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**

# «ВЕБ-ПАНЕЛЬ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ У РЕАЛЬНОМУ ЧАСІ»

Спеціальність ***123 – Комп'ютерна інженерія***

Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Олена ПЛАХОТЯ**
*(підпис)*

Науковий керівник _____ **Ольга МОРОЗ**
*( підпис)*

Харків – 2026

# АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та чотирьох додатків. Загальний обсяг роботи становить 113 сторінки, з яких 50 сторінок основної частини, що містить 18 рисунків, 4 таблиці, 18 найменувань у списку використаних джерел.

***Метою кваліфікаційної роботи*** є забезпечення оперативного отримання, передавання, зберігання та візуалізації телеметричних даних із мінімальними затримками за рахунок системи моніторингу IoT-пристроїв у режимі реального часу на основі MQTT-протоколу, WebSocket-з'єднань та хмарних сервісів.

***Об'єктом дослідження*** є процеси збору, передавання та обробки телеметричних даних у системах Інтернету речей.

***Предметом дослідження*** є методи та програмні засоби побудови веб-орієнтованих систем моніторингу IoT-пристроїв на основі MQTT-протоколу, WebSocket-з'єднань та хмарних сервісів збереження даних.

У результаті роботи реалізовано програмний комплекс, який забезпечує моніторинг стану IoT-пристроїв, відображення телеметричних параметрів у режимі реального часу, побудову графіків часових рядів, фільтрацію та пошук пристроїв, ведення журналу подій і підтримку динамічного додавання нових вузлів моніторингу. Також реалізовано механізми контролю критичних станів та формування аварійних повідомлень.

Проведене тестування підтвердило працездатність, стабільність та відмовостійкість розробленої системи. Середній час передачі телеметричних даних від периферійного вузла до відображення у веб-інтерфейсі становив 0,3-0,6 секунди, що відповідає вимогам систем реального часу.

***Ключові слова:*** інформаційні технології, IoT, веб-панель моніторингу, телеметрія, MQTT, WebSockets, SCADA, React, TypeScript, Firebase, NoSQL, реальний час.

# ABSTRACT

The explanatory note to the thesis consists of an introduction, three chapters, conclusions, a list of references, and four appendices. The total length of the thesis is 113 pages, of which 50 pages comprise the main body, which includes 18 figures, 4 tables, and 18 entries in the list of references.

***The aim of the qualification*** thesis is to ensure the real-time acquisition, transmission, storage, and visualization of telemetry data with minimal latency through an IoT device monitoring system based on the MQTT protocol, WebSocket connections, and cloud services.

***The object of the research*** is the processes of collecting, transmitting, and processing telemetry data in Internet of Things systems.

***The subject of the research*** is the methods and software tools for developing web-based IoT device monitoring systems using the MQTT protocol, WebSocket connections, and cloud data storage services.

As a result of the work, a software system was developed that enables monitoring of IoT device status, real-time visualization of telemetry parameters, time-series graphing, device filtering and search, event logging, and support for dynamic addition of new monitoring nodes. Mechanisms for critical state detection and alarm generation were also implemented.

Testing confirmed the functionality, stability, and fault tolerance of the developed system. The average latency of telemetry transmission from the peripheral node to its display in the web interface ranged from 0,3-0,6 seconds, which meets the requirements of real-time systems.

***Keywords:*** information technologies, IoT, web-based monitoring panel, telemetry, MQTT, WebSockets, SCADA, React, TypeScript, Firebase, NoSQL, real-time systems.

# ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 6
РОЗДІЛ 1. АНАЛІЗ АРХІТЕКТУР ТА ЗАСОБІВ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ У РЕАЛЬНОМУ ЧАСІ 9
1.1 Архітектура та особливості функціонування IoT-систем 9
1.2 Методи збору, передавання та маршрутизації телеметричних даних у IoT-системах 10
1.3 Аналіз існуючих веб-панелей моніторингу 13
1.4 Аналіз апаратно-програмних платформ для побудови IoT-вузлів 17
1.5 Постановка задачі 17
Висновок до розділу 1 19
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ 21
2.1 Архітектура системи та структурна схема взаємодії компонентів 21
2.2 Обґрунтування вибору технологій 24
2.3 Організація бази даних та збереження даних 26
2.4 Проєктування користувацького інтерфейсу 29
2.5 Забезпечення інформаційної безпеки та відмовостійкості системи 37
Висновок до розділу 2 38
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ 41
3.1 Реалізація програмної моделі IoT-вузла на базі ESP32 41
3.2 Реалізація MQTT-обміну 42
3.3 Реалізація frontend-частини веб-панелі 44
3.4 Реалізація backend та збереження даних 47
3.5 Візуалізація даних у реальному часі 48
3.6 Тестування та оцінювання продуктивності системи 54
Висновок до розділу 3 56
ВИСНОВКИ 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТКИ 62

# ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

IoT – Internet of Things. Концепція мережі взаємопов'язаних фізичних пристроїв для збору та обміну даними.

HTTP – HyperText Transfer Protocol. Протокол передавання даних у вебмережі.

REST – Representational State Transfer. Архітектурний стиль побудови вебсервісів.

MQTT – Message Queuing Telemetry Transport. Легковаговий протокол обміну повідомленнями для IoT-систем.

QoS – Quality of Service. Рівень гарантії доставки повідомлень у MQTT.

LWT – Last Will and Testament. Механізм MQTT для надсилання повідомлення при аварійному відключенні клієнта.

TCP – Transmission Control Protocol. Транспортний протокол надійної передачі даних у мережі.

API – Application Programming Interface. Інтерфейс взаємодії між програмними компонентами.

TLS – Transport Layer Security. Криптографічний протокол захищеного передавання даних.

M2M – Machine-to-Machine. Технологія автоматизованого обміну даними між пристроями.

CoAP – Constrained Application Protocol. Мережевий протокол для IoT-пристроїв із обмеженими ресурсами.

UI – User Interface. Користувацький інтерфейс програмної системи.

UX – User Experience. Досвід та зручність взаємодії користувача із системою.

IP – Internet Protocol. Мережевий протокол адресації та маршрутизації пакетів даних.

DHT22 – цифровий датчик температури та вологості.

JSON – JavaScript Object Notation. Текстовий формат обміну структурованими даними.

Wi-Fi – Wireless Fidelity. Технологія бездротового доступу до мережі.

EMQX – MQTT-брокер для організації обміну повідомленнями в IoT-системах.

NoSQL – Not Only SQL. Тип нереляційних баз даних для роботи з неструктурованими даними.

CSS – Cascading Style Sheets. Мова опису стилів вебсторінок.

SCADA – Supervisory Control and Data Acquisition. Система диспетчерського керування та збору даних.

WSS – WebSocket Secure. Захищений протокол WebSocket із використанням TLS.

ACL – Access Control List. Список правил керування доступом до ресурсів.

GPIO – General Purpose Input/Output. Універсальні контакти введення/виведення мікроконтролера.

SPA – Single Page Application. Односторінковий вебзастосунок.

MQ – Message Queue. Черга повідомлень для асинхронного обміну даними.

PPM – Parts Per Million. Одиниця вимірювання концентрації речовин «частин на мільйон».

UART – Universal Asynchronous Receiver-Transmitter. Універсальний асинхронний приймачепередавач.

SPI – Serial Peripheral Interface. Послідовний периферійний інтерфейс.

$I^2C$ – Inter-Integrated Circuit. Міжінтегральна схема (дводротовий послідовний шинний інтерфейс).

# РОЗДІЛ 1
# АНАЛІЗ АРХІТЕКТУР ТА ЗАСОБІВ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ У РЕАЛЬНОМУ ЧАСІ

## 1.1 Архітектура та особливості функціонування IoT-систем

Одним із провідних напрямків розвитку сучасних інформаційних технологій є концепція IoT. Вона передбачає формування глобальної мережі фізичних об'єктів («речей»), обладнаних інтегрованими сенсорами, датчиками, виконавчими механізмами та засобами комунікації, призначеними для здійснення взаємодії як між собою, так і з навколишнім середовищем без безпосередньої участі людини або взаємодії «людина-комп'ютер», що суттєво прискорює обробку даних і підвищує загальну ефективність систем.

*Інтернет речей (Internet of Things, IoT) – це глобальна мережа підключених до Інтернету речей – пристроїв, оснащених сенсорами, датчиками, засобами передавання сигналів* [1].

Пристрої IoT охоплюють широкий спектр об'єктів – від повсякденних речей до складних промислових систем. До них належать як механічні, так і цифрові машини, транспортні засоби, побутові прилади, медичні апарати, промислове обладнання та інші об'єкти, здатні передавати дані мережею. *Річчю в Інтернеті речей може бути людина з імплантованим кардіомонітором, сільськогосподарська тварина з біочіповим транспондером, автомобіль із вбудованими датчиками, що попереджають водія про низький тиск у шинах, або будь-який інший природний чи створений людиною об'єкт, якому можна призначити адресу Інтернет-протоколу та який може передавати дані через мережу* [2].

Нині організації з різних секторів активно застосовують IoT-технології для підвищення продуктивності, покращення якості обслуговування клієнтів, оптимізації процесів прийняття рішень та збільшення бізнес-цінності. Особливу роль в екосистемі IoT відіграють системи моніторингу в реальному часі. Під терміном «реальний час» у контексті веб-панелей розуміють здатність

обробляти телеметричні дані та оновлювати графічний інтерфейс користувача із мінімальною, практично непомітною для людини затримкою – як правило, в межах часток секунди або кількох секунд.

Щоб забезпечити таку ефективність, архітектура IoT-систем повинна бути чітко структурована та містити засоби для швидкої передачі даних, обробки телеметрії й візуалізації інформації. Оскільки веб-панелі моніторингу забезпечують централізований контроль, аналіз стану пристроїв та оперативне реагування на зміни параметрів у режимі реального часу, вони є ключовим компонентом сучасних інформаційних систем.

Зазвичай при побудові архітектури IoT використовують трирівневу модель (рис. 1.1), яка забезпечує масштабованість, модульність та гнучкість в роботі.

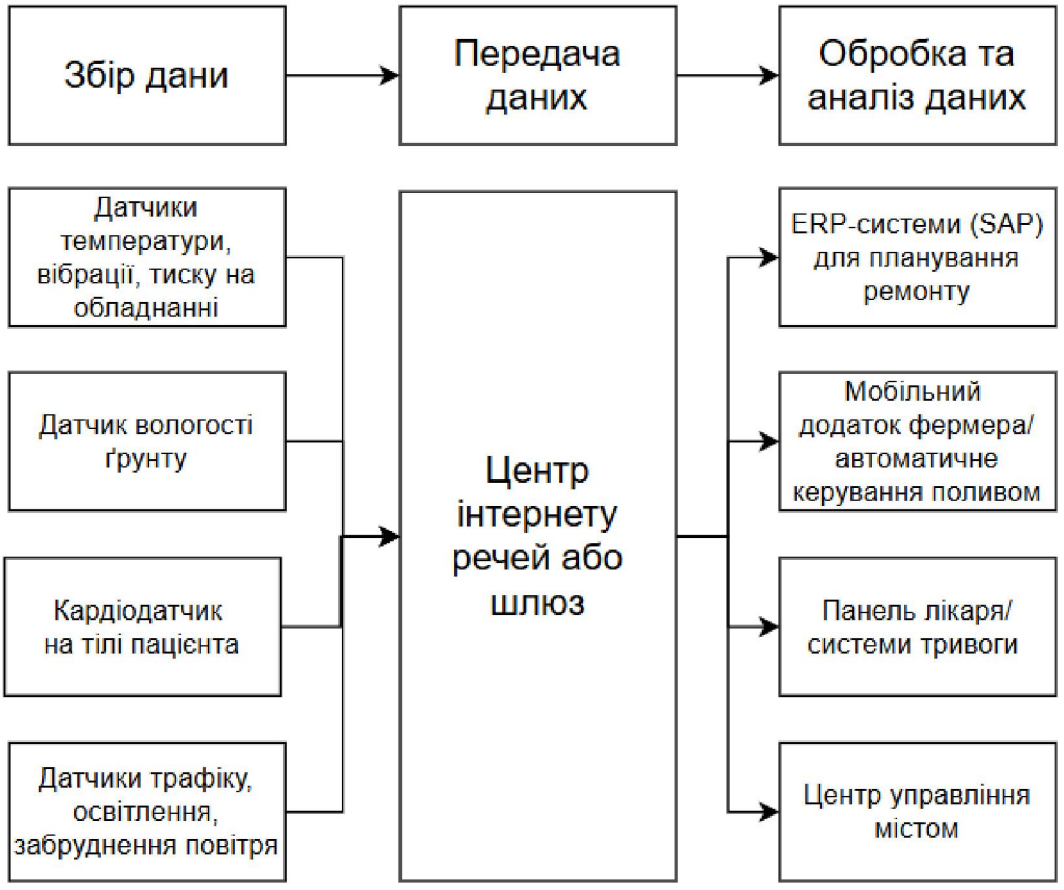



Рисунок. 1.1 – Приклад системи Інтернету речей.

## 1.2 Методи збору, передавання та маршрутизації телеметричних даних у IoT-системах

Вибір протоколів передачі даних є одним із ключових етапів проєктування IoT-систем, оскільки впливає на швидкість оновлення телеметрії, надійність зв'язку, масштабованість архітектури та енерговитрати пристроїв. Для систем моніторингу в реальному часі важливими є мінімальні затримки передачі, підтримка великої кількості підключень і сумісність із пристроями з обмеженими ресурсами, зокрема ESP32.

У сучасних IoT-рішеннях найчастіше використовуються HTTP/REST, WebSockets та MQTT.

*MQTT (Message Queuing Telemetry Transport) був розроблений у 1999 році Енді Стенфорд-Кларком з IBM та Арленом Ніппером з Arcom для моніторингу нафтопроводів через супутник — в умовах, коли пропускна здатність була дорогою, з'єднання ненадійними, а пристрої мали серйозні обмеження живлення. Саме в таких умовах досі працює більшість систем Інтернету речей* [3].

Основними перевагами MQTT є мінімальний розмір заголовків (від 2 байт), низьке енергоспоживання, підтримка нестабільних мереж, висока масштабованість, підтримка рівнів QoS та механізму LWT.

HTTP/REST функціонує за моделлю «запит-відповідь» і широко використовується для реалізації REST API, конфігурування пристроїв, OTA-оновлень та інтеграції із зовнішніми сервісами. Проте для систем реального часу він менш ефективний через необхідність періодичного опитування сервера, значний обсяг службових даних та більші затримки передачі. Тому HTTP доцільніше використовувати для адміністративних операцій і задач із низькою частотою запитів.

Для усунення цих обмежень було створено WebSockets. Після одноразового HTTP-handshake між клієнтом і сервером встановлюється постійне повнодуплексне TCP-з'єднання. У IoT-системах WebSockets

переважно використовуються для оперативного оновлення графіків, таблиць і статусів пристроїв у браузері без перезавантаження сторінки.

Недоліком технології є необхідність підтримки постійної TCP-сесії, що потребує додаткових ресурсів пам'яті та процесора. Тому пряме використання WebSockets на мікроконтролерах менш ефективне порівняно зі спеціалізованими IoT-протоколами.

У сучасних архітектурах часто застосовується MQTT через WebSockets, що дозволяє браузерним клієнтам напряму підключатися до MQTT-брокера через WebSocket-з'єднання.

Для наочності проведемо порівняльний аналіз зазначених протоколів (таблиця 1.1).

*Таблиця 1.1.*

**Порівняльний аналіз протоколів передачі даних в IoT.**

| Критерії порівнянн | MQTT | HTTP / REST | WebSockets |
|---|---|---|---|
| Архітектурна модель | Видавець-підписник (через брокер) | Запит-відповідь (Клієнт-Сервер) | Повнодуплексний двосторонній канал |
| Розмір накладних витрат (заголовки) | Низький (від 2 байт) | Великий (від сотен байт до кілобайт) | Мінімальний (кілька байт на фрейм) |
| Орієнтація технології | Машино-машинна взаємодія (M2M, IoT) | Веб-додатки, завантаження документів | Інтерактивний веб (чати, ігри, біржі) |
| Споживання енергії | Дуже низьке (ідеально для батарейних пристроїв) | Високе (через постійні TCP-хендшейки) | Середнє (потребує ресурсів на сесію) |
| Підтримка веб-браузерами | Потребує обгортки (MQTT через WebSockets) | Повна | Повна |

## 1.3 Аналіз існуючих веб-панелей моніторингу

Сучасний ринок програмного забезпечення пропонує широкий спектр рішень для збору, агрегації та відображення телеметричної інформації. Розвиток таких платформ зумовлений стрімким зростанням кількості IoT-вузлів та необхідністю уніфікації процесів обробки даних. Існуючі системи автоматизації

та моніторингу відрізняються архітектурою, призначенням, вимогами до ресурсів і гнучкістю налаштування інтерфейсу.

Серед найбільш популярних open-source платформ для роботи з IoT-системами можна виділити ThingsBoard, Grafana та Node-RED. Дані платформи представляють різні підходи до побудови IoT-інфраструктури.

Для систематизації аналізу проведено порівняння розглянутих платформ за ключовими критеріями (таблиця 1.2).

*Таблиця 1.2.*

**Порівняльний аналіз існуючих веб-панелей моніторингу**

| Критерії | ThingsBoard | Grafana | Node-RED |
|---|---|---|---|
| Основне призначення | Комплексна IoT-платформа | Візуалізація та аналітика | Flow-based інтеграція |
| Підтримка MQTT | Так | Через datasource | Так |
| Керування пристроями | Повне | Обмежене | Так |
| Гнучкість UI/UX | Середня | Середня | Низька |
| Масштабованість | Висока | Висока | Середня |
| Вимоги до серверних ресурсів | Високі | Середні | Низькі |
| Взаємодія в реальному часі | Частково | Обмежена | Висока |
| Простота кастомізації | Середня | Середня | Низька |

*ThingsBoard — це платформа Інтернету речей з відкритим кодом для збору, обробки, візуалізації та управління пристроями даних.*

*Він забезпечує підключення пристроїв через стандартні галузеві протоколи Інтернету речей – MQTT, CoAP та HTTP, а також підтримує як хмарне, так і локальне розгортання. ThingsBoard поєднує масштабованість, відмовостійкість та продуктивність, тому ви ніколи не втратите свої дані* [4].

До основних переваг ThingsBoard належать підтримка MQTT, CoAP та HTTP, наявність вбудованого Rule Engine для автоматизації обробки подій, а також готові dashboard-віджети для швидкого створення панелей моніторингу. Платформа підтримує автентифікацію пристроїв, мультиорендність (Multi-tenancy) та масштабування для роботи з великою кількістю IoT-пристроїв.

Водночас платформа потребує значних серверних ресурсів і складної початкової конфігурації з використанням додаткових компонентів інфраструктури (PostgreSQL, Cassandra, Kafka тощо). Крім того, кастомізація інтерфейсу обмежена можливостями вбудованої системи віджетів і часто потребує складного налаштування або комерційної ліцензії.

Grafana є однією з найпоширеніших open-source платформ для аналітики та візуалізації часових рядів. На відміну від повноцінних IoT-платформ, вона не приймає телеметрію безпосередньо від пристроїв, а працює поверх зовнішніх джерел даних, таких як InfluxDB, Prometheus або TimescaleDB.

На офіційному сайті Grafana зазначають, що платформа є відкритою для моніторингу та візуалізації даних, що підтримує модульну архітектуру [5].

Перевагами Grafana є високий рівень візуалізації, підтримка великої кількості джерел даних, розвинена система alerting та широкий набір плагінів для побудови інтерактивних аналітичних панелей.

Проте Grafana орієнтована переважно на аналіз даних. За замовчуванням вона не забезпечує повноцінного двостороннього зв’язку з IoT-пристроями та не призначена для реалізації логіки керування обладнанням. Для надсилання команд необхідно використовувати сторонні плагіни або окремі backend-сервіси. Крім того, швидкість оновлення інформації часто залежить від частоти оновлення бази даних, що може створювати додаткові затримки у системах реального часу.

Node-RED – це інструмент візуального програмування потоків даних, розроблений компанією IBM для швидкої інтеграції IoT-пристроїв, API та сервісів.

