────────────────────────────
  mosquitto:
    image: eclipse-mosquitto:2
    container_name: mqtt-broker
    restart: unless-stopped
    ports:
      - "1883:1883"
    volumes:
      - ./mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
      - mosquitto-data:/mosquitto/data
      - mosquitto-log:/mosquitto/log

  # ── InfluxDB ─────────────────────────────────────────────
  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    restart: unless-stopped
    env_file: .env
    ports:
      - "8086:8086"
    volumes:
      - influxdb-data:/var/lib/influxdb2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=${INFLUXDB_USERNAME}
      - DOCKER_INFLUXDB_INIT_PASSWORD=${INFLUXDB_PASSWORD}
      - DOCKER_INFLUXDB_INIT_ORG=${INFLUXDB_ORG}
      - DOCKER_INFLUXDB_INIT_BUCKET=${INFLUXDB_BUCKET}
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${INFLUXDB_TOKEN}
    healthcheck:
      test: ["CMD", "influx", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
```

```
  # ── Telegraf ──────────────────────────────────────────
  telegraf:
    image: telegraf:1.30
    container_name: telegraf
    restart: unless-stopped
    env_file: .env
    depends_on:
      influxdb:
        condition: service_healthy
      mosquitto:
        condition: service_started
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
    environment:
      - INFLUXDB_URL="http://influxdb:8086"
      - INFLUXDB_TOKEN=${INFLUXDB_TOKEN}
      - INFLUXDB_ORG=${INFLUXDB_ORG}
      - INFLUXDB_BUCKET=${INFLUXDB_BUCKET}
      - MQTT_BROKER_URL="tcp://mosquitto:1883"

  # ── Grafana ───────────────────────────────────────────
  grafana:
    image: grafana/grafana:10.4.0
    container_name: grafana
    restart: unless-stopped
    env_file: .env
    ports:
      - "3000:3000"
    depends_on:
      influxdb:
        condition: service_healthy
    volumes:
      - grafana-data:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning:ro
    environment:
      - GF_SECURITY_ADMIN_USER=${GRAFANA_USER}
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD}
      - GF_USERS_ALLOW_SIGN_UP=false
      - INFLUXDB_URL="http://influxdb:8086"
      - INFLUXDB_ORG=${INFLUXDB_ORG}
      - INFLUXDB_BUCKET=${INFLUXDB_BUCKET}
      - INFLUXDB_TOKEN=${INFLUXDB_TOKEN}

volumes:
  mosquitto-data:
  mosquitto-log:
  influxdb-data:
  grafana-data:
```

Додаток В – Конфігураційний файл параметрів MQTT-брокера

У даному додатку наведено файл системних налаштувань брокера повідомлений Eclipse Mosquitto (mosquitto.conf). Файл визначає параметри конфігурування мережевих сокетів, параметрів логування та персистентного збереження даних.

Лістинг В.1 – Текст конфігураційного файлу mosquitto.conf

```
listener 1883 0.0.0.0

# У production замінити на: password_file /mosquitto/config/passwd
allow_anonymous true

persistence true
persistence_location /mosquitto/data/

log_dest stdout
log_dest file /mosquitto/log/mosquitto.log
log_type all
```

Додаток Г – Файл конфігурації сервісу збору та структурування метрик

У даному додатку наведено конфігураційний файл сервісного агента збору даних Telegraf. Специфікація визначає параметри підписки на MQTT-топіки, правила динамічного розбору (парсингу) назв каналів для ідентифікації приміщень та реквізити для потокового запису даних у СУБД InfluxDB.

Лістинг Г.1 – Текст конфігураційного файлу telegraf.conf

```
[agent]
  interval       = "5s"
  flush_interval = "10s"
  hostname       = "telegraf-iot"
  omit_hostname  = false

[[processors.pivot]]
  tag_key   = "field"
  value_key = "value"
  [processors.pivot.tagpass]
    firmware = ["esphome"]

[[outputs.influxdb_v2]]
  urls         = ["http://influxdb:8086"]
  token        = "${INFLUXDB_TOKEN}"
  organization = "${INFLUXDB_ORG}"
  bucket       = "${INFLUXDB_BUCKET}"

[[inputs.mqtt_consumer]]
  servers     = ["tcp://mosquitto:1883"]
  client_id   = "telegraf-esp-all"
  data_format = "value"
  data_type   = "float"
  topics = [
    "esphome/+/dht22/+",
    "esphome/+/bmp180/+",
    "esphome/+/ds18b20/+",
    "esphome/+/mq2/+",
    "esphome/+/device/+",
  ]
  [[inputs.mqtt_consumer.topic_parsing]]
    topic       = "esphome/+/+/+"
    measurement = "_/_/measurement/_"
    tags        = "_/room/_/field"
  [inputs.mqtt_consumer.tags]
    firmware = "esphome"
```

Додаток Д – Структура каталогів та конфігураційних файлів проєкту

У цьому додатку наведено повну ієрархічну структуру каталогу розробленого програмно-апаратного комплексу системи моніторингу. Організація файлової структури базується на принципах модульності та розділення логіки проєкту на три ключові рівні: периферійний рівень збору даних (esphome), серверний контейнеризований стек обробки та візуалізації телеметрії (monitoring-stack), а також рівень моделювання апаратної частини (wokwi).

Лістинг Д.1 – Ієрархічна структура файлів та папок проєкту

```
├── esphome/
│   ├── room1.yaml
│   └── room2.yaml
├── monitoring-stack/
│   ├── grafana/
│   │   └── provisioning/
│   │       ├── dashboards/
│   │       │   ├── dashboards.yml
│   │       │   └── iot-dashboard.json
│   │       └── datasources/
│   │           └── influxdb.yml
│   ├── mosquitto/
│   │   └── config/
│   │       └── mosquitto.conf
│   ├── telegraf/
│   │   └── telegraf.conf
│   ├── .env
│   └── docker-compose.yml
└── wokwi/
    ├── room1/
    │   ├── diagram.json
    │   └── wokwi.toml
    └── room2/
        ├── diagram.json
        └── wokwi.toml
```