Система дозволяє створювати логіку взаємодії між компонентами шляхом графічного з'єднання блоків без необхідності написання великої кількості коду. Node-RED має вбудовану підтримку MQTT, WebSockets, HTTP та інших протоколів, а також dashboard-модуль для швидкого створення простих веб-інтерфейсів.

Аналізуючи інструмент Node-RED, варто відзначити його основні переваги: низький поріг входження для розробника та можливість швидкого прототипування систем автоматизації. Завдяки візуальній побудові логіки інтеграція пристроїв виконується без написання великої кількості коду, а вбудована підтримка MQTT і WebSockets дозволяє швидко об'єднувати різноманітні IoT-вузли в єдину систему.

Водночас система має й вагомі недоліки, які обмежують її використання у професійних проєктах. Насамперед це стосується жорстких рамок при кастомізації UI/UX та застарілого візуального дизайну стандартних dashboard-компонентів, які важко адаптувати під сучасні графічні вимоги. Крім того, при масштабуванні системи виникають складнощі з підтримкою та читанням великих flow-схем, а архітектурні обмеження продуктивності, зумовлені однопотоковою моделлю платформи Node.js, роблять Node-RED менш ефективним при обробці високонавантажених потоків телеметрії [6].

## 1.4 Аналіз апаратно-програмних платформ для побудови IoT-вузлів

Ефективність функціонування систем Інтернету речей значною мірою залежить від характеристик апаратної платформи, яка використовується для збору, первинної обробки та передачі телеметричних даних. Продуктивність периферійних вузлів визначає швидкість обробки інформації, можливість підтримки мережевих протоколів, рівень енергоспоживання та масштабованість усієї системи.

Найбільшого поширення набули мікроконтролери ESP8266 та ESP32, а також одноплатні комп'ютери сімейства Raspberry Pi.

ESP8266 є одним із перших масових рішень класу System-on-Chip, у якому на одному кристалі поєднано мікроконтролер і модуль бездротового

зв'язку Wi-Fi. Завдяки низькій вартості та високій енергоефективності мікроконтролер широко використовується в системах дистанційного моніторингу. Водночас обмежений обсяг оперативної пам'яті, невелика кількість периферійних інтерфейсів та відсутність спеціалізованих апаратних засобів криптографічного прискорення обмежують її використання в задачах, що потребують обробки значних потоків даних або підтримки захищених мережевих з'єднань [7].

ESP32 була розроблена з урахуванням сучасних вимог до продуктивності, багатозадачності та інформаційної безпеки. На відміну від ESP8266, мікроконтролер має значно більший обсяг оперативної пам'яті, підтримує одночасне використання технологій Wi-Fi та Bluetooth, а також містить розширений набір периферійних модулів, включаючи інтерфейси UART, SPI, $I^2C$ та багатоканальні аналого-цифрові перетворювачі. Важливою особливістю ESP32 є наявність апаратних криптографічних модулів, які забезпечують підтримку сучасних алгоритмів шифрування та дозволяють реалізовувати захищений обмін даними без великого навантаження на процесор [8].

На відміну від мікроконтролерів, платформи сімейства Raspberry Pi належать до класу одноплатних комп'ютерів і функціонують під керуванням повноцінних операційних систем на базі ядра Linux. Такі пристрої оснащуються багатоядерними ARM-процесорами, мають значний обсяг оперативної пам'яті та підтримують широкий спектр мережевих інтерфейсів, включаючи Ethernet, Wi-Fi та Bluetooth. Висока обчислювальна потужність дозволяє використовувати Raspberry Pi для виконання складних алгоритмів аналізу даних, локальної обробки інформації та реалізації концепції периферійних обчислень. Разом із цим підвищене енергоспоживання, відсутність вбудованих аналогових входів та вища вартість роблять використання таких платформ не завжди доцільним для задач циклічного збору та передачі телеметричних даних [9].

Для систематизації та узагальнення результатів проведеного дослідження, а також оцінки придатності різних апаратних рішень для побудови IoT-вузлів, основні технічні характеристики розглянутих платформ наведено в таблиці 1.3.

*Таблиця 1.3.*

**Порівняльний аналіз апаратних платформ для побудови IoT-вузлів**

| Технічний критерій | ESP8266 | ESP32 | Raspberry Pi (Model 4) |
|---|---|---|---|
| Архітектурний клас | 32-bit Мікроконтролер | 32-bit Мікроконтролер | 64-bit Одноплатна ЕОМ |
| Обчислювальні ядра / Частота | 1 ядро / 80 МГц | 2 ядра / 160–240 МГц | 4 ядра / 1,5 ГГц |
| Об'єм оперативної пам'яті (RAM) | ~80 КБ | 520 КБ | 2 - 8 ГБ |
| Мережеві бездротові інтерфейси | Wi-Fi 2.4 ГГц | Wi-Fi 2.4 ГГц, BLE 4.2/5.0 | Wi-Fi 2,4/5 ГГц, Bluetooth |
| Аналогові інтерфейси (АЦП) | 1 канал (10-bit) | 18 каналів (12-bit) | Відсутні (потребує розширення) |
| Криптографічний захист зв'язку | Обмежений (програмний) | Повний апаратний (AES, RSA) | Повний (на рівні ядра ОС) |
| Енергоефективність архітектури | Висока | Висока | Низька (потребує активного живлення) |
| Економічна доцільність вузла | Максимальна | Оптимальна | Низька (для локальних задач) |

Проведений аналіз дозволяє зробити висновок, що для реалізації периферійного рівня систем моніторингу в режимі реального часу найбільш доцільним є використання платформи ESP32. Двоядерна архітектура забезпечує можливість одночасного виконання мережевих та прикладних задач, що сприяє підвищенню стабільності роботи вузла. Підтримка сучасних засобів криптографічного захисту дозволяє реалізувати безпечну передачу телеметричних даних через мережу Інтернет, а широкий набір периферійних інтерфейсів забезпечує інтеграцію різноманітних сенсорів та виконавчих пристроїв. Крім того, поєднання обчислювальної потужності, енергоефективності та невисокої вартості робить ESP32 оптимальним мікроконтролером для побудови масштабованих IoT-систем моніторингу реального часу.

## 1.5 Постановка задачі

Метою даної роботи є проєктування та програмна реалізація веб-панелі для моніторингу телеметричних даних IoT-пристроїв у режимі реального часу. Розроблювана система повинна забезпечувати повний цикл обробки даних — від генерації показників датчиків на мікроконтролері до їх відображення у веб-інтерфейсі без необхідності ручного оновлення сторінки.

Основним призначенням системи є безперервний збір, передавання, обробка та візуалізація телеметричних параметрів, зокрема температури та вологості, із використанням сучасних IoT-протоколів і вебтехнологій. Особлива увага приділяється мінімізації затримок передачі даних, стабільності з'єднання та ефективній роботі в умовах обмежених ресурсів мікроконтролерів.

Для реалізації поставленої мети система повинна відповідати низці архітектурних, функціональних і технічних вимог.

Архітектура має будуватися за модульним принципом із розділенням на рівні збору даних, транспорту та представлення інформації. Це забезпечує незалежність компонентів системи та спрощує її масштабування і модифікацію.

Як джерело телеметрії передбачається використання віртуального мікроконтролера ESP32 у середовищі Wokwi з підключеним цифровим датчиком, наприклад DHT22. Передавання даних має здійснюватися через мережу Wi-Fi з використанням MQTT-брокера, що підтримує TCP/IP-з'єднання.

Клієнтська частина повинна бути реалізована у вигляді кросплатформеного вебзастосунку, здатного працювати у сучасних браузерах без встановлення додаткового програмного забезпечення. Для забезпечення швидкої взаємодії frontend-частина має підтримувати пряме підключення до джерела даних через MQTT over WebSockets.

Система повинна забезпечувати автоматичний збір телеметричних даних із сенсорів, їх пакування у формат JSON та передавання через MQTT-протокол. Веб-панель має підтримувати постійне з'єднання з MQTT-брокером і автоматично реагувати на появу нових повідомлень у режимі реального часу.

Інтерфейс користувача повинен відображати поточні значення параметрів за допомогою цифрових віджетів і графіків часових рядів, які автоматично оновлюються без перезавантаження сторінки. Також має бути реалізований механізм контролю стану з'єднання з IoT-пристроєм. У разі відсутності телеметричних пакетів протягом визначеного часу веб-панель повинна автоматично змінювати статус пристрою на «Офлайн».

Додатково передбачається можливість двосторонньої взаємодії між веб-панеллю та IoT-вузлом, зокрема надсилання керуючих команд на мікроконтролер для керування виконавчими елементами або індикаторами.

Одним із ключових критеріїв системи є мінімальна затримка передачі даних. Час від моменту зміни параметра датчика до відображення оновленого значення у веб-інтерфейсі не повинен перевищувати 2–3 секунди.

Система має забезпечувати низький рівень мережевих накладних витрат завдяки використанню MQTT-протоколу з мінімальним розміром службових заголовків. Це дозволяє зменшити обсяг мережевого трафіку та енергоспоживання IoT-пристроїв порівняно з традиційними HTTP-запитами.

Веб-панель повинна підтримувати плавне оновлення графіків та інтерфейсу без повного перерендерингу сторінки, забезпечуючи стабільну роботу навіть при тривалому моніторингу. Важливою вимогою є також надійність системи. У разі втрати мережевого з'єднання або перезапуску MQTT-брокера як мікроконтролер, так і вебзастосунок повинні автоматично виконувати повторне підключення без втручання користувача.

Таким чином, розроблювана система повинна поєднувати властивості моніторингу в реальному часі, енергоефективності, масштабованості та кросплатформеності, забезпечуючи ефективну взаємодію між IoT-пристроями та веб-інтерфейсом користувача.

**Висновок до розділу 1**

У першому розділі проведено аналіз архітектурних особливостей систем Інтернету речей та сучасних підходів до організації збору, передачі й візуалізації телеметричних даних. Встановлено, що IoT-технології забезпечують

автоматизовану взаємодію великої кількості пристроїв та є основою побудови сучасних систем моніторингу в реальному часі. Розглянуто трирівневу архітектуру IoT-систем, яка включає рівень збору даних, транспортний рівень та рівень представлення інформації.

У межах дослідження виконано порівняльний аналіз протоколів HTTP/REST, WebSockets та MQTT. Визначено, що HTTP є доцільним для реалізації веб-сервісів та адміністративних функцій, проте поступається за ефективністю в задачах реального часу. WebSockets забезпечують постійне двостороннє з'єднання з низькими затримками, тоді як MQTT завдяки моделі publish-subscribe, мінімальним накладним витратам і низькому енергоспоживанню є найбільш придатним для IoT-систем моніторингу.

Також проведено аналіз існуючих платформ ThingsBoard, Grafana та Node-RED, який показав, що кожне рішення має власні переваги та обмеження щодо масштабованості, функціональності та гнучкості налаштування. Крім того, досліджено апаратні платформи ESP8266, ESP32 та Raspberry Pi. За результатами порівняння встановлено, що ESP32 є найбільш збалансованим рішенням завдяки поєднанню достатньої обчислювальної потужності, енергоефективності, широких можливостей підключення периферії та підтримки сучасних засобів захисту даних.

На основі проведеного аналізу сформульовано вимоги до розроблюваної системи моніторингу. Визначено, що система повинна забезпечувати безперервний збір, передачу та відображення телеметричних даних у режимі реального часу, підтримувати автоматичне оновлення інтерфейсу, контроль стану пристроїв, можливість масштабування та надійне функціонування в умовах нестабільного мережевого з'єднання.

Таким чином, результати аналізу підтвердили доцільність розробки веб-панелі моніторингу IoT-пристроїв на основі MQTT, WebSockets та мікроконтролера ESP32, що дозволить забезпечити високу швидкодію, масштабованість і ефективність роботи системи в режимі реального часу.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ МОНІТОРИНГУ IOT-ПРИСТРОЇВ

## 2.1 Архітектура системи та структурна схема взаємодії компонентів

Архітектура розроблюваної системи побудована з урахуванням вимог до IoT-застосунків реального часу, серед яких ключовими є мінімальна затримка передачі телеметрії, масштабованість, стабільність з'єднання та можливість одночасної роботи великої кількості клієнтів. Для досягнення цих вимог використано модульний підхід із розподілом функцій між окремими рівнями системи: рівнем збору даних, транспортним рівнем та рівнем представлення інформації.

Структурна схема системи наведена на рисунку 2.1.




Рисунок 2.1 – Структурна схема архітектури системи моніторингу IoT-пристроїв.

На відміну від класичних вебзастосунків із синхронною REST API взаємодією, у даній системі впроваджено модель асинхронного стрімінгу через

протокол MQTT over WebSockets на базі патерну «Видавець-Підписник». Такий підхід забезпечує слабку зв'язаність компонентів, спрощує масштабування системи та дозволяє модернізувати окремі модулі без впливу на роботу всієї архітектури.

На рівні збору телеметрії використовується віртуальний мікроконтролер ESP32 у середовищі Wokwi. Обраний підхід дозволяє виконувати повноцінне моделювання роботи IoT-пристрою без використання фізичного обладнання, що суттєво спрощує тестування та налагодження системи. До ESP32 підключено цифровий датчик DHT22, з якого зчитуються значення температури та вологості. Отримані значення пакуються у формат JSON та передаються через MQTT-брокер за допомогою Wi-Fi-з'єднання. Використання JSON забезпечує просту інтеграцію між різними компонентами системи та спрощує подальший парсинг даних у frontend-застосунку.

Центральним елементом транспортного рівня є MQTT-брокер, який виконує маршрутизацію повідомлень між вузлами системи, підтримує підписки клієнтів та забезпечує оперативну доставку телеметрії. Для взаємодії вебзастосунку з брокером використовується MQTT over WebSockets, що дозволяє браузеру підтримувати постійне з'єднання через стандартний HTTPS-порт та отримувати дані в режимі реального часу.

Особливістю реалізованої системи є використання MQTT over WebSockets для взаємодії frontend-застосунку з брокером. У такому режимі браузер відкриває постійне WebSocket-з'єднання з MQTT-брокером через стандартний HTTPS-порт 443, що забезпечує сумісність із сучасними браузерами та мережевими обмеженнями. Після отримання нового MQTT-повідомлення frontend асинхронно оновлює стан застосунку та виконує локальний ререндер лише необхідних компонентів інтерфейсу.

Backend-рівень виконує допоміжні функції, пов'язані із збереженням історичних даних, наданням REST API та обробкою аналітичних сценаріїв. Такий підхід зменшує навантаження на серверну частину під час передачі телеметрії та підвищує загальну продуктивність системи.

Клієнтська частина реалізована з використанням React і TypeScript. Вебпанель забезпечує відображення поточних показників, графіків часових рядів, статусів підключення та активності пристроїв. Для підвищення надійності реалізовано механізми автоматичного перепідключення до MQTT-брокера та контролю активності IoT-вузлів із автоматичним переведенням їх у стан «Offline» за відсутності телеметричних повідомлень.

Запропонована архітектура забезпечує високу швидкодію, модульність і масштабованість системи, а поєднання MQTT, WebSockets та React дозволяє реалізувати ефективний моніторинг телеметричних даних у режимі реального часу з мінімальними затримками.

## 2.2 Обґрунтування вибору технологій

При проєктуванні системи основними критеріями вибору технологічного стеку було забезпечення мінімальної затримки, масштабованість інтерфейсу, енергоефективність комунікаційного каналу для мікроконтролерів.

За допомогою компонентної архітектури React, робота із постійним потоком даних, які надходять від датчиків, стає простішою, оскільки даний фреймворк дозволяє створювати повторно використовуванні та незалежні елементи інтерфейсу. Даний вибір особливо ефективний для web-панелі моніторингу Iot-пристроїв у реальному часі, оскільки інтерфейс складається із багатьох динамічних компонентів, таких як індикатори стану, діаграми, панелі телеметрії, сповіщення тощо. React оновлює ці частини сторінки за допомогою використання технології Virtual DOM без повного перезавантаження, що мінімізує навантаження на центральний процесор клієнта.

Ще однією перевагою React є підтримка управління станом та обробки даних у режимі реального часу. Він добре інтегрується з такими технологіями, як WebSockets та MQTT, які були використані для передачі даних. Це дозволяє панелі миттєво відображати зміни значень датчиків, станів пристроїв, сповіщень та іншої телеметричної інформації без необхідності перезавантаження сторінки.

Дані від мікроконтролерів надходять до системи асинхронно, мають динамічний характер та можуть бути нестабільними. Для уникнення каскадних Runtime-помилок та мережевих збоїв добре підходить використання суворої типізації TypeScript, що дозволяє зафіксувати структури даних на рівні інтерфейсів.

Інструментом збірки було обрано Vite, оскільки він забезпечує швидке та сучасне середовище розробки. На відміну від стандартних пакетів, він використовує нативні модулі ES, що значно скорочують час запуску та перебудови. Це дуже зручний елемент, оскільки часто оновлюються компоненти інтерфейсу, діаграми та логіка візуалізації в реальному часі, без повного оновлення сторінки, підвищуючи ефективність та продуктивність розробки.

Vite також створює оптимізовані збірки для середовища з ефективним розділенням коду та об'єднанням ресурсів, що покращує швидкість завантаження програм та загальну продуктивність. Це є важливим елементом для веб-панелі моніторингу даних, які можуть працювати на різних пристроях та мережах, обробляючи великі обсяги телеметричних даних у режимі реального часу.

При проєктуванні системи протоколи HTTP (модель запит-відповідь) було відкинуто через неефективність для панелі, оскільки потрібно постійно проводити опитування сервера клієнтом, що перевантажує мережевий трафік і призводить до надмірного енергоспоживання мікроконтролерів. Замість нього було надано перевагу протоколам MQTT (модель видавець-підписник), щоб забезпечити низький рівень затримок та високої пропускної здатності. Він має мінімальний розмір службового заголовка пакета (від 2 байт), що дозволяє мікроконтролеру миттєво публікувати дані в канал мережі й одразу вивільняти радіоканал. Але використання лише протоколу MQTT є недостатнім, оскільки веб-браузери не мають вбудованої підтримки чистих TCP/IP сокетів. Їх взаємодію адаптовано через WebSocket-міст на базі хмарного брокера EMQX. Це дозволяє утримувати єдине постійне двостороннє з'єднання, через яке брокер самостійно надсилає нові JSON-пакети телеметрії в React-додаток у

момент їх генерації залізом. Використання захищеного протоколу wss забезпечує шифрування трафіку за стандартом TLS.

Оскільки мікроконтролери генерують та надсилають пакети у текстовому форматі JSON, NoSQL база даних дозволяє зберігати їх такими, якими вони є, без додаткових важких трансформацій. Гнучка безсхемна структура бази даних забезпечує легке масштабування системи: у разі розширення пакета телеметрії нові поля записуються автоматично без необхідності зупинки бази даних чи зміни зв'язків реляційних таблиць.

Завдяки вбудованому інструменту живих слухачів, Firebase забезпечує автоматичну синхронізацію даних між хмарою та інтерфейсом React. Це створює додатковий рівень відмовоїстійкості: при короткочасних втратах зв'язку з MQTT-брокером, інтерфейс оператора зберігає стабільність, відображаючи останні валідні дані з бази, та коректно будує графіки на основі вибірки фіксованої кількості історичних подій.

## 2.3 Організація бази даних та збереження даних

Для забезпечення швидкої обробки телеметричних потоків, синхронізації даних у режимі реального часу та мінімізації затримок у веб-інтерфейсі у системі було використано хмарну NoSQL-платформу Firebase. Використання Firebase обумовлене її орієнтованістю на real-time застосунки, підтримкою асинхронної взаємодії та можливістю автоматичної синхронізації даних між серверною інфраструктурою та клієнтськими застосунками.

На відміну від реляційних систем керування базами даних, де структура інформації жорстко визначається таблицями та зв’язками між ними, Firebase використовує документо-орієнтовану модель зберігання даних. Такий підхід є більш придатним для IoT-систем, оскільки телеметричні повідомлення мають динамічний характер, надходять асинхронно та часто формуються у вигляді JSON-об’єктів [10].

Організація структури бази даних виконана з урахуванням необхідності окремого зберігання поточного стану пристроїв та історичних телеметричних

записів. Це дозволяє зменшити навантаження на клієнтську частину системи та оптимізувати процеси відображення даних у реальному часі.

У структурі бази даних виділено дві основні логічні сутності:

- devices_status – вузол для збереження актуального стану IoT-пристроїв;
- telemetry_history – вузол для накопичення історичних телеметричних даних.

Логічна структура збереження інформації наведена у вигляді JSON-представлення:

```
{
 "devices_status": {
  "esp32_node_01": {
   "device_name": "ESP32 Sensor",
   "is_online": true,
   "last_update": 1779645800,
   "latest_temperature": 23.5,
   "latest_humidity": 48.0
  }
 },
 "telemetry_history": {
  "esp32_node_01": {
   "msg_1779645500": {
    "timestamp": 1779645500,
    "temperature": 23.2,
    "humidity": 49.1
   },
   "msg_1779645800": {
    "timestamp": 1779645800,
    "temperature": 23.5,
    "humidity": 48.0
   }
  }
 }
}
```

Лістінг 2.1 – Логічна NoSQL-структура документів та колекцій часових рядів телеметрії.

Вузол devices_status використовується для оперативного відображення поточного стану IoT-пристрою у веб-панелі. При надходженні нового MQTT-повідомлення попередні значення автоматично замінюються актуальними, що забезпечує швидкий доступ до останнього стану пристрою без необхідності аналізу накопичених телеметричних даних. Такий підхід зменшує

навантаження на клієнтську частину системи та прискорює оновлення інтерфейсу.

Вузол telemetry_history призначений для накопичення історичних даних. На відміну від devices_status, записи в ньому не перезаписуються, а послідовно додаються у вигляді окремих JSON-об'єктів із часовими мітками. Це дозволяє формувати графіки часових рядів, аналізувати зміни параметрів та виконувати подальшу обробку телеметрії.

Для підвищення продуктивності веб-панель завантажує лише обмежену кількість останніх записів, що знижує споживання пам'яті браузера та запобігає перевантаженню інтерфейсу під час тривалої роботи системи.

Важливою особливістю Firebase є підтримка постійних підписок на зміни даних. Завдяки цьому клієнтський застосунок автоматично отримує оновлення без періодичного HTTP-опитування сервера, забезпечуючи безперервну синхронізацію між IoT-пристроями, базою даних та веб-інтерфейсом.

Додатковою перевагою Firebase є підвищення відмовостійкості системи. У разі короткочасної втрати з'єднання веб-панель продовжує працювати з локально кешованими даними, а після відновлення зв'язку автоматично виконується синхронізація та оновлення інформації.

## 2.4 Проєктування користувацького інтерфейсу

Одним із ключових етапів розробки систем моніторингу IoT-пристроїв є проектування ефективного користувацького інтерфейсу, здатного забезпечити швидке сприйняття телеметричної інформації, оперативне реагування на аварійні події та стабільну роботу в умовах безперервного надходження даних. Для систем диспетчеризації та моніторингу реального часу особливого значення набувають швидкість візуального аналізу, мінімізація когнітивного навантаження на оператора та адаптація інтерфейсу до різних типів пристроїв.

Розроблена веб-панель побудована за модульним принципом із використанням бібліотеки React та мови TypeScript. Такий підхід дозволяє ізолювати окремі функціональні елементи системи, спростити масштабування програмного комплексу та забезпечити зміну даних компонентів інтерфейсу без

повного оновлення сторінки. Для стилізації інтерфейсу використовується CSS-фреймворк Tailwind CSS, який забезпечує реалізацію адаптивного дизайну та уніфікованої системи візуального оформлення.

Основою користувацького інтерфейсу є центральна інформаційна панель (Dashboard), що виконує роль єдиного диспетчерського простору для відображення поточного стану IoT-системи. Архітектура Dashboard проектувалася таким чином, щоб оператор отримував повну інформацію про стан обладнання без необхідності переходу між окремими сторінками чи вкладками. Верхня частина інтерфейсу містить блок агрегованих показників системи, де відображається загальна кількість пристроїв, кількість активних вузлів та поточний режим роботи системи моніторингу. Розташування даних у верхній області забезпечує швидке сканування критичних параметрів під час роботи оператора.

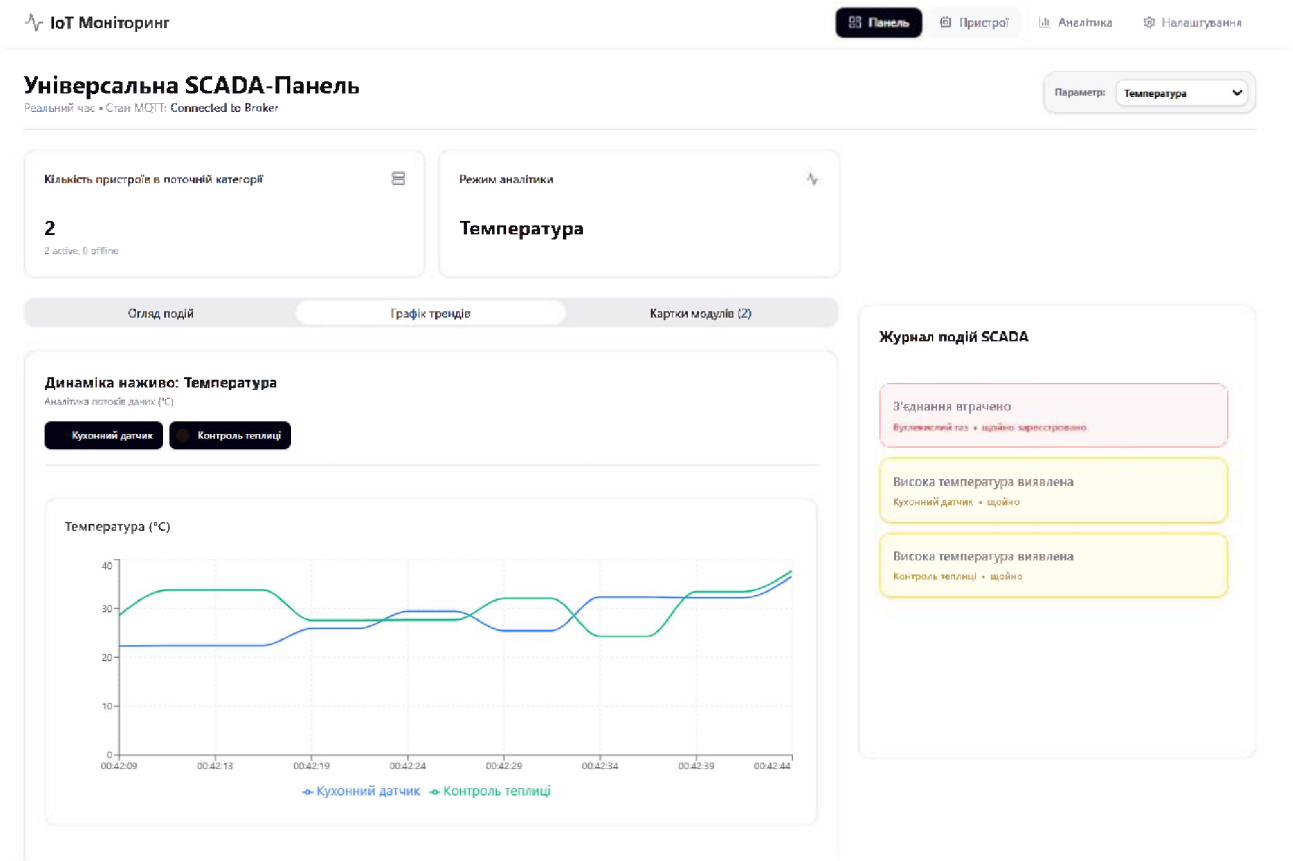



Рисунок 2.2 – Головне вікно веб-панелі моніторингу IoT-пристроїв.

Основна робоча область реалізована у вигляді блокової структури (див. рис. 2.2). Ліва частина інтерфейсу використовується для відображення динамічних компонентів системи – графіків, аналітичних модулів та списку пристроїв. Права частина містить окремий модуль журналу подій та аварійних повідомлень, який постійно залишається у полі зору користувача. Такий підхід дозволяє забезпечити безперервний контроль критичних інцидентів незалежно від того, який функціональний модуль відкрито в центральній області Dashboard.

Для забезпечення реактивності інтерфейсу всі компоненти підключені до централізованого стану застосунку через механізм React Context. При надходженні нового MQTT-повідомлення відбувається локальне оновлення лише тих компонентів, які залежать від змінених даних. Це дозволяє мінімізувати навантаження на браузер та забезпечити плавне оновлення інформації в режимі реального часу.

Ключовим функціональним елементом веб-панелі є картки пристроїв (Device Cards), які виконують роль ізольованих мікроінтерфейсів для кожного IoT-вузла (див. рис. 2.3.). На відміну від класичних реалізацій, де один компонент відображає лише окремий параметр, у розробленій системі використовується багатоканальна структура відображення даних. Картка пристрою може одночасно відображати температуру, вологість, концентрацію $CO_2$, стан реле та інші телеметричні параметри залежно від конфігурації конкретного пристрою.

У межах системи також реалізовано уніфіковану систему візуального кодування метрик. Кожен тип телеметричних даних асоціюється з фіксованою кольоровою схемою та графічною піктограмою. Наприклад, температура відображається помаранчевим кольором із піктограмою термометра, вологість – синім кольором із символом краплі води, а показники якості повітря – зеленими індикаторами. Такий підхід формує стійкі візуальні асоціації та прискорює розпізнавання критичних параметрів.

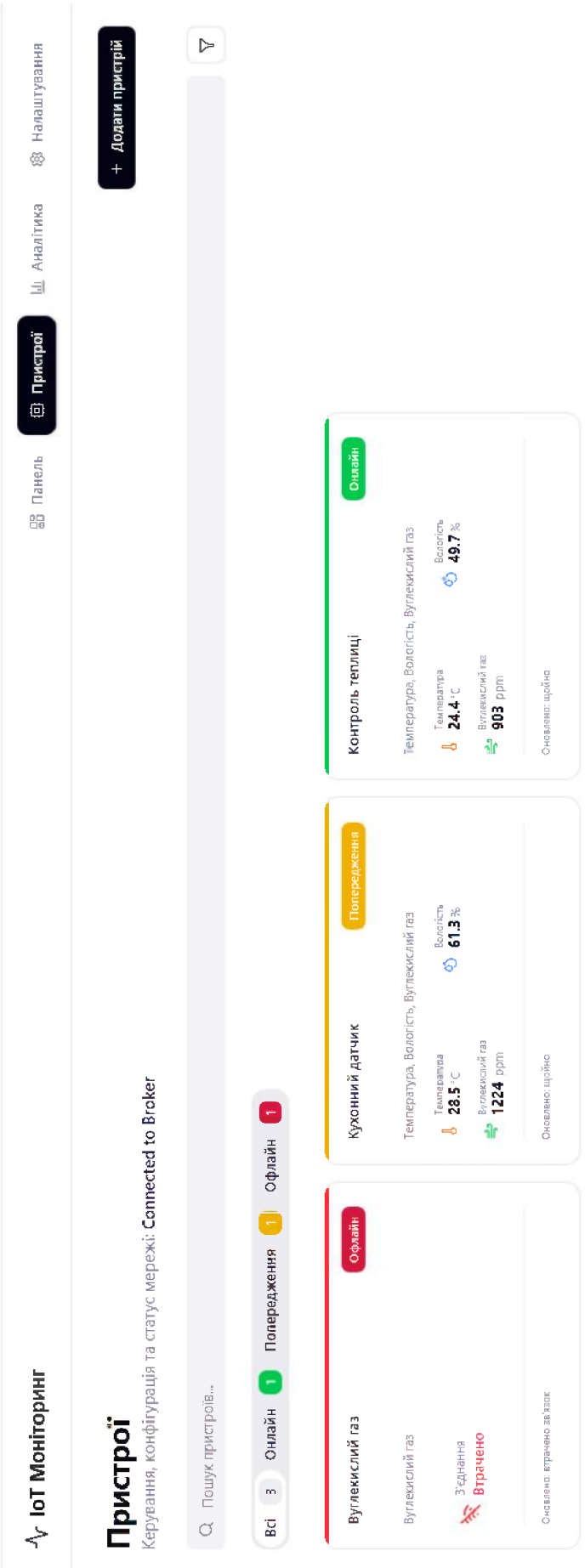



Рисунок 2.3 – Вікно пристроїв веб-панелі моніторингу IoT-пристроїв.

Одним із найважливіших модулів інтерфейсу є система графічної візуалізації часових рядів. У веб-панелі реалізовано два типи графіків: графіки оперативного моніторингу та архівні аналітичні графіки. Графіки оперативного контролю призначені для відображення поточних змін параметрів у режимі реального часу (див. рис. 2.2.) . Дані надходять із Firebase та MQTT-брокера й автоматично додаються до графіка без перезавантаження сторінки. Для оптимізації швидкості завантаження інтерфейсу використовується механізм вибірки останніх точок телеметрії, що дозволяє миттєво відобразити початкову лінію після відкриття веб-панелі.

Окрему увагу приділено забезпеченню коректної роботи графіків у випадку втрати зв’язку з пристроєм. Якщо IoT-вузол переходить у стан Offline, система автоматично очищує буфер поточних даних, а лінія графіка переривається. Такий підхід дозволяє уникнути появи помилкових безперервних значень на часових графіках.

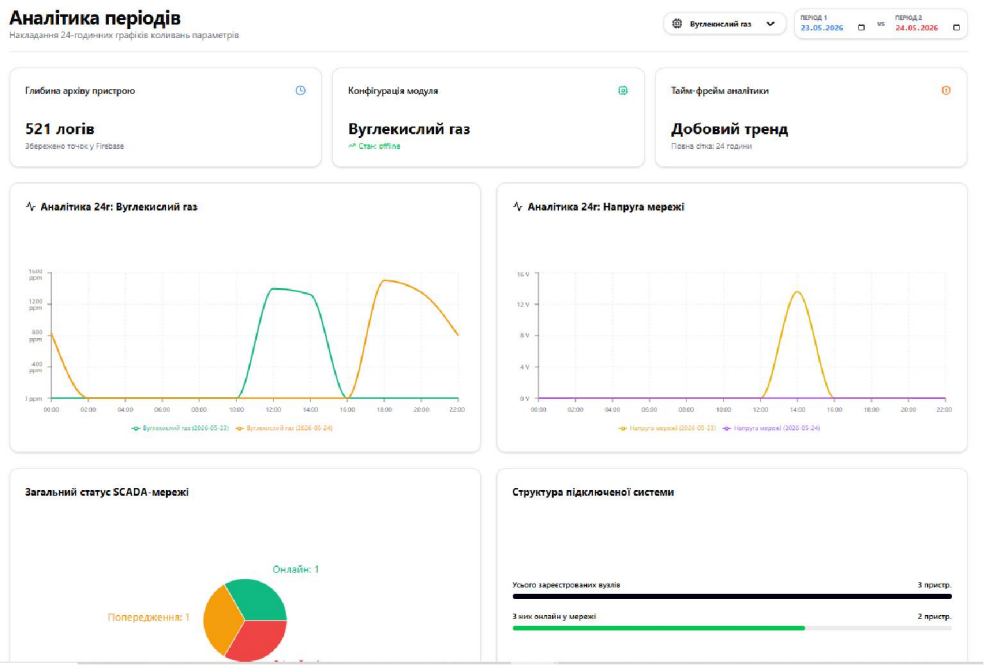



Рисунок 2.4 – Вікно аналітики даних веб-панелі моніторингу IoT-пристроїв.

Для реалізації ретроспективного аналізу у веб-панелі передбачено модуль порівняльної аналітики (див. рис. 2.4.). Користувач може обирати різні часові інтервали та порівнювати зміни параметрів між різними днями або періодами експлуатації системи. Графіки будуються шляхом накладання часових рядів на спільну шкалу часу, що дозволяє швидко виявляти аномалії та закономірності у роботі IoT-пристроїв.

Система моніторингу також реалізує багаторівневий механізм відображення станів пристроїв. Для цього використовується принцип кольорової індикації, де зелений статус означає штатний режим роботи, жовтий – перевищення допустимих порогових значень, а червоний – втрату зв'язку або аварійний стан (див. рис. 2.3). У випадку критичних подій система автоматично генерує повідомлення у журналі подій та активує візуальні анімаційні ефекти для привернення уваги оператора (див. рис. 2.2).

Контроль втрати з'єднання реалізовано за допомогою спеціального механізму Watchdog. Оскільки пристрій у разі аварійного вимкнення не може повідомити систему про власний стан, веб-панель виконує фонову перевірку часу останнього отриманого повідомлення. Якщо телеметричні пакети не надходять протягом визначеного інтервалу часу, система автоматично переводить пристрій у стан Offline та формує аварійне повідомлення.

Окремим функціональним модулем веб-панелі є сторінка налаштувань, призначена для конфігурації параметрів моніторингу та керування системою сповіщень. Інтерфейс дозволяє змінювати допустимі мінімальні та максимальні межі телеметричних показників, зокрема температури, вологості та концентрації $CO_2$. У разі перевищення встановлених значень система автоматично формує попередження та змінює статус пристрою (див. рис. 2.5).

Крім того, користувач може централізовано керувати сповіщеннями: повністю вимикати їх, залишати лише критичні аварійні повідомлення або лише про вимкнені пристрої. Для зручності взаємодії сторінка використовує перемикачі, форми введення та адаптивне компонування, що забезпечує коректну роботу як на комп'ютерах, так і на мобільних пристроях.

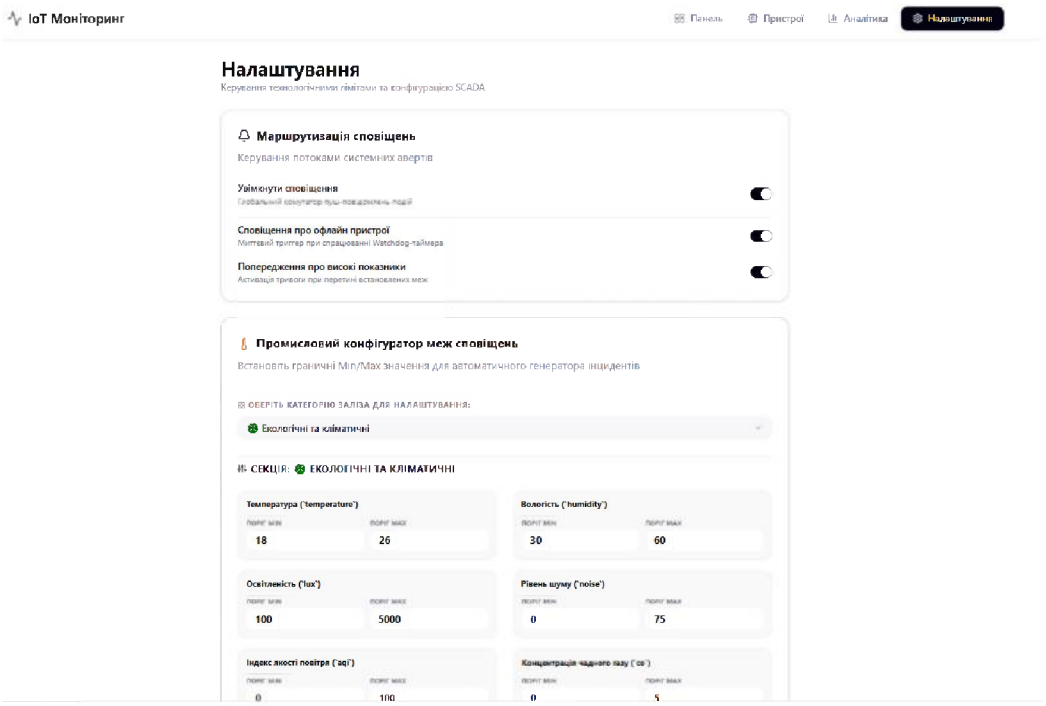



Рисунок 2.5 – Вікно налаштувань веб-панелі моніторингу IoT-пристроїв.

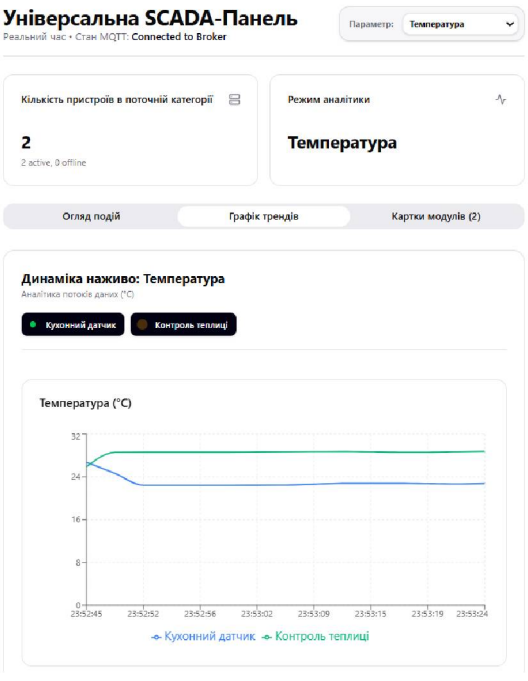



Рисунок 2.6 – Адаптивне вертикальне відображення інтерфейсу на екрані планшетного пристрою.

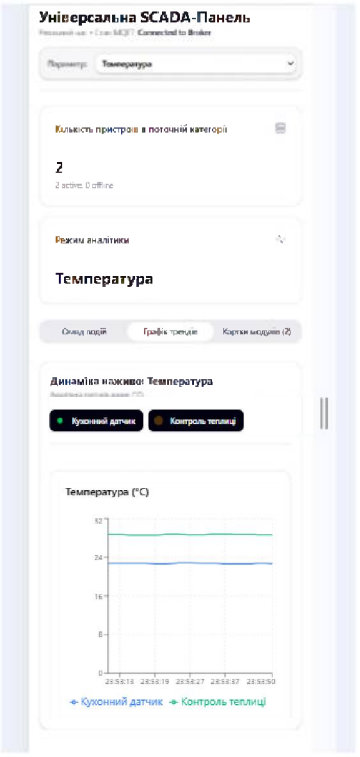


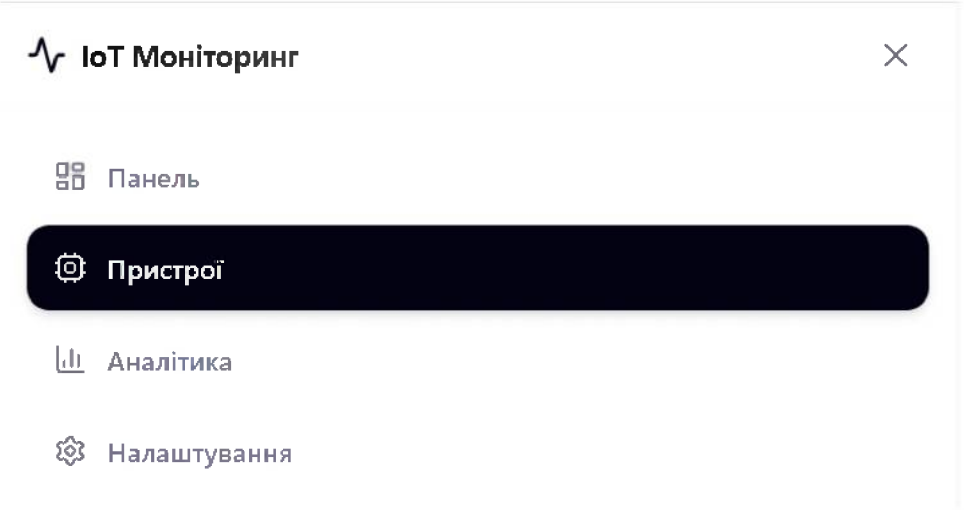

Рисунок 2.7 – Адаптивне вертикальне відображення інтерфейсу на екрані мобільного пристрою.

Важливою вимогою до сучасних SCADA та IoT-систем є підтримка адаптивного дизайну. Розроблена веб-панель оптимізована для роботи на персональних комп'ютерах, планшетах і мобільних пристроях, що зображено на рисунках 2.2, 2.6 та 2.7.

На великих екранах інтерфейс відображає одночасно кілька карток пристроїв, графіки та журнал подій у багатоколонковому режимі. На планшетах структура автоматично перебудовується у двоколонковий формат, а на мобільних пристроях усі компоненти розташовуються вертикально для забезпечення коректного сенсорного керування.

Під час адаптації графічних компонентів особливу увагу приділено роботі бібліотеки Recharts на малих екранах. Для запобігання накладанню елементів інтерфейсу графіки ізольовано у контейнерах із фіксованою мінімальною висотою, що гарантує коректне відображення шкал, підписів та легенд навіть на смартфонах.

Навігаційна система веб-панелі також адаптується до типу пристрою. На мобільних екранах горизонтальне меню автоматично замінюється кнопкою, яка відкриває повноекранну навігаційну панель (див. рис. 2.8). Це дозволяє ефективно використовувати обмежений простір дисплея та забезпечує зручність взаємодії з інтерфейсом.




пристрою.

Таким чином, розроблений користувацький інтерфейс поєднує властивості адаптивності, високої швидкодії, реактивності та інформаційної ефективності. Реалізована архітектура веб-панелі забезпечує стабільне відображення телеметричних даних у режимі реального часу, підтримує оперативне реагування на аварійні ситуації та створює зручне середовище для моніторингу IoT-пристроїв незалежно від типу використовуваного клієнтського пристрою.

## 2.5 Забезпечення інформаційної безпеки та відмовостійкості системи

Під час проєктування універсальної системи диспетчеризації та моніторингу IoT-пристроїв значна увага приділяється захисту інформаційних потоків між периферійними пристроями, MQTT-брокером та веб-застосунком. Необхідність реалізації комплексної моделі безпеки обумовлена вимогами до систем реального часу, для яких критичними є швидкодія, стабільність передачі телеметрії та захист від несанкціонованого доступу.

Архітектура безпеки побудована за багаторівневим принципом і охоплює клієнтський рівень, що відповідає за автентифікацію та контроль доступу користувачів, а також транспортний рівень MQTT-взаємодії, де реалізовано ізоляцію потоків даних і контроль прав публікації та підписки.

Для автентифікації користувачів передбачається використання сервісу Firebase Authentication, який працює на основі цифрових токенів доступу. Контроль доступу до функціональних модулів вебзастосунку реалізується через механізм захищених маршрутів, відповідно до якого доступ до внутрішніх сторінок надається лише після успішної перевірки токена. За відсутності або недійсності токена система блокує доступ до захищених компонентів інтерфейсу.

Для захисту каналів передавання телеметричних даних на стороні MQTT-брокера використовується система ACL-політик, що визначає права доступу різних типів клієнтів до тематичних каналів MQTT. Основним

принципом є розмежування прав публікації та підписки між пристроями збору даних і клієнтськими застосунками.

Периферійним вузлам на базі ESP32 надаються права лише на публікацію телеметрії у визначені канали, що унеможливлює доступ до інших потоків даних. Веб-застосунок взаємодіє з брокером через захищене WSS-з'єднання та має права виключно на читання повідомлень. Відсутність можливості публікації мінімізує ризик надсилання небажаних або деструктивних команд до IoT-пристроїв.

Запропонована архітектура безпеки забезпечує цілісність телеметричних даних, ізоляцію джерел інформації та зниження ризиків несанкціонованого втручання в роботу системи. Поєднання механізмів автентифікації, контролю доступу та ACL-політик MQTT-брокера формує комплексний підхід до захисту інформаційних потоків в IoT-інфраструктурі реального часу.

**Висновок до розділу 2**

У другому розділі було виконано проєктування веб-панелі моніторингу IoT-пристроїв у режимі реального часу та сформовано архітектурні принципи побудови системи. У процесі проєктування визначено основні вимоги до системи, серед яких ключовими є мінімальна затримка передачі телеметричних даних, масштабованість, відмовостійкість, адаптивність інтерфейсу та підтримка одночасної роботи великої кількості клієнтів.

У межах розділу розроблено структурну схему взаємодії компонентів системи та обґрунтовано використання багаторівневої архітектури з розподілом функцій між рівнем збору телеметрії, транспортним рівнем та рівнем представлення інформації. Встановлено, що застосування архітектурного патерна «Видавець-підписник» у поєднанні з протоколом MQTT забезпечує ефективну асинхронну передачу даних, низькі затримки та зменшення мережевого навантаження порівняно з класичними HTTP-підходами.

Проведено обґрунтування вибору технологічного стеку системи. Визначено, що використання React, TypeScript та Vite дозволяє реалізувати високопродуктивний frontend-застосунок із реактивним оновленням інтерфейсу,

компонентною архітектурою та підтримкою потокової обробки телеметричних даних. Застосування MQTT over WebSockets забезпечує сумісність із веб-браузерами та дозволяє організувати постійний двосторонній канал зв'язку між MQTT-брокером і клієнтським застосунком.

У процесі проєктування бази даних було сформовано структуру збереження поточного стану пристроїв та історичних телеметричних записів. Використання NoSQL-платформи Firebase дозволило забезпечити автоматичну синхронізацію даних у режимі реального часу, підтримку асинхронної взаємодії та підвищення відмовостійкості системи при короткочасних втратах з'єднання.

Окрему увагу приділено проєктуванню користувацького інтерфейсу веб-панелі. Реалізовано концепцію адаптивного Dashboard-інтерфейсу з підтримкою динамічних інформаційних віджетів, графіків часових рядів, журналу подій та системи кольорової індикації станів IoT-пристроїв. Запропонований підхід забезпечує зручність візуального аналізу телеметричної інформації та оперативне реагування на критичні події.

Також у межах розділу спроєктовано систему безпеки, яка включає механізми автентифікації користувачів, захист транспортного рівня за допомогою протоколу WSS та систему ACL-політик MQTT-брокера для розмежування прав доступу між IoT-пристроями та клієнтськими застосунками.

Таким чином, результати проєктування підтверджують можливість створення масштабованої, реактивної та високопродуктивної системи моніторингу IoT-пристроїв, здатної забезпечити стабільну обробку телеметричних потоків у режимі реального часу та ефективну взаємодію між усіма компонентами програмного комплексу.

# РОЗДІЛ 3
# ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ

## 3.1 Реалізація програмної моделі IoT-вузла на базі ESP32

Практична реалізація периферійного рівня системи збору телеметричних даних була виконана із використанням спеціалізованого середовища онлайн-емуляції Wokwi. Застосування даної платформи дозволило змоделювати роботу IoT-пристрою, виконати налаштування мережевої взаємодії та перевірити коректність функціонування програмного забезпечення без використання фізичної апаратної бази на етапі створення прототипу системи.

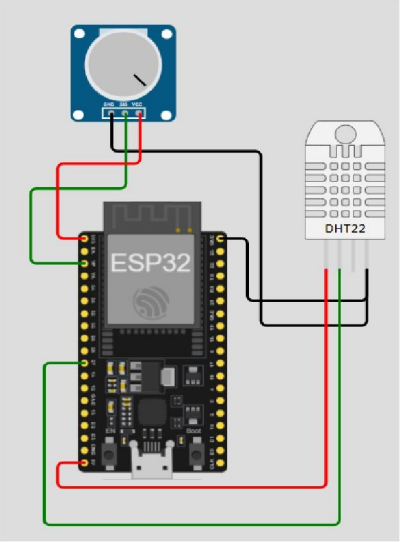



Рисунок 3.1. – Схема підключення датчика DHT22 до мікроконтролера ESP32 у Wokwi.

Основою віртуального вузла збору даних став мікроконтролер ESP32, який оснащений інтегрованим модулем бездротового зв'язку Wi-Fi та широко використовується у системах Інтернету речей завдяки високій продуктивності та підтримці сучасних мережевих протоколів. Для отримання телеметричних параметрів до мікроконтролера було підключено цифровий датчик температури та вологості DHT22. З'єднання сенсора з ESP32 реалізовано через цифровий

GPIO-інтерфейс, що забезпечило передачу даних між пристроями у межах моделі віртуальної схеми.

Схему підключення датчика DHT22 до мікроконтролера ESP32 у середовищі Wokwi наведено на рисунку 3.1.

Програмне забезпечення імітатора було розроблено мовою C++ із використанням Arduino Core для ESP32. Логіка роботи програми побудована за циклічним принципом, характерним для вбудованих систем реального часу. Після запуску пристрою у функції setup() виконувалася ініціалізація послідовного інтерфейсу для відладки, підключення драйвера датчика DHT22, а також налаштування бездротового з'єднання Wi-Fi для забезпечення доступу до мережевої інфраструктури.

У процесі роботи програми в основному циклі loop() здійснювалося періодичне зчитування температури та вологості із сенсора за допомогою функцій readTemperature() та readHumidity(). Отримані значення проходили перевірку на коректність шляхом аналізу результатів зчитування та контролю появи нечислових значень. У випадку успішного отримання валідних параметрів система переходила до етапу формування телеметричних пакетів для подальшої передачі через MQTT-мережу.

Таким чином, реалізований програмно-апаратний емулятор дозволив створити повноцінне джерело телеметричних даних для тестування роботи веб-панелі моніторингу та перевірки механізмів обробки інформації в режимі реального часу.

## 3.2 Реалізація MQTT-обміну

Транспортний рівень взаємодії між імітатором IoT-пристрою та клієнтським вебзастосунком був реалізований на основі протоколу MQTT, який є одним із найбільш поширених рішень для організації обміну повідомленнями в системах Інтернету речей. Використання MQTT забезпечило мінімальне навантаження на мережу, підтримку асинхронного обміну даними та можливість передачі телеметрії в режимі реального часу.

Для маршрутизації повідомлень було використано публічний хмарний MQTT-брокер, який забезпечував приймання, обробку та перенаправлення повідомлень між периферійним пристроєм та клієнтським застосунком. Підключення мікроконтролера до брокера реалізовано за допомогою бібліотеки PubSubClient, що забезпечує підтримку MQTT-протоколу для платформ Arduino та ESP32.

У програмному коді були визначені основні параметри мережевої взаємодії, зокрема адреса MQTT-брокера, порт підключення та унікальний ідентифікатор клієнта. Після встановлення з'єднання мікроконтролер виконував авторизацію на брокері та переходив у режим постійної передачі телеметричних даних.

З метою забезпечення стабільності роботи системи було реалізовано механізм автоматичного відновлення сесії у випадку втрати мережевого з'єднання. Для цього у програмі передбачено окрему функцію реконекту, яка циклічно перевіряла стан підключення та у разі його втрати повторно ініціювала процедуру підключення до MQTT-брокера. Такий підхід дозволив уникнути повного припинення роботи системи при короткочасних мережевих збоях.

Для організації потоків телеметричних даних було сформовано ієрархічну структуру тематичних каналів MQTT. Передача повідомлень від імітатора ESP32 здійснювалася у спеціалізований канал із використанням унікального ідентифікатора пристрою. Подібна структура дозволила забезпечити логічне розділення потоків даних та спростила масштабування системи при підключенні декількох IoT-вузлів.

Телеметричні дані перед передачею серіалізувалися у формат JSON із використанням бібліотеки ArduinoJson. До складу сформованого пакета включалися ідентифікатор пристрою, показники температури та вологості, а також часовий штамп формування повідомлення. Використання JSON забезпечило сумісність даних із вебтехнологіями та спростило подальшу обробку інформації на стороні React-застосунку.

Для забезпечення роботи системи в режимі реального часу та уникнення надмірного навантаження на мережу було встановлено часовий інтервал передачі телеметрії. Надсилання повідомлень до MQTT-брокера здійснювалося з періодичністю приблизно 2–3 секунди. Контроль часових інтервалів реалізовано із використанням системного таймера millis(), що дозволило уникнути застосування блокуючих затримок delay() та забезпечило безперервне функціонування мережевої взаємодії.

Приклад роботи системи, включно з процесом підключення до Wi-Fi-мережі, MQTT-брокера та циклічною передачею JSON-повідомлень, наведено на рисунку 3.2.

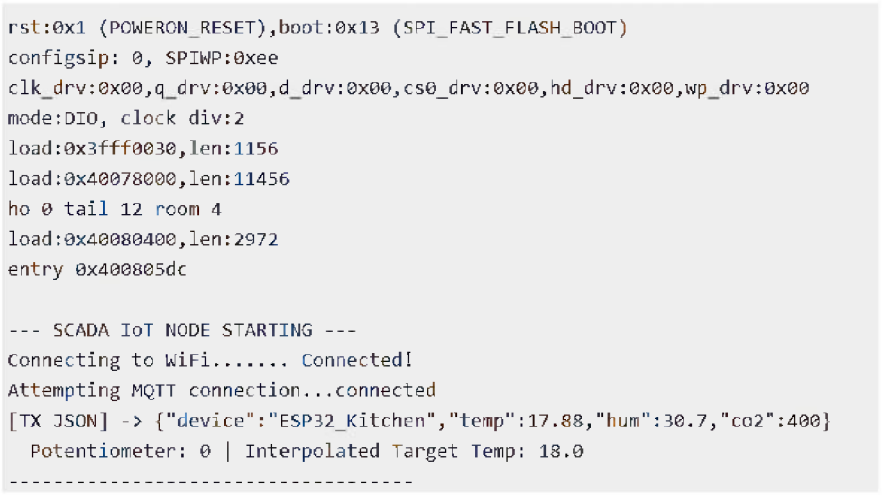



Рисунок 3.2 – Консоль налагодження Wokwi із підключення до Wi-Fi, MQTT та передачею JSON-повідомлень.

## 3.3 Реалізація frontend-частини веб-панелі

Прикладний рівень системи моніторингу було реалізовано у вигляді односторінкового веб-застосунку (SPA) із використанням бібліотеки React та мови програмування TypeScript. Застосування TypeScript дозволило забезпечити сувору типізацію структур телеметричних даних, підвищити надійність взаємодії між компонентами та зменшити кількість потенційних помилок, пов’язаних із некоректною обробкою асинхронних повідомлень.

Архітектура клієнтського застосунку була побудована за модульним принципом із розподілом функціональних компонентів відповідно до їх призначення. Центральним елементом керування станом та маршрутизацією виступав головний компонент застосунку, у межах якого реалізовано логіку взаємодії між окремими модулями системи. Для організації навігації між функціональними сторінками застосовано механізм декларативної маршрутизації на базі бібліотеки react-router. Такий підхід забезпечив можливість динамічного переходу між розділами веб-панелі без необхідності повного перезавантаження сторінки браузера.

Інтерфейс системи було організовано у вигляді спільного каркасного компонента, який інтегрував навігаційне меню та область відображення контенту. У межах вебпанелі реалізовано окремі функціональні сторінки для моніторингу телеметрії, перегляду інформації про пристрої, аналітичних модулів та системних налаштувань. Подібна структура забезпечила масштабованість застосунку та спростила подальше розширення функціональних можливостей системи.

Для відображення актуального стану IoT-пристроїв було розроблено спеціалізований компонент картки пристрою (DeviceCard), який отримував телеметричні параметри через механізм передачі властивостей. У межах компонента реалізовано механізм аналізу часової мітки останнього отриманого повідомлення, що дозволило визначати поточний мережевий статус пристрою. У випадку перевищення допустимого інтервалу очікування система автоматично змінювала візуальний індикатор стану пристрою та відображала повідомлення про втрату зв’язку (див. рис. 3.3).

Для підвищення зручності роботи з великою кількістю IoT-вузлів у верхній частині вебпанелі було реалізовано систему вкладок фільтрації, що дозволяла відображати всі пристрої, лише активні вузли або пристрої, які перебували у стані Offline. Перемикання вкладок виконувало динамічну фільтрацію масиву станів, забезпечуючи оперативний доступ до необхідної групи обладнання.

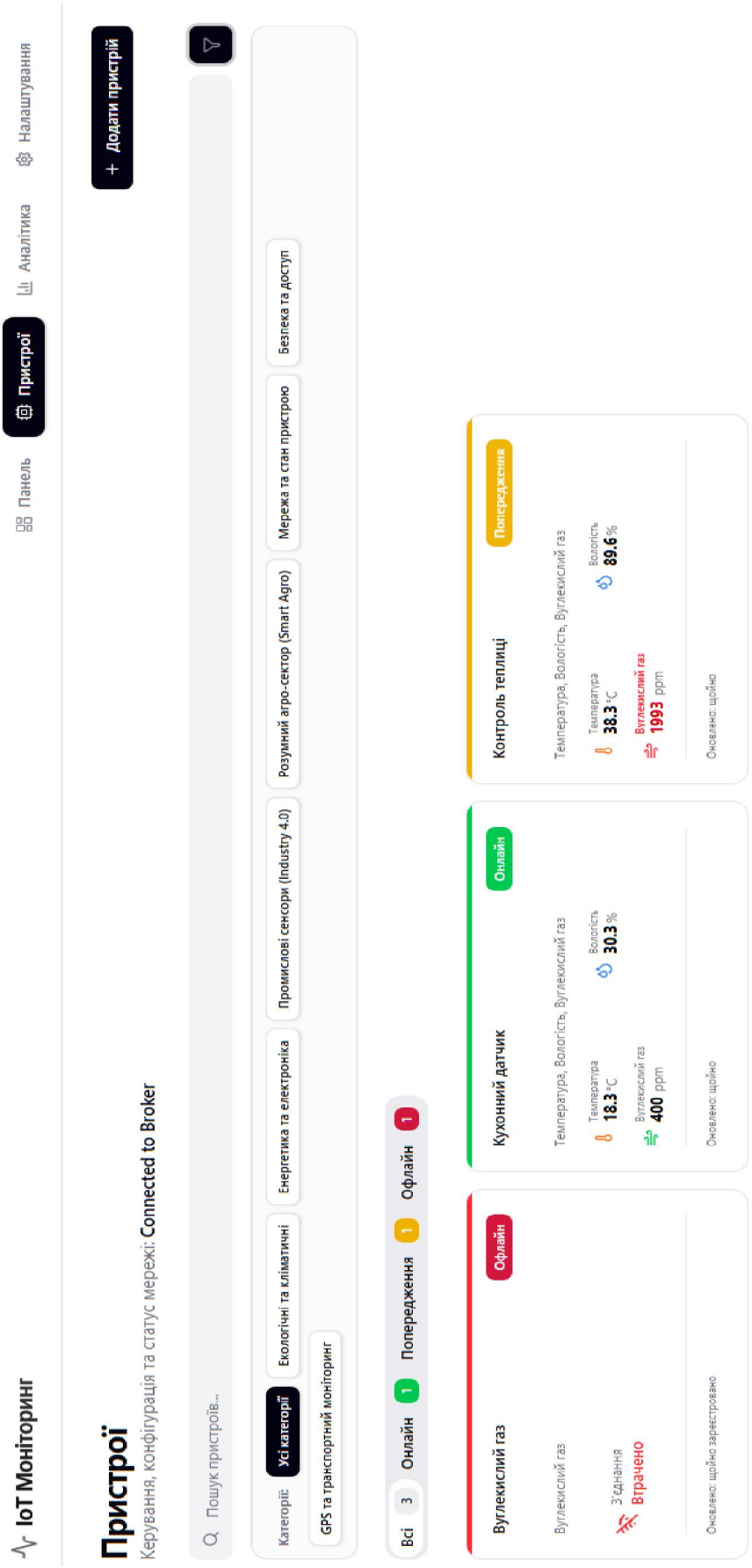



Рисунок 3.3 – Вікно пристроїв веб-панелі моніторингу IoT-пристроїв із фільтрами та індикаторами стану

Додатково в інтерфейс було інтегровано механізм текстового пошуку за ідентифікатором пристрою або його назвою. Пошук реалізовано засобами React через фільтрацію масиву станів у режимі реального часу. Такий підхід дозволив забезпечити швидке знаходження необхідного IoT-вузла без повторних запитів до сервера або бази даних.

Для візуалізації телеметричних показників у часовому вимірі використано бібліотеку Recharts, яка забезпечує побудову інтерактивних графіків у середовищі React. Графічний модуль отримував масив історичних даних та формував динамічні часові ряди для відображення температури, вологості та інших параметрів. Для побудови графіків використовувалися компоненти LineChart, XAxis, YAxis та Tooltip, що забезпечили формування осей координат, інтерактивних підказок та плавного відображення змін параметрів у часі.

При надходженні нових телеметричних повідомлень стан React-застосунку автоматично оновлювався, що викликало локальний перерендер графічного компонента. Завдяки цьому графіки динамічно зміщувалися та відображали дані в режимі реального часу.

## 3.4 Реалізація backend та збереження даних

Серверна частина системи та механізми довготривалого збереження інформації були реалізовані із використанням хмарної Serverless-платформи Firebase. Застосування Firebase дозволило відмовитися від розгортання та адміністрування окремого фізичного сервера, забезпечивши при цьому масштабовану інфраструктуру для роботи з телеметричними потоками в режимі реального часу.

Для організації зберігання даних було використано Firebase Realtime Database, який забезпечує підтримку NoSQL-структур та синхронізацію даних між клієнтами в реальному часі. Взаємодія між MQTT-брокером та хмарним сховищем реалізовувалася через окремий фоновий сервіс-обробник, основним завданням якого було приймання повідомлень із тематичних MQTT-каналів, перевірка їх валідності та запис у структуру бази даних Firebase.

Передавання інформації між компонентами системи виконувалося через WebSocket-з'єднання, що забезпечило мінімальні затримки доставки повідомлень та оперативну синхронізацію телеметрії між периферійними пристроями, базою даних і клієнтським веб-застосунком.

Структура сховища Firebase була організована у вигляді окремих функціональних вузлів. Один із вузлів використовувався для збереження поточного стану IoT-пристроїв та містив лише останній валідний пакет телеметрії від кожного мікроконтролера. Дані в цьому вузлі постійно оновлювалися, що дозволяло вебпанелі отримувати актуальну інформацію про стан пристроїв у режимі реального часу.

Окремий вузол бази даних використовувався для накопичення історії телеметричних показників. Усі нові пакети даних додавалися у сховище як окремі записи із часовими мітками, що забезпечило можливість подальшого аналізу історичних даних та побудови графіків динаміки змін параметрів.

Однією з ключових особливостей реалізованого бекенду стала підтримка механізму «живих слухачів» (Live Listeners). На відміну від традиційного підходу із циклічними HTTP-запитами, Firebase забезпечував автоматичне надсилання змінених даних до браузера користувача одразу після їх запису в базу. Це дозволило суттєво зменшити мережеве навантаження та забезпечити миттєве оновлення інтерфейсу без необхідності ручного перезавантаження сторінки.

Крім функцій збереження інформації, серверна частина виконувала базову обробку телеметричних потоків та контроль критичних станів системи. У випадку перевищення допустимих значень температури або втрати зв'язку з IoT-пристроєм система автоматично формувала подію тривоги. Згенеровані повідомлення записувалися у відповідний вузол бази даних та передавалися на frontend-частину застосунку, де відображалися у журналі подій SCADA-панелі у вигляді повідомлень різного рівня критичності.

Приклад структури бази даних Firebase та історичних записів телеметрії наведено на рисунку 3.4.




Рисунок 3.4 – Структура Firebase Database.

## 3.5 Візуалізація даних у реальному часі

Однією з ключових характеристик розробленої системи моніторингу є здатність забезпечувати відображення телеметричних даних у режимі реального часу з мінімальною затримкою між їх отриманням від периферійних пристроїв та візуалізацією на інтерфейсі користувача. Реалізація даного функціонального модуля базується на використанні потокової моделі обміну даними, що виключає необхідність ручного оновлення сторінки або використання класичних HTTP-запитів у циклічному режимі.

Інтеграція механізму реального часу була здійснена шляхом застосування MQTT-орієнтованого підходу з використанням WebSocket-з'єднання, що забезпечує постійний канал двосторонньої комунікації між клієнтським застосунком та брокером повідомлень. У межах React-застосунку обробка вхідних даних реалізована через підписку на тематичні канали, при цьому отримання повідомлень здійснюється асинхронно в межах життєвого циклу компонентів.

Після надходження нового телеметричного пакета від пристроїв, система виконує його десеріалізацію з формату JSON та оновлює внутрішній стан застосунку. Це призводить до автоматичного перерендерингу лише тих компонентів інтерфейсу, які залежать від змінених даних, що відповідає

принципам оптимізації React та дозволяє уникнути зайвого навантаження на користувацький інтерфейс.

Візуалізація часових рядів реалізована із використанням бібліотеки Recharts, яка забезпечує побудову інтерактивних лінійних графіків. Отримані телеметричні дані відображаються у вигляді динамічних кривих, що оновлюються у міру надходження нових значень. У межах одного графіка одночасно відображаються два незалежні потоки даних, що відповідають різним джерелам телеметрії, зокрема «Кухонний датчик» та «Контроль теплиці». Такий підхід забезпечує можливість порівняльного аналізу параметрів різних об’єктів у єдиному часовому просторі.

Кожне нове оновлення даних призводить до поступового зсуву історичних значень по осі часу, що створює ефект безперервної стрічки даних. Завдяки цьому оператор має можливість оперативно оцінювати динаміку змін параметрів мікроклімату та виявляти потенційні відхилення від нормальних значень без затримок у відображенні інформації.

Приклад реалізації динамічного оновлення графічного інтерфейсу під час одночасної роботи двох джерел телеметричних даних наведено на рисунку 3.5.

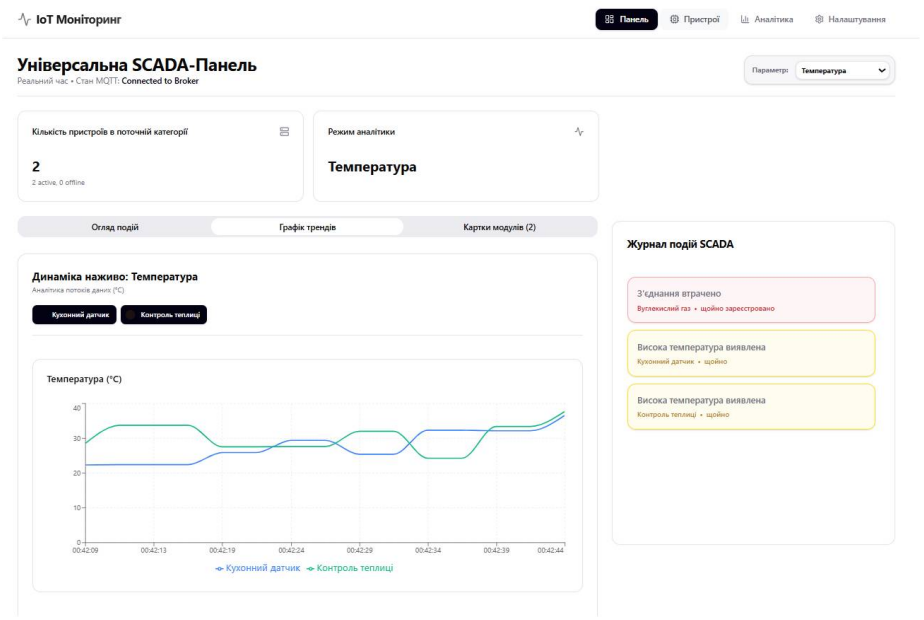



Рисунок 3.5 – Інтерфейс веб-панелі з графіком динаміки температури для двох джерел телеметрії.

З метою забезпечення універсальності та адаптивності розробленої SCADA-панелі на рівні клієнтського застосунку було реалізовано механізми підтримки різнотипних периферійних пристроїв і телеметричних потоків. Архітектура frontend-частини проєктувалася з урахуванням можливості підключення сенсорів різного функціонального призначення без необхідності суттєвої модифікації програмного коду. Для цього в інтерфейсі застосунку реалізовано гнучку модель обробки JSON-пакетів, що дозволяє автоматично інтерпретувати отримані дані відповідно до типу підключеного пристрою.

У процесі роботи система підтримувала обробку кліматичних датчиків, газоаналізаторів та промислових сенсорів. Для кліматичних датчиків типу DHT22 забезпечувалося відображення температури та вологості із використанням графічних індикаторів і динамічних часових графіків. Показники температури інтерпретувалися у градусах Цельсія, а значення вологості – у відсотках відносної вологості повітря.

Для датчиків екологічного моніторингу, зокрема сенсорів типу MQ-135, система виконувала обробку значень концентрації газів та вуглекислого газу у форматі PPM. У межах компонента DeviceCard реалізовано механізм аналізу критичних значень, який автоматично змінював візуальний стан картки при перевищенні допустимих порогів концентрації шкідливих речовин.

Підтримка промислових сенсорів була реалізована шляхом динамічної адаптації графічних компонентів під різні інженерні одиниці вимірювання. Залежно від значення параметра deviceType, отриманого у складі JSON-пакета, система автоматично налаштовувала підписи осей координат та масштаби графіків бібліотеки Recharts. Завдяки цьому веб-панель могла коректно працювати із сенсорами тиску, вібрації, навантаження та іншими промисловими вимірювальними пристроями.

Приклад відображення аналітичних показників для різних типів периферійних сенсорів наведено на рисунку 3.6.

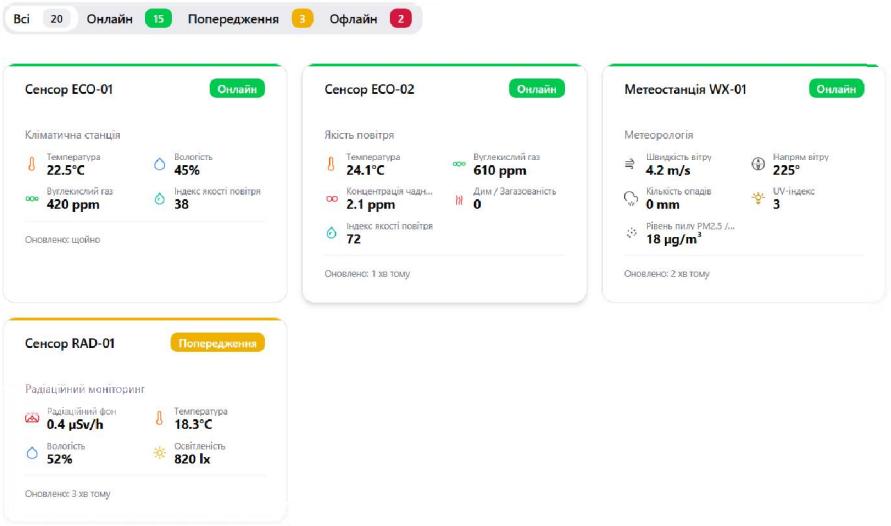



Рисунок 3.6 – Вікно моніторингу пристроїв: відображення різнотипних телеметричних метрик.

Для забезпечення масштабованості програмного комплексу та підтримки розширення мережі IoT-пристроїв у веб-панелі було реалізовано модуль динамічного додавання нових вузлів моніторингу. Реалізація даного функціоналу дозволила здійснювати інтеграцію нових пристроїв без внесення змін у вихідний код застосунку або перезапуску системи.

Модуль додавання пристроїв був реалізований у вигляді окремої форми конфігурації на сторінці керування пристроями. У процесі реєстрації нового IoT-вузла користувач вводив унікальний ідентифікатор пристрою, його назву, тип сенсора та тематичний MQTT-канал для приймання телеметричних повідомлень. На клієнтському рівні виконувалася перевірка введених даних на коректність, унікальність та відсутність недопустимих символів.

Після успішного проходження валідації застосунок за допомогою Firebase SDK виконував асинхронний запис нового пристрою у вузол devices_status хмарної бази даних. Завдяки використанню механізму Live Listeners застосунок автоматично отримував повідомлення про зміну структури бази даних та без затримок формував нову картку пристрою у загальному інтерфейсі моніторингу.

Одночасно із додаванням нового вузла клієнтський MQTT-модуль автоматично ініціював підписку на відповідний тематичний канал, що

забезпечувало негайне підключення нового джерела телеметрії до системи реального часу. Такий підхід дозволив забезпечити гнучкість архітектури та підтримку динамічного масштабування IoT-інфраструктури.

Приклад реалізації інтерфейсу додавання нового пристрою наведено на рисунку 3.7.

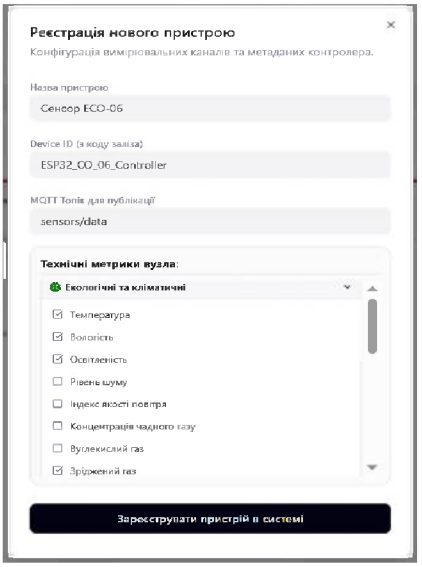



Рисунок 3.7. – Інтерфейс додавання нового пристрою до системи моніторингу

Для забезпечення можливості гнучкої адаптації системи під різні умови експлуатації у структурі вебзастосунку було реалізовано окремий модуль системних налаштувань. Даний модуль забезпечував централізоване керування параметрами мережевої взаємодії, конфігурацією порогових значень та параметрами користувацького інтерфейсу.

У межах вкладки наявний функціональний блок, який забезпечує налаштування критичних порогових значень для контрольованих параметрів. Користувач має можливість задавати верхні та нижні межі допустимих показників температури, вологості або концентрації газів за допомогою інтерактивних елементів керування. Встановлені значення зберігалися у локальному стані застосунку та синхронізувалися з Firebase. Надалі саме ці параметри використовувалися модулями аналізу телеметрії для автоматичного

формування аварійних та попереджувальних повідомлень у журналі подій SCADA-панелі.

Приклад реалізації сторінки системних налаштувань наведено на рис. 3.8.

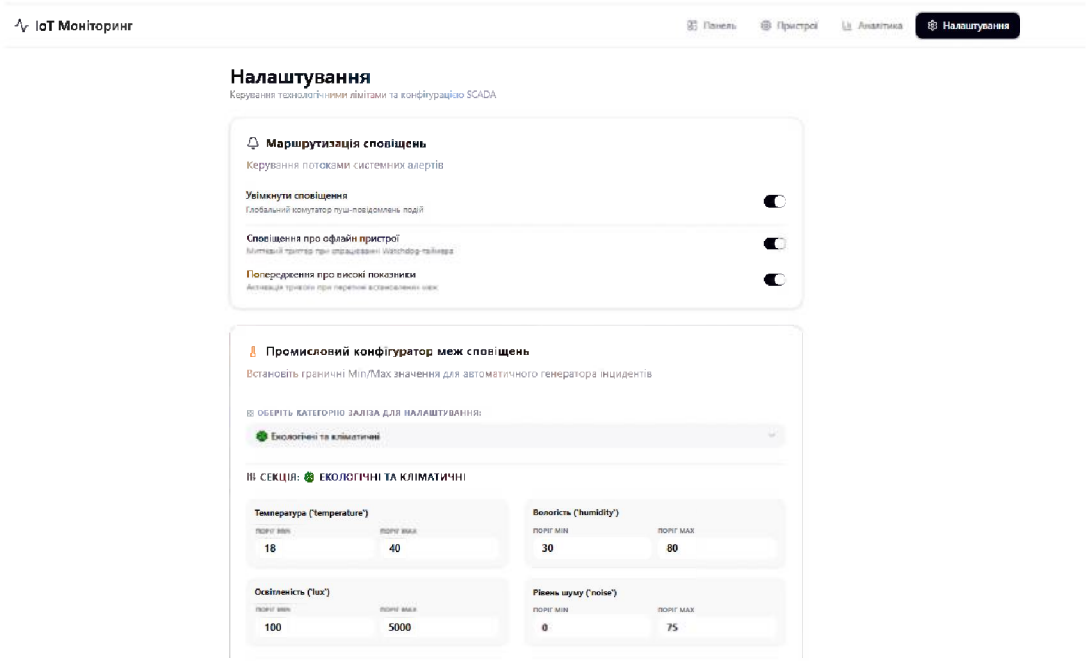



Рисунок 3.8 – Сторінка системних налаштувань веб-панелі моніторингу

## 3.6 Тестування та оцінювання продуктивності системи

Завершальним етапом розробки програмного комплексу стало комплексне тестування, спрямоване на перевірку коректності функціонування всіх компонентів системи, оцінку її продуктивності, стабільності та відмовостійкості. Тестування охоплювало як мережевий рівень передачі даних, так і користувацький інтерфейс вебзастосунку.

У процесі перевірки швидкодії системи було проведено вимірювання затримки передачі телеметричних даних від моменту їх генерації на рівні імітатора IoT-пристрою до моменту відображення на графічному інтерфейсі користувача. Аналіз здійснювався шляхом порівняння часових міток, зафіксованих у середовищі емуляції та інструментах розробника браузера. Отримані результати показали, що середній час скрізної передачі даних у

системі становить від 0,3 до 0,6 секунди, що відповідає вимогам до систем реального часу.

Навантажувальне тестування проводилося шляхом моделювання безперервної роботи пристроїв із частотою генерації телеметрії приблизно один пакет за секунду. У ході експерименту система демонструвала стабільну роботу без втрати пакетів даних та без деградації продуктивності клієнтської частини застосунку, що підтверджує її придатність до тривалої експлуатації в умовах постійного потоку інформації.

Окрему увагу було приділено перевірці поведінки системи у випадку виникнення критичних ситуацій. Зокрема, моделювалися сценарії втрати зв'язку з пристроєм та виходу контрольованих параметрів за межі допустимих значень. У випадку порушення мережевої взаємодії система фіксувала розрив з'єднання та автоматично формувала відповідне сповіщення, яке відображалося у журналі подій інтерфейсу. При перевищенні температурних порогів система генерувала попереджувальні повідомлення, що дозволяло оперативно інформувати користувача про потенційно небезпечні стани.

Додатково було проведено тестування користувацького інтерфейсу, яке включало перевірку працездатності елементів керування, інтерактивних підказок графіків, механізмів пошуку та фільтрації даних. Також було підтверджено коректну адаптивність інтерфейсу при зміні розмірів екрана, що забезпечує його стабільне відображення на різних типах пристроїв.

Отримані результати тестування підтверджують коректність реалізації програмного комплексу, його стабільність, відмовостійкість та відповідність поставленим функціональним вимогам. Приклад відображення результатів тестування у журналі подій системи наведено на рисунку 3.9.




Рисунок 3.9 – Журнал подій SCADA з відображенням критичних та попереджувальних повідомлень.

### 3.7. Оцінювання затримок передачі телеметричних даних

Для оцінювання швидкодії розробленої системи було виміряно часові затримки на основних етапах передачі телеметричних даних від IoT-пристрою до користувацького інтерфейсу. Передача інформації здійснюється за багаторівневою схемою, яка включає надсилання даних від мікроконтролера до MQTT-брокера, їх подальшу маршрутизацію через WebSocket-канал та відображення у вебінтерфейсі. Результати вимірювань наведено в таблиці 3.1.

Найбільша частина часу припадає на передачу даних від мікроконтролера ESP32 до MQTT-брокера. На цьому етапі затримка становить у середньому близько 120 мс, що пов’язано з передаванням пакета через бездротову мережу Wi-Fi, маршрутизацією через мережу Інтернет та обробкою повідомлення на стороні брокера.

Наступним етапом є передача даних від MQTT-брокера до WebSocket-шлюзу, яка в середньому займає близько 80 мс. Даний час включає обробку MQTT-повідомлення, визначення активних підписників, обробку структури JSON-пакета та його подальшу передачу через WebSocket-з’єднання.

Найменша затримка спостерігається на етапі доставки даних до клієнтського застосунку та їх відображення в інтерфейсі користувача. Середній час доставки даних до React-застосунку та оновлення елементів інтерфейсу становить близько 50 мс.

Сумарна наскрізна затримка передачі телеметричних даних від моменту їх формування мікроконтролером до відображення у вебпанелі складає приблизно 250 мс. Такий показник свідчить про можливість функціонування системи в режимі, близькому до реального часу, та забезпечує своєчасне відображення стану IoT-пристроїв і оперативне реагування користувача на зміну контрольованих параметрів.

Таблиця 3.1.

Часові характеристики етапів маршрутизації даних

| **Етап** | **Час** |
|---|---|
| ESP32 → MQTT | 123 мс |
| MQTT → WebSocket | 79 мс |
| WebSocket → React | 50 мс |
| Загальна затримка | 252 мс |

**Висновок до розділу 3**

У межах третього розділу було виконано повний цикл програмної реалізації та тестування веб-панелі моніторингу IoT-пристроїв, що включає розробку периферійного рівня, організацію мережевої взаємодії, створення клієнтського інтерфейсу, інтеграцію хмарного сховища даних та перевірку працездатності системи в умовах, наближених до реального часу.

На етапі реалізації імітатора IoT-пристрою було змодельовано роботу фізичного вузла збору телеметрії із використанням середовища Wokwi та мікроконтролера ESP32 із підключеним датчиком DHT22. Це дозволило відтворити процес отримання температурних і вологісних показників та сформувати основу для подальшої передачі даних у мережу.

У процесі реалізації MQTT-обміну було організовано транспортний рівень взаємодії між пристроєм та веб-застосунком із використанням хмарного MQTT-брокера. Забезпечено стабільну передачу телеметричних даних у форматі JSON, реалізовано механізм автоматичного відновлення з’єднання, а також сформовано ієрархічну структуру тематичних каналів, що дозволило забезпечити масштабованість системи та логічне розділення потоків даних.

Розробка frontend-частини вебпанелі здійснювалася із застосуванням React та TypeScript, що забезпечило модульність архітектури, типізацію даних та зручність подальшого розширення функціоналу. Було реалізовано систему компонентів для відображення стану IoT-пристроїв, механізми фільтрації та пошуку, а також інтерактивну візуалізацію телеметричних даних із використанням бібліотеки Recharts. Це дозволило забезпечити відображення інформації у вигляді динамічних графіків у режимі реального часу.

Серверна частина системи, побудована на базі Firebase, забезпечила довготривале зберігання телеметричних даних та їх синхронізацію між пристроями і клієнтським застосунком. Використання моделі Live Listeners дозволило реалізувати миттєве оновлення інтерфейсу без необхідності періодичних запитів, що підвищило ефективність використання мережевих ресурсів. Додатково реалізовано механізм обробки критичних подій та формування системних сповіщень.

У межах реалізації модуля візуалізації даних було забезпечено потокове оновлення графіків із мінімальною затримкою, що дозволило досягти ефекту безперервного моніторингу стану об’єктів. Порівняльне відображення кількох джерел телеметрії дало змогу здійснювати оперативний аналіз змін параметрів у єдиному часовому просторі.

На завершальному етапі було проведено комплексне тестування системи, яке включало перевірку швидкодії, стабільності, відмовостійкості та коректності відображення користувацького інтерфейсу. Отримані результати засвідчили, що середній час затримки передачі даних перебуває в межах, прийнятних для систем реального часу, а також підтвердили стабільну роботу системи при тривалому навантаженні та відсутність втрати телеметричних пакетів.

Загалом результати, отримані в межах третього розділу, підтверджують коректність реалізації запропонованої архітектури, її практичну працездатність та відповідність поставленим вимогам до системи моніторингу IoT-пристроїв у режимі реального часу.

# ВИСНОВКИ

У даній кваліфікаційній роботі вирішено актуальну науково-практичну задачу, що полягає у проєктуванні, програмній реалізації та комплексному тестуванні універсальної веб-панелі для моніторингу телеметричних даних з периферійних пристроїв Інтернету речей у режимі реального часу. Під час виконання дослідження було послідовно реалізовано всі поставлені завдання, а архітектурні й програмні рішення повністю підтвердили свою ефективність та відповідність жорстким критеріям, що висуваються до сучасних систем диспетчеризації та збору даних (SCADA).

Проведений аналіз наявних теоретичних джерел та сучасних ринкових аналогів (ThingsBoard, Grafana, Node-RED) дозволив виявити їхні ключові архітектурні обмеження, пов'язані з високою ресурсомісткістю, жорсткими рамками кастомізації інтерфейсу оператора та затримками під час обробки високоінтенсивних потоків даних. Це обґрунтувало доцільність створення власної гнучкої системи, побудованої на базі розподіленої трирівневої моделі з використанням передових веб-технологій та спеціалізованих мережевих протоколів.

У ході архітектурного проєктування та програмної розробки було успішно реалізовано концепцію слабкої зв'язаності компонентів системи. Використання асинхронного протоколу MQTT на транспортному рівні у поєднанні з технологією WebSockets дозволило подолати технологічний бар'єр веб-браузерів, які не мають нативної підтримки чистих TCP-сокетів. Пряма потокова трансляція телеметрії безпосередньо від брокера повідомлень до клієнтського застосунку мінімізувала мережеві накладні витрати та навантаження на серверну інфраструктуру, усунувши необхідність у неефективному HTTP-опитуванні сервера.

Розробка клієнтської частини на базі бібліотеки React та мови TypeScript забезпечила високу реактивність, компонентну ізоляцію та строгу типізацію інтерфейсу користувача. Завдяки застосуванню Virtual DOM оновлення

віджетів, графіків часових рядів та системних журналів подій здійснюється локально, виключаючи повний перерендер сторінок додатка. Інтеграція хмарної NoSQL-платформи Firebase Realtime Database та механізму Live Listeners дозволила організувати ефективне розділене зберігання поточних статусів пристроїв і ретроспективних часових рядів із забезпеченням автоматичної синхронізації в реальному часі та високого рівня відмовостійкості інтерфейсу за умов короткочасних мережевих збоїв.

Особливу наукову й практичну цінність має розроблена універсальна модель парсингу даних. Веб-панель продемонструвала здатність динамічно адаптуватися до гетерогенного периферійного заліза, успішно обробляючи та візуалізуючи в єдиному інформаційному просторі різнотипні інженерні метрики: екологічні параметри, метеорологічні показники та рівні радіаційного фону тощо. Створені модулі динамічного додавання пристроїв, конфігурації критичних порогів за допомогою гнучкого управління системою сповіщень суттєво розширили експлуатаційний потенціал системи, забезпечивши її безперешкодне горизонтальне масштабування без зупинки чи модифікації вихідного коду сервісів.

Експериментальна перевірка та комплексне тестування розробленого програмного комплексу (із залученням програмних імітаторів мікроконтролерів ESP32 у середовищі Wokwi) підтвердили високу надійність та швидкодію системи. Показники скрізної затримки передачі пакетів JSON від моменту фіксації фізичної події датчиком DHT22 до графічного рендерингу на екрані оператора склали 0,3-0,6 секунди, що є відмінним результатом для систем реального часу. Реалізований адаптивний дизайн UI/UX підтвердив зручність та стабільність відображення аналітики на різних класах пристроїв, включаючи мобільні платформи та планшети. Таким чином, результати роботи засвідчили успішне виконання інженерної мети, а розроблена веб-панель може бути рекомендована до впровадження у реальних проєктах автоматизації будівель, екологічного моніторингу та промислового Інтернету речей.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Проєктування та розробка веб-орієнтованих систем моніторингу [Електронний ресурс] // Електронний репозитарій УжНУ (Ужгородський національний університет). – Режим доступу: https://dspace.uzhnu.edu.ua/server/api/core/bitstreams/4d75a531-1af6-4dcc-8fe4-7a85f317cc23/content.
2. Internet of Things (IoT) [Електронний ресурс] // TechTarget IoT Agenda. – 2025. – Режим доступу: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT.
3. MQTT vs CoAP vs HTTP vs WebSocket for IoT [Електронний ресурс] // AgileSoft Labs Blog. – 2026. – Режим доступу: https://www.agilesoftlabs.com/blog/2026/04/mqtt-vs-coap-vs-http-vs-websocket-iot.
4. ThingsBoard – Open-source IoT Platform [Електронний ресурс]. – 2026. – Режим доступу: https://thingsboard.io/.
5. Grafana: The open observability platform [Електронний ресурс]. – 2026. – Режим доступу: https://grafana.com/.
6. Node-RED: Low-code programming for event-driven applications [Електронний ресурс]. – 2026. – Режим доступу: https://nodered.org/.
7. ESP8266 [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/ESP8266.
8. ESP32 [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/ESP32.
9. Raspberry Pi [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Raspberry_Pi.
10. Firebase: Google's mobile and web application development platform [Електронний ресурс]. – 2026. – Режим доступу: https://firebase.google.com/.

11. Comparative Performance Analysis Between the MQTT and WebSocket Protocols / M. F. Tsaqief, J. Sutopo // Jurnal Buletin Tambun. – 2023. – Режим доступу: https://jurnal.kdi.or.id/index.php/bt/article/view/3223/1701.
12. Internet of Things and Nodemcu: A review of use of Nodemcu ESP8266 in IoT products [Електронний ресурс] // ResearchGate. – 2019. – Режим доступу: https://www.researchgate.net/publication/337656615_Internet_of_Things_and_Nodemcu_A_review_of_use_of_Nodemcu_ESP8266_in_IoT_products.
13. System Design: Client-Server Model [Електронний ресурс] // GeeksforGeeks. – 2025. – Режим доступу: https://www.geeksforgeeks.org/system-design/client-server-model/.
14. Croll A. Complete Web Monitoring / A. Croll, S. Power. – O'Reilly Media, 2009. – 716 с.
15. WebSocket Protocol Comparisons [Електронний ресурс] // WebSocket.org. – 2024. – Режим доступу: https://websocket.org/comparisons/.
16. IoT Protocols: HTTP/2, WebSocket, MQTT, Zigbee, Matter, and Thread [Електронний ресурс] // Medium (Code Factory Berlin). – 2024. – Режим доступу: https://medium.com/code-factory-berlin/iot-protocols-http-2-websocket-mqtt-zigbee-matter-and-thread-bf93eef3fb0c.
17. Тренди та сфери застосування Internet of Things (IoT) [Електронний ресурс] // Блог Імена.UA. – 2023. – Режим доступу: https://www.imena.ua/blog/top-10-scope-iot/.
18. Технології Інтернету речей в управлінні пристроями на мікроконтролерах : навч. посіб. [Електронний ресурс] / І. Ш. Невлюдов, В. А. Андрусевич, С. П. Новоселов, О. Г. Резніченко. – Електрон. текст. дані (1 файл : 2,88 Mb). – Харків : ХНУРЕ, 2023. – 214 с. – Режим доступу: електронне видання кафедри ТАПР ХНУРЕ.

# ДОДАТКИ

## Додаток А

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

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ

(підпис)
«13» жовтня 2025року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

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

**ПЛАХОТІ ОЛЕНИ ОЛЕКСАНДРІВНИ**
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **ВЕБ-ПАНЕЛЬ МОНІТОРИНГУ ІОТ-ПРИСТРОЇВ У РЕАЛЬНОМУ ЧАСІ**
керівник роботи **Мороз Ольга Юріївна, PhD з комп.наук, доцент зво кафедри,**
(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) телеметричні параметри IoT-пристроїв: температура, вологість, статус підключення;
   2) протоколи передачі даних: MQTT, WebSockets;
   3) вимоги до веб-застосунку для моніторингу та візуалізації телеметрії;
   4) персональний комп'ютер для розробки та тестування системи;
   5) критерії ефективності: затримка передачі даних, стабільність з'єднання, коректність візуалізації;
   6) обмеження щодо обсягу даних.
4. Перелік питань, які потрібно розробити:
   1) Проаналізувати принципи побудови IoT-систем, способи передачі телеметрії та веб-архітектуру реального часу.
   2) Дослідити існуючі програмні рішення для моніторингу IoT-пристроїв.
   3) Розробити концепцію та архітектуру веб-панелі моніторингу.
   4) Реалізувати інтерфейс веб-додатка з використанням сучасних front-end технологій.
   5) Забезпечити отримання та оновлення даних у реальному часі (REST, WebSockets або MQTT).
   6) Реалізувати модулі візуалізації телеметричних даних (графіки, індикатори, таблиці).
   7) Провести тестування, оцінити працездатність та ефективність розробленої системи.
   8) Підготувати висновки щодо застосування веб-панелі у практичних IoT-сценаріях.

4. План роботи

| № з/п | Назви етапів роботи | Термін виконання етапів роботи |
|---|---|---|
| 1. | Аналіз сучасних підходів до побудови систем моніторингу IoT-пристроїв у реальному часі | 12.10.2025 – 15.11.2025 |
| 2. | Огляд архітектурних рішень та протоколів передачі даних, аналіз методів обробки телеметричної інформації | 16.11.2025 – 14.01.2026 |
| 3. | Розробка концептуальної та структурної моделі автоматизованої системи моніторингу IoT-пристроїв | 15.01.2026 – 31.01.2026 |
| 4. | Проєктування та реалізація веб-панелі моніторингу (верстка інтерфейсу, створення структури сторінки) | 01.02.2026 – 14.02.2026 |
| 5. | Реалізація механізму імітації збору, передачі та візуалізації телеметричних даних | 15.02.2026 – 10.03.2026 |
| 6. | Тестування розробленої системи та перевірка коректності роботи функціоналу | 11.03.2026 – 24.03.2026 |
| 7. | Аналіз результатів тестування, визначення обмежень системи та можливостей її масштабування | 25.03.2026 – 14.04.2026 |
| 8. | Підготовка рекомендацій щодо вдосконалення системи та оформлення пояснювальної записки | 15.04.2026 – 30.04.2026 |
| 9. | Підготовка і оформлення звітних матеріалів. Оформлення списку літератури. | 01.04.2025 – 20.05.2025 |
| 10. | Написання статті за матеріалами кваліфікаційної роботи | 01.03.2025 – 20.05.2025 |
| 11. | Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог до звітів про НДР. | 01.05.2025 – 30.05.2025 |
| 12. | Оформлення звіту про переддипломну практику. | 21.05.2026 - 25.05.2026 |
| 13. | Представлення кваліфікаційної роботи керівнику та рецензенту | 26.05.2026 - 30.05.2026 |

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

Здобувач вищої освіти _____ Олена ПЛАХОТЯ
підпис — ініціали, прізвище

Керівник роботи _____ Ольга МОРОЗ
підпис — ініціали, прізвище

**Додаток Б**

## Технічне завдання
## на розробку програмного виробу
## «Веб-панель моніторингу IoT-пристроїв у реальному часі»

| Назва розділу | Назва і зміст підрозділу |
|---|---|
| 1. Введення | 1.1. Назва програмного вироби – Веб-панель моніторингу IoT-пристроїв у реальному часі.<br>1.2. Галузь застосування – промисловість. |
| 2. Підстава для розробки | 2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія.<br>2.2. Завдання на кваліфікаційну бакалаврську роботу |
| 3. Призначення розробки | 3.1. Мета розробки програмного вироби – підвищення ефективності моніторингу та контролю IoT-пристроїв за рахунок централізованої обробки даних, застосування адаптивної аналітики, автоматичного виявлення аномалій та забезпечення масштабованості системи<br>3.2. Призначення програмного виробу – забезпечення віддаленого контролю за станом обладнання та автоматизація процесу збору даних з IoT-датчиків через зручний веб-інтерфейс.<br>3.3. Початкові дані для розробки: опис обладнання та вимоги до програмного середовища. |
| 4. Технічні вимоги до програмного виробу | 4.1. Вимоги до функціональних характеристик:<br>1) збір та обробка даних<br>2) відображення списку підключених IoT-пристроїв та їх поточного стану<br>3) відображення основних телеметричних параметрів пристроїв<br>4) оновлення даних у режимі, наближеному до реального часу, без перезавантаження сторінки<br>5) візуалізація даних у вигляді графіків<br>4.2. Вимоги до надійності:<br>1) система повинна забезпечувати стабільну роботу при безперервному оновленні даних<br>2) у разі втрати з'єднання з IoT-пристроєм або сервером система повинна коректно відображати статус «OFFLINE» та не допускати аварійного завершення роботи<br>3) система повинна забезпечувати коректну обробку помилок передачі даних без втрати працездатності інтерфейсу<br>4) веб-панель повинна зберігати працездатність при багаторазовому оновленні телеметричних даних<br>4.3. Вимоги до умов експлуатації приміщення з доступом до мережі Інтернет, обладнані комп'ютерною технікою та стабільним електроживленням для роботи IoT-обладнання.<br>4.4. Вимоги до складу і параметрів технічних засобів Персональний комп'ютер у повній комплектації (ноутбук) |

| | |
|---|---|
| | 4.5. Вимоги до інформаційної та програмної сумісності<br>система повинна забезпечувати сумісність із сучасними веб-браузерами та використовувати стандартні формати й протоколи обміну даними для взаємодії з іншими програмними компонентами.<br>4.6. Вимоги до маркування та упаковки відсутні.<br>4.7. Вимоги до транспортування і зберігання відсутні.<br>4.8. Спеціальні вимоги не пред'являються. |
| 5. Вимоги до програмної документації. | Програмною документацією до виробу «Веб-панель моніторингу IoT-пристроїв у реальному часі» вважати:<br>1) Справжнє Технічне завдання на розробку програмного вироби (представити у вигляді Додатку Б до пояснювальної записки до дипломної роботи).<br>2) Програму і методику випробувань розробленого програмного вироби (представити у вигляді Додатку В до пояснювальної записки до дипломної роботи).<br>3) Опис програмного вироби (представити в розділі ___ пояснювальної записки до дипломної роботи).<br>4) Текст програми (представити в Додатку Г до пояснювальної записки до дипломної роботи). |
| 6. Техніко-економічні показники | В даному розділі можуть бути представлені:<br>1) орієнтовна оцінка ефективності розробленої системи визначається скороченням часу моніторингу стану IoT-пристроїв, підвищенням оперативності виявлення несправностей та зручністю централізованого контролю телеметричних даних через єдиний веб-інтерфейс.<br>2) Вимоги до розрахунку техніко-економічних показників не потрібні. |

**7. Стадії і етапи розробки**

| № з/п | Назви етапів роботи | Термін виконання етапів роботи |
|---|---|---|
| 1. | Аналіз сучасних підходів до побудови систем моніторингу IoT-пристроїв у реальному часі | 12.10.2025 – 15.11.2025 |
| 2. | Огляд архітектурних рішень та протоколів передачі даних, аналіз методів обробки телеметричної інформації | 16.11.2025 – 14.01.2026 |
| 3. | Розробка концептуальної та структурної моделі автоматизованої системи моніторингу IoT-пристроїв | 15.01.2026 – 31.01.2026 |
| 4. | Проєктування та реалізація веб-панелі моніторингу (верстка інтерфейсу, створення структури сторінки) | 01.02.2026 – 14.02.2026 |
| 5. | Реалізація механізму імітації збору, передачі та візуалізації телеметричних даних | 15.02.2026 – 10.03.2026 |
| 6. | Тестування розробленої системи та перевірка коректності роботи функціоналу | 11.03.2026 – 24.03.2026 |
| 7. | Аналіз результатів тестування, визначення обмежень системи та можливостей її масштабування | 25.03.2026 – 14.04.2026 |
| 8. | Підготовка рекомендацій щодо вдосконалення системи та оформлення пояснювальної записки | 15.04.2026 – 30.04.2026 |
| 9. | Підготовка і оформлення звітних матеріалів. Оформлення списку літератури. | 01.04.2025 – 20.05.2025 |
| 10. | Написання статті за матеріалами кваліфікаційної роботи | 01.03.2025 – 20.05.2025 |
| 11. | Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог до звітів про НДР. | 01.05.2025 – 30.05.2025 |
| 12. | Оформлення звіту про переддипломну практику. | 21.05.2026 - 25.05.2026 |
| 13. | Представлення кваліфікаційної роботи керівнику та рецензенту | 26.05.2026 - 30.05.2026 |

**8. Порядок контролю і приймання**

1) Перевірку ходу розроблення програмного виробу здійснювати керівником роботи не рідше одного разу на тиждень.
2) Завершену веб-панель моніторингу IoT-пристроїв подати у вигляді працюючої демонстраційної версії.
3) Пояснювальну записку та проектну документацію подати в електронному вигляді та на паперових носіях в одному примірнику.
4) Приймання роботи здійснюється за умови відповідності розробленої системи поставленим функціональним вимогам.
5) Остаточне приймання програмного виробу відбувається після успішного захисту роботи.

Виконавець
студент групи КІ-41
ПЛАХОТЯ О.О.
_____

Замовник
PH. D., доцент каф.КСР
МОРОЗ О.Ю.
_____

4. Вимоги до інформаційної та програмної сумісності:

Сумісність із сучасними веб-браузерами (Google Chrome, Mozilla Firefox, Edge) та підтримка роботи з серверними технологіями.

5. Вимоги до маркування та упаковки відсутні.

6. Вимоги до транспортування і зберігання відсутні.

7. Спеціальні вимоги не пред'являються.

**5. Вимоги до програмної документації**

Склад програмної документації, що подається на випробування, включає:

1) Технічне завдання на розробку програмного вироби (представлено в Додатку Б до пояснювальної записки до дипломної роботи).

2) Програма та методика випробувань (даний документ, Додаток В).

3) Опис програми (розділ 3 пояснювальної записки).

**6. Засоби і порядок випробувань**

**6.1.Засоби випробувань**

Технічні засоби: персональний комп'ютер (ноутбук) розробника.

Програмні засоби: сучасні веб-браузери (Google Chrome, Mozilla Firefox), інтегроване середовище розробки Visual Studio Code, локальне середовище збірки Vite, хмарна NoSQL база даних Firebase Realtime Database, хмарний MQTT-брокер.

Периферійне апаратне забезпечення: середовище емуляції Wokwi.

**6.2. Порядок проведення випробувань**

Етап 1. Перевірка програмної документації

Проведено аудит комплектності та якості оформлення документації. Встановлено, що склад і графічне виконання пояснювальної записки та додатків повністю відповідають вимогам ДСТУ та стандартам вищого навчального закладу. Наявні всі необхідні підписи, структурні розділи, схеми та лістинги коду.

Етап 2. Функціональні випробування програмного комплексу

**Функціональні тести**

Тест 1. Перевірка ініціалізації та запуску системи

Методика: Запуск локального інструменту збірки Vite в IDE Visual Studio Code, ініціалізація клієнтського застосунку React у браузері, відкриття інструментів розробника Chrome DevTools (вкладка Console та Network).

Очікуваний результат: Успішне розгортання SPA-застосунку, встановлення повнодуплексного WSS-з'єднання з хмарним MQTT-брокером через порт 443, відсутність Runtime-помилок типізації TypeScript у консолі.

Результат випробування: веб-панель моніторингу успішно завантажується у браузері, модулі навігації працюють справно відповідно до декларативної маршрутизації. Консоль відладки підтверджує успішну ініціалізацію хмарного Firebase SDK. Система перебуває в робочому стані.

Тест 2. Перевірка асинхронного збору даних та парсингу JSON

Методика: Запуск периферійних пристроїв та активація проміжного серверного шлюзу mqtt-bridge.js у середовищі виконання Node.js. Моніторинг потоку повідомлень у консолі розробника.

Очікуваний результат: серверний міст успішно підключається до MQTT-брокера, асинхронно перехоплює JSON-пакети від ідентифікованих пристроїв (ESP32_Wokwi, ESP32_CO2_Controller), автоматично додає мітку часу платформи Date.now() та виконує атомарний запис у NoSQL-гілки бази даних за допомогою методів ref() та push().

Результат випробування: зафіксовано безперебійний паралельний прийом об'єктів телеметрії від кількох гетерогенних пристроїв. Консоль відладки (рисунок В.1) документально підтверджує успішний парсинг метрик температури (temp), вологості (hum), концентрації вуглекислого газу (co2) та пилу (pm_data), а також виводить службове повідомлення про успішне збереження кожної транзакції у хмарі Firebase без втрати пакетів.

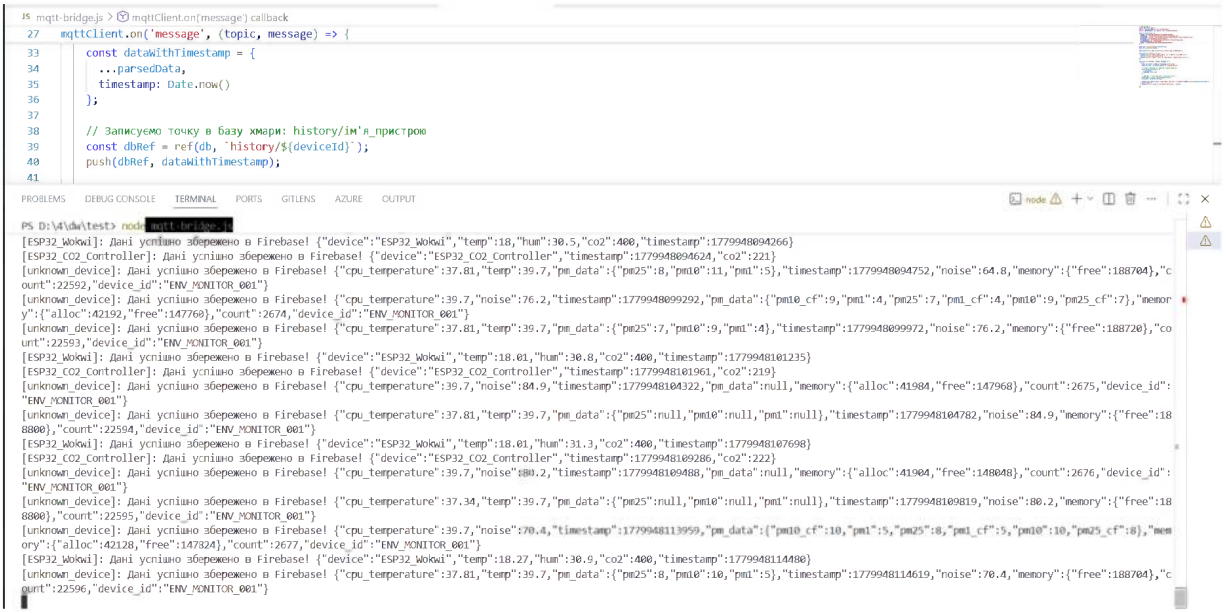



Рисунок В.1 – Консоль сервісу маршрутизації «mqtt-bridge.js» із логами успішного парсингу та синхронізації JSON-пакетів у базу даних Firebase.

Тест 3. Перевірка універсальності UI та візуалізації гетерогенних метрик

Методика: Подача на вхід веб-панелі через серверний шлюз потоків десеріалізованих даних від різнотипних клієнтських категорій пристроїв: екологічних, енергетичних, промислових сенсорів (Industry 4.0) та розумного агро-сектору. Візуальний контроль реактивності інтерфейсу.

Очікуваний результат: Компонент DeviceCard динамічно адаптує структуру внутрішніх полів, текстові назви та специфічні інженерні одиниці виміру під архітектуру конкретного підключеного заліза на основі NoSQL-конфігурації.

Результат випробування:

Експериментально підтверджено повну універсальність розробленого інтерфейсу. Система безпомилково відрендерила метрики радіаційного фону ($\mu$Sv/h), тиску в трубопроводах (Bar), швидкості обертів двигуна (RPM), споживання енергії (kWh) та кислотності ґрунту (pH). Клієнтські фільтри категорій (верхнє меню) та вкладки інтегральних статусів («Всі: 20», «Онлайн: 15», «Попередження: 3», «Офлайн: 2») працюють у режимі реального часу. Загальний вигляд панелі в момент комплексного збору гетерогенної телеметрії наведено на рисунку В.2.

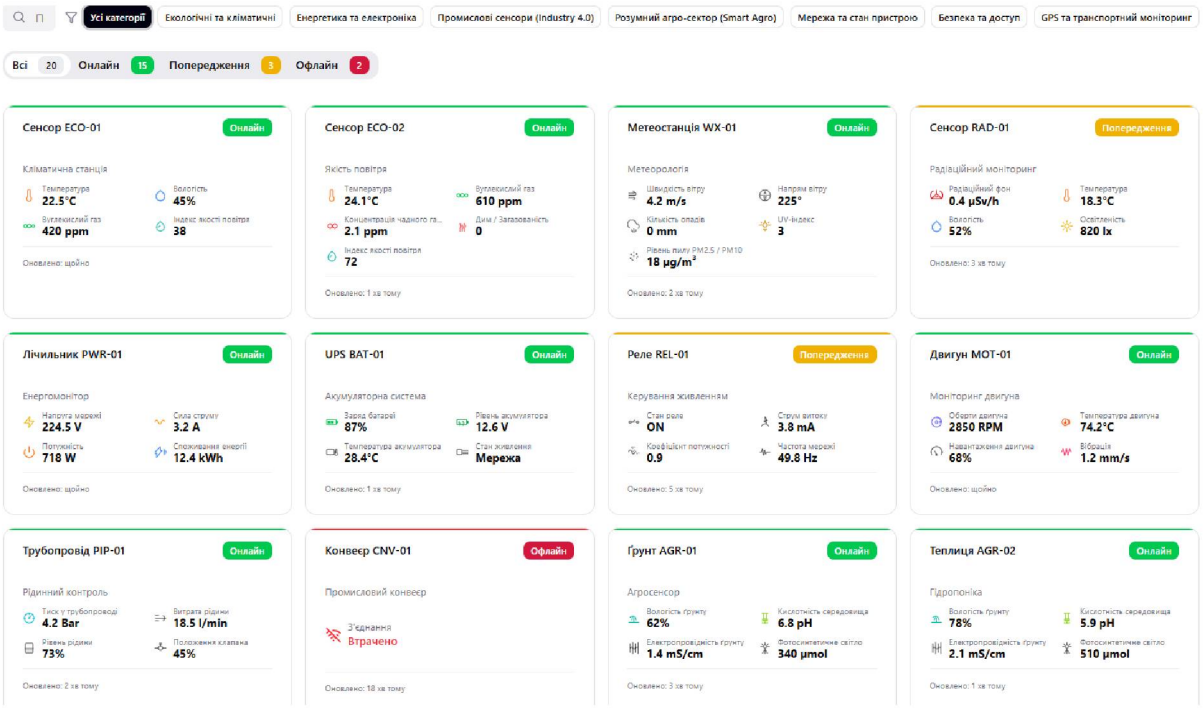



Рисунок В.2 – Головне вікно веб-панелі моніторингу в момент комплексної обробки гетерогенних потоків телеметрії.

Тест 4. Перевірка виявлення аномалій, відмовоїстійкості та Watchdog-контролю

Методика: Моделювання двох критичних сценаріїв:

1) штучне підняття повзунка температури у Wokwi понад 30°C ;

2) повна зупинка симуляції пристрою вуглекислого газу (імітація апаратного збою).

Очікуваний результат: при перевищенні лімітів система має згенерувати жовту картку попередження. При припиненні передачі пакетів Watchdog-таймер системи має автоматично перевести пристрій у статус «Офлайн» та сформувати червону картку тривоги (рисунок В.3). Всі аварії мають логуватися у Журналі подій SCADA.

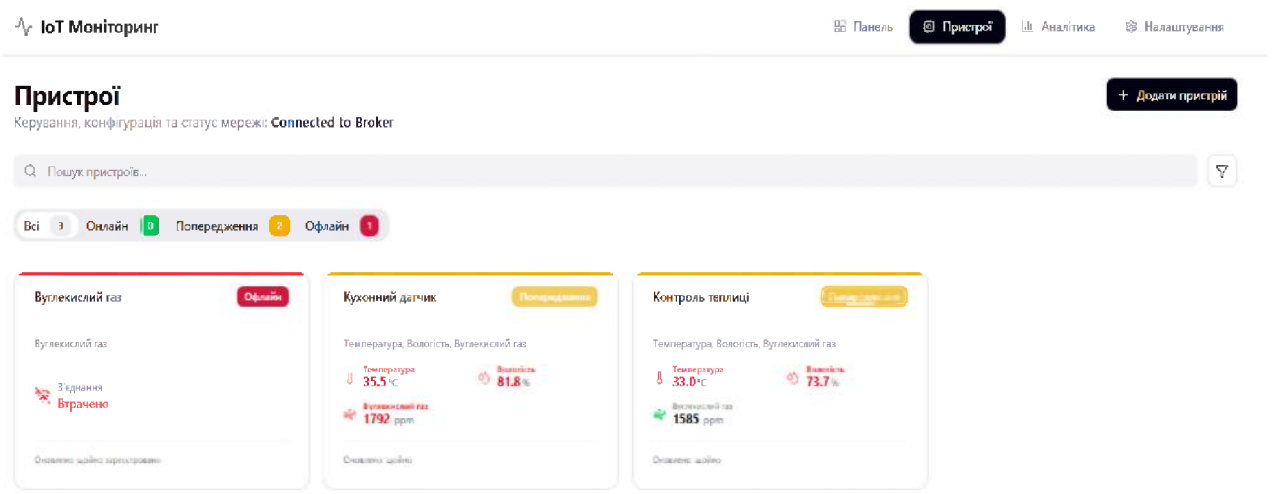



Рисунок В.3 – Вкладка пристроїв із відображенням критичних та попереджувальних повідомлень.

Результат випробування: алгоритми аналізу критичних станів та обробки помилок відпрацювали безпомилково. Веб-панель миттєво зреагувала на вихід параметрів за межі норми та на зникнення зв'язку з пристроєм, згенерувавши відповідні сповіщення. Локальний кеш забезпечив стабільність графіків, не допустивши збою UI. Фіксацію тривожних інцидентів у Журналі подій SCADA представлено на рисунку В.3.




Рисунок В.4 – Журнал подій SCADA з відображенням критичних та попереджувальних повідомлень .

Тест 5. Перевірка модуля конфігурації та збереження налаштувань

Методика: перехід за маршрутом /settings, зміна критичних порогових меж та параметрів MQTT-сервера за допомогою UI-елементів керування. Перезавантаження сторінки браузера.

Очікуваний результат: внесені зміни мають проходити клієнтську валідацію, асинхронно записуватися у NoSQL-структуру бази даних Firebase через Firebase SDK та успішно відновлюватися після оновлення сторінки.

Результат випробування: елементи форми (форми введення, перемикачі) працюють коректно. Завдяки механізму синхронізації Firebase, налаштування зберігаються атомарно і застосовуються логікою системи в реальному часі. Нові порогові ліміти успішно зчитуються Watchdog-компонентами.

### 6.3. Вимірювання швидкісних характеристик

Під час тестування швидкодії було проведено серію порівняльних вимірювань часових міток генерації пакета в Wokwi та його фіксації в React за допомогою Chrome DevTools. Середня скрізна затримка проходження інформації зафіксована в межах 0,3-0,6 секунди, що повністю задовольняє вимоги до систем моніторингу реального часу. Навантажувальне випробування підтвердило стабільність споживання оперативної пам'яті та відсутність втрат пакетів.

### Висновок

На основі проведених приймально-здавальних випробувань та аналізу отриманих результатів встановлено, що програмний виріб «Веб-панель моніторингу IoT-пристроїв у реальному часі» демонструє високу швидкодію, реактивність, стабільність та відмовоостійкість. Програмний комплекс повністю відповідає вимогам затвердженого Технічного завдання, функціонує без збоїв і помилок та є цілком готовим до практичної експлуатації за цільовим призначенням.

Виконавець: студент групи КІ-41 Плахотя О.О._____

**Додаток Г**

## Лістинг програмної реалізації

Лістинг Г.1 – Програмна реалізація асинхронного серверного шлюзу «mqtt-bridge.js» для маршрутизації та NoSQL-збереження потокових телеметричних даних.

```
import mqtt from 'mqtt';
import { initializeApp } from 'firebase/app';
import { getDatabase, ref, push } from 'firebase/database';

const firebaseConfig = {
  apiKey: "AIzaSyDz08rYjQlWa5Jej-pvCLUE6JW7vDfdaSM",
  authDomain: "iot-dashboard-project-dec3a.firebaseapp.com",
                                                    databaseURL:
"https://iot-dashboard-project-dec3a-default-rtdb.firebaseio.com/",
  projectId: "iot-dashboard-project-dec3a",
                                                  storageBucket:
"iot-dashboard-project-dec3a.firebasestorage.app",
  messagingSenderId: "15538146086",
  appId: "1:15538146086:web:d6e76aae06cd064c400fe7"
};

const app = initializeApp(firebaseConfig);
const db = getDatabase(app);

const mqttClient = mqtt.connect('wss://broker.emqx.io:8084/mqtt');

mqttClient.on('connect', () => {
   console.log('=== [MICT] Успішно підключено до MQTT-брокера EMQX
===');
  mqttClient.subscribe('sensors/#');
     console.log('===  [MICT]  Почалося  прослуховування  топіку
sensors/# ===');
});

mqttClient.on('message', (topic, message) => {
  try {
    const parsedData = JSON.parse(message.toString());
    const deviceId = parsedData.device || "unknown_device";

    const dataWithTimestamp = {
      ...parsedData,
      timestamp: Date.now()
    };

    const dbRef = ref(db, `history/${deviceId}`);
    push(dbRef, dataWithTimestamp);

    console.log(`[${deviceId}]: Дані успішно збережено в Firebase!
${JSON.stringify(dataWithTimestamp)}`);
  } catch (error) {
    console.error('Помилка обробки MQTT повідомлення:', error);
  }
});
```

Лістинг Г.2 – Програмна реалізація реактивного компонента «Devices» для менеджменту, категоризації та динамічної реєстрації IoT-вузлів.

```
import { useState, useMemo } from "react";
import { useApp } from "../components/AppContext";
import { DeviceCard } from "../components/DeviceCard";
import { Input } from "../components/ui/input";
import { Button } from "../components/ui/button";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "../components/ui/tabs";
import { Badge } from "../components/ui/badge";
import { Dialog, DialogContent, DialogDescription, DialogFooter, DialogHeader, DialogTitle, DialogTrigger } from "../components/ui/dialog";
import { Search, Filter, Plus, Trash2, CheckSquare, Square, ChevronDown, ChevronRight, AlertTriangle } from "lucide-react";

const METRIC_CATEGORIES = [
  {
    id: "eco",
    title: "Екологічні та кліматичні",
    metrics: [
      { key: "temperature", label: "Температура" },
      { key: "humidity", label: "Вологість" },
      { key: "lux", label: "Освітленість" },
      { key: "noise", label: "Рівень шуму" },
      { key: "aqi", label: "Індекс якості повітря" },
      { key: "co", label: "Концентрація чадного газу" },
      { key: "co2", label: "Вуглекислий газ" },
      { key: "lpg", label: "Зріджений газ" },
      { key: "methane", label: "Метан" },
      { key: "smoke", label: "Дим / Загазованість" },
      { key: "uv", label: "UV-індекс" },
      { key: "wind_speed", label: "Швидкість вітру" },
      { key: "wind_dir", label: "Напрям вітру" },
      { key: "rain", label: "Кількість опадів" },
      { key: "pm25", label: "Рівень пилу PM2.5 / PM10" },
      { key: "radiation", label: "Радіаційний фон" },
    ]
  },
  {
    id: "energy",
    title: "Енергетика та електроніка",
    metrics: [
      { key: "voltage", label: "Напруга мережі" },
      { key: "current", label: "Сила струму" },
      { key: "power", label: "Потужність" },
      { key: "energy_kwh", label: "Споживання енергії" },
      { key: "frequency", label: "Частота мережі" },
      { key: "cos_phi", label: "Коефіцієнт потужності" },
      { key: "battery_pct", label: "Заряд батареї" },
      { key: "battery_v", label: "Рівень акумулятора" },
      { key: "battery_temp", label: "Температура акумулятора" },
      { key: "power_status", label: "Стан живлення" },
      { key: "energy_daily", label: "Енергоспоживання за добу" },
      { key: "leakage_current", label: "Струм витоку" },
      { key: "relay_status", label: "Стан реле" },
    ]
  },
  {
    id: "industrial",
```

```
    title: "Промислові сенсори (Industry 4.0)",
    metrics: [
      { key: "vibration", label: "Вібрація" },
      { key: "rpm", label: "Оберти двигуна" },
      { key: "liquid_level", label: "Рівень рідини" },
      { key: "flow_rate", label: "Витрата рідини" },
      { key: "conveyor_speed", label: "Швидкість конвеєра" },
      { key: "valve_position", label: "Положення клапана" },
      { key: "motor_load", label: "Навантаження двигуна" },
      { key: "motor_temp", label: "Температура двигуна" },
      { key: "pipe_pressure", label: "Тиск у трубопроводі" },
      { key: "weight", label: "Вага / Маса" },
      { key: "distance", label: "Дистанція / Наближення" },
    ]
  },
  {
    id: "agro",
    title: "Розумний агро-сектор (Smart Agro)",
    metrics: [
      { key: "soil_moisture", label: "Вологість ґрунту" },
      { key: "ph", label: "Кислотність середовища" },
      { key: "ec", label: "Електропровідність ґрунту" },
      { key: "par_light", label: "Фотосинтетичне світло" },
    ]
  },
  {
    id: "network",
    title: "Мережа та стан пристрою",
    metrics: [
      { key: "rssi", label: "RSSI сигнал Wi-Fi" },
      { key: "ping", label: "Ping / затримка" },
      { key: "net_quality", label: "Якість мережі" },
      { key: "uptime", label: "Uptime пристрою" },
      { key: "ram_usage", label: "Використання RAM" },
      { key: "cpu_load", label: "Завантаження CPU" },
      { key: "free_mem", label: "Вільна пам'ять" },
      { key: "mqtt_status", label: "Статус MQTT" },
      { key: "last_packet", label: "Час останнього пакета" },
      { key: "error_count", label: "Кількість помилок" },
      { key: "device_ip", label: "Локальна IP-адреса" },
      { key: "firmware_v", label: "Версія прошивки" },
      { key: "soc_temp", label: "Температура чіпа SoC" },
    ]
  },
  {
    id: "security",
    title: "Безпека та доступ",
    metrics: [
      { key: "door_status", label: "Відкриття дверей" },
      { key: "pir_motion", label: "Рух у приміщенні" },
      { key: "gas_smoke_alarm", label: "Виявлення диму" },
      { key: "water_leak", label: "Виявлення витоку води" },
      { key: "alarm_state", label: "Стан сигналізації" },
      { key: "lock_state", label: "Стан замка" },
      { key: "rfid_nfc", label: "Ідентифікація RFID/NFC" },
      { key: "presence", label: "Наявність людини" },
    ]
  },
```

```
      {
        id: "gps",
        title: "GPS та транспортний моніторинг",
        metrics: [
          { key: "gps_coords", label: "Координати GPS" },
          { key: "speed", label: "Швидкість руху" },
          { key: "altitude", label: "Висота над рівнем моря" },
          { key: "heading", label: "Напрям руху" },
          { key: "fuel_level", label: "Рівень пального" },
          { key: "engine_temp", label: "Температура двигуна авто" },
          { key: "mileage", label: "Пробіг / Одометр" },
        ]
      }
    ];

    const ALL_METRICS_FLAT = METRIC_CATEGORIES.flatMap(c => c.metrics);

    const CATEGORY_FILTER_OPTIONS = [
      { value: "all", label: "Усі категорії" },
      ...METRIC_CATEGORIES.map((c) => ({ value: c.id, label: c.title })),
    ];

    const autoDetectCategoryId = (typeStr: string): string => {
      const s = (typeStr || "").toLowerCase();
      if (s.includes("температура") || s.includes("вологість") || s.includes("вуглекислий") || s.includes("co2") || s.includes("клімат") || s.includes("повітря") || s.includes("метео")) return "eco";
      if (s.includes("напруга") || s.includes("лічильник") || s.includes("живлення") || s.includes("реле") || s.includes("струм") || s.includes("ups") || s.includes("батарея")) return "energy";
      if (s.includes("двигун") || s.includes("тиск") || s.includes("конвеєр") || s.includes("промисловий") || s.includes("обороти")) return "industrial";
      if (s.includes("ґрунт") || s.includes("агро") || s.includes("теплиця") || s.includes("гідропоніка")) return "agro";
      if (s.includes("шлюз") || s.includes("контролер") || s.includes("вузол") || s.includes("мереж") || s.includes("wifi")) return "network";
      if (s.includes("охорона") || s.includes("сигналізація") || s.includes("безпека") || s.includes("замок")) return "security";
      if (s.includes("трекер") || s.includes("gps") || s.includes("транспорт") || s.includes("пробіг")) return "gps";
      return "eco";
    };

    export function Devices() {
      const { devices, status: mqttStatus, registerNewDevice, deleteDevice } = useApp();

      const [searchQuery, setSearchQuery] = useState("");
      const [categoryFilter, setCategoryFilter] = useState("all");
      const [statusTab, setStatusTab] = useState("all");

      const [showFilters, setShowFilters] = useState(false);

      const [isOpen, setIsOpen] = useState(false);
```

```
        const [isDeleteOpen, setIsDeleteOpen] = useState(false);

        const [newName, setNewName] = useState("");
        const [newId, setNewId] = useState("");
        const [newTopic, setNewTopic] = useState("sensors/data");
                const     [selectedMetrics,     setSelectedMetrics]     =
useState<string[]>([]);
         const [expandedCategories, setExpandedCategories] = useState<{
[key: string]: boolean }>({ eco: true });

           const   [isCustomMetricEnabled,   setIsCustomMetricEnabled]   =
useState(false);
        const [customMetricLabel, setCustomMetricLabel] = useState("");
        const [customMetricUnit, setCustomMetricUnit] = useState("");
          const  [deviceToDelete,  setDeviceToDelete]  =  useState<{  id:
string; name: string } | null>(null);

        const toggleCategory = (catId: string) => {
                setExpandedCategories(prev   =>   ({   ...prev,   [catId]:
!prev[catId] }));
        };

        const toggleMetricSelection = (metricKey: string) => {
          setSelectedMetrics(prev =>
            prev.includes(metricKey) ? prev.filter(k => k !== metricKey)
: [...prev, metricKey]
          );
        };

        const handleRegisterDevice = async (e: React.FormEvent) => {
          e.preventDefault();
          if (!newName || !newId || !newTopic) return;

          let typeDescription = selectedMetrics
            .map(k => ALL_METRICS_FLAT.find(m => m.key === k)?.label)
            .filter(Boolean)
            .join(", ");

          if (isCustomMetricEnabled && customMetricLabel) {
                    typeDescription   +=   `${typeDescription   ?   ",   "   :
""}${customMetricLabel}`;
          }

          try {
                    await   registerNewDevice(newId,   newName,   newTopic,
typeDescription || "Універсальний вузол");
            setNewName("");
            setNewId("");
            setNewTopic("sensors/data");
            setSelectedMetrics([]);
            setIsCustomMetricEnabled(false);
            setCustomMetricLabel("");
            setCustomMetricUnit("");
            setIsOpen(false);
          } catch (error) {
            console.error(error);
          }
        };
```

```
        const openDeleteModal = (id: string, name: string) => {
          setDeviceToDelete({ id, name });
          setIsDeleteOpen(true);
        };

        const confirmDeleteDevice = async () => {
          if (deviceToDelete) {
            await deleteDevice(deviceToDelete.id);
            setIsDeleteOpen(false);
            setDeviceToDelete(null);
          }
        };

        const filtered = useMemo(() => {
          return devices.filter((d: any) => {
            const matchSearch =
                d.name.toLowerCase().includes(searchQuery.toLowerCase())
||
              d.type.toLowerCase().includes(searchQuery.toLowerCase());

            const detectedCatId = autoDetectCategoryId(d.type);
            const matchCategory =
                      categoryFilter  ===  "all"  ||  detectedCatId  ===
categoryFilter;

            const matchStatus =
              statusTab === "all" || d.status === statusTab;

            return matchSearch && matchCategory && matchStatus;
          });
        }, [devices, searchQuery, categoryFilter, statusTab]);

        const counts = useMemo(() => {
          return {
            all: devices.length,
                   online:  devices.filter((d:  any)  =>  d.status  ===
"online").length,
                  warning:  devices.filter((d:  any)  =>  d.status  ===
"warning").length,
                  offline:  devices.filter((d:  any)  =>  d.status  ===
"offline").length,
          };
        }, [devices]);

        return (
          <div className="container mx-auto p-6 space-y-6">
                <div  className="flex  justify-between  items-start  gap-4
flex-wrap">
              <div>
                                    <h1   className="text-3xl   font-bold
tracking-tight">Пристрої</h1>
                <p className="text-muted-foreground">
                        Керування, конфігурація та статус мережі: <span
className="text-primary font-semibold">{mqttStatus}</span>
                </p>
              </div>
```

```
            <Dialog open={isOpen} onOpenChange={setIsOpen}>
              <DialogTrigger asChild>
                <Button className="gap-2"><Plus className="h-4 w-4" />
Додати пристрій</Button>
              </DialogTrigger>
               <DialogContent className="sm:max-w-[480px] max-h-[85vh]
overflow-y-auto">
                <DialogHeader>
                                        <DialogTitle>Реєстрація  нового
пристрою</DialogTitle>
                          <DialogDescription>Конфігурація вимірювальних
каналів та метаданих контролера.</DialogDescription>
                </DialogHeader>
                                    <form  onSubmit={handleRegisterDevice}
className="space-y-4 py-2">
                  <div className="space-y-1">
                                    <label  className="text-xs  font-semibold
text-muted-foreground">Назва пристрою</label>
                     <Input placeholder="напр., Контроль CO2 в теплиці"
value={newName}  onChange={(e)  =>  setNewName(e.target.value)}  required
/>
                  </div>
                  <div className="space-y-1">
                                    <label  className="text-xs  font-semibold
text-muted-foreground">Device ID (з коду заліза)</label>
                    <Input placeholder="напр., ESP32_CO2_Controller"
value={newId} onChange={(e) => setNewId(e.target.value)} required />
                  </div>
                  <div className="space-y-1">
                                    <label  className="text-xs  font-semibold
text-muted-foreground">MQTT Топік для публікації</label>
                                  <Input  value={newTopic}  onChange={(e)  =>
setNewTopic(e.target.value)} required />
                  </div>

                      <div className="space-y-2 border rounded-xl p-3
bg-muted/20">
                       <label className="text-sm font-bold block pb-1
border-b">Технічні метрики вузла:</label>
                                    <div  className="space-y-2  max-h-[30vh]
overflow-y-auto pr-1">
                      {METRIC_CATEGORIES.map((category) => {
                                                const  isExpanded  =
expandedCategories[category.id];
                        return (
                                    <div key={category.id} className="border
rounded-lg bg-background overflow-hidden shadow-sm">
                                  <div
                                                  className="flex  items-center
justify-between  p-2  bg-muted/40  cursor-pointer  hover:bg-muted/70
transition-colors select-none"
                                                    onClick={()  =>
toggleCategory(category.id)}
                                  >
                                                <span  className="text-xs
font-bold">{category.title}</span>
                                               {isExpanded  ?  <ChevronDown
className="h-3.5 w-3.5" /> : <ChevronRight className="h-3.5 w-3.5" />}
```

```
                                </div>

                                {isExpanded && (
                                    <div className="p-2 grid grid-cols-1 gap-1.5 bg-background border-t animate-in slide-in-from-top-1 duration-150">
                                        {category.metrics.map((metric) => {
                                            const isChecked = selectedMetrics.includes(metric.key);
                                            return (
                                                <div
                                                    key={metric.key}
                                                    className="flex items-center gap-2 cursor-pointer text-xs select-none p-1 rounded hover:bg-muted/30"
                                                    onClick={() => toggleMetricSelection(metric.key)}
                                                >
                                                    {isChecked ? <CheckSquare className="h-3.5 w-3.5 text-primary" /> : <Square className="h-3.5 w-3.5 text-muted-foreground" />}
                                                    <span>{metric.label}</span>
                                                </div>
                                            );
                                        })}
                                    </div>
                                )}
                            </div>
                        );
                    })}
                </div>
            </div>
            <DialogFooter className="pt-2">
                <Button type="submit" className="w-full h-10 font-semibold">Зареєструвати пристрій в системі</Button>
            </DialogFooter>
        </form>
    </DialogContent>
</Dialog>

<Dialog open={isDeleteOpen} onOpenChange={setIsDeleteOpen}>
    <DialogContent className="sm:max-w-[400px]">
        <DialogHeader className="flex flex-col items-center text-center space-y-3">
            <div className="p-3 rounded-full bg-destructive/10 text-destructive">
                <AlertTriangle className="h-6 w-6 animate-bounce" />
            </div>
            <DialogTitle className="text-xl font-bold">Підтвердження дії</DialogTitle>
            <DialogDescription className="text-sm text-muted-foreground">
                Ви дійсно хочете видалити пристрій <span className="font-bold text-foreground">"{deviceToDelete?.name}"</span>?
                Цю дію не можна скасувати, а всі логи з хмари Firebase будуть стерті.
            </DialogDescription>
```

```
                </DialogHeader>
                  <DialogFooter className="grid grid-cols-2 gap-2 pt-4
border-t w-full">
                                <Button variant="outline" onClick={() =>
setIsDeleteOpen(false)} className="w-full h-10">Скасувати</Button>
                                          <Button variant="destructive"
onClick={confirmDeleteDevice}    className="w-full    h-10    font-bold
shadow-sm">Видалити</Button>
                  </DialogFooter>
                </DialogContent>
              </Dialog>
            </div>

            <div className="space-y-4">
              <div className="flex items-center gap-3 w-full">
                <div className="relative flex-1">
                            <Search  className="absolute  left-3  top-1/2
-translate-y-1/2 h-4 w-4 text-muted-foreground" />
                  <Input
                    placeholder="Пошук пристроїв..."
                    value={searchQuery}
                    onChange={(e) => setSearchQuery(e.target.value)}
                    className="pl-10"
                  />
                </div>

                <Button
                  variant={showFilters ? "default" : "outline"}
                  size="icon"
                  onClick={() => {
                    setShowFilters(!showFilters);
                    if (showFilters) setCategoryFilter("all");
                  }}
                  className="shrink-0 transition-all duration-200"
                >
                            <Filter  className={`h-4  w-4  ${showFilters  ?
"animate-pulse" : ""}`} />
                </Button>
              </div>

              {showFilters && (
                <div className="flex items-center gap-2 flex-wrap border
p-3     rounded-xl     bg-muted/20     animate-in     slide-in-from-top-2
duration-200">
                                      <span  className="text-xs  font-bold
text-muted-foreground mr-1">Категорії:</span>
                  {CATEGORY_FILTER_OPTIONS.map((opt) => (
                    <Button
                      key={opt.value}
                      size="sm"
                       variant={categoryFilter === opt.value ? "default"
: "outline"}
                           className="text-xs h-7 px-2.5 transition-all
shadow-sm"
                      onClick={() => setCategoryFilter(opt.value)}
                    >
                                     {opt.value  ===  "all"  ?  opt.label  :
opt.label.split(" ").slice(1).join(" ")}
```

```
                    </Button>
                  ))}
                </div>
              ))}
            </div>

                  <Tabs  value={statusTab}  onValueChange={setStatusTab}
className="w-full">
              <TabsList>
                <TabsTrigger value="all">
                  Всі
                              <Badge  variant="secondary"  className="ml-2
text-xs">{counts.all}</Badge>
                </TabsTrigger>
                <TabsTrigger value="online">
                  Онлайн
                               <Badge  className="ml-2  text-xs  bg-green-500
hover:bg-green-500 text-white">{counts.online}</Badge>
                </TabsTrigger>
                <TabsTrigger value="warning">
                  Попередження
                            <Badge  className="ml-2  text-xs  bg-yellow-500
hover:bg-yellow-500 text-white">{counts.warning}</Badge>
                </TabsTrigger>
                <TabsTrigger value="offline">
                  Офлайн
                             <Badge  variant="destructive"  className="ml-2
text-xs">{counts.offline}</Badge>
                </TabsTrigger>
              </TabsList>

              <TabsContent value={statusTab} className="mt-6">
                {filtered.length === 0 ? (
                                    <div  className="text-center  py-16
text-muted-foreground border border-dashed rounded-xl bg-muted/10">
                    Пристроїв не знайдено
                  </div>
                ) : (
                          <div className="grid  grid-cols-1  md:grid-cols-2
lg:grid-cols-3 xl:grid-cols-4 gap-4">
                    {filtered.map((device: any) => (
                       <div key={device.id} className="relative group
transition-all flex flex-col">
                          <DeviceCard {...device} />

                          <Button
                            variant="destructive"
                            size="icon"
                                 className="absolute top-3 right-3 h-7 w-7
opacity-0   group-hover:opacity-100   transition-opacity   duration-200
shadow-md z-10"
                                 onClick={() => openDeleteModal(device.id,
device.name)}
                          >
                            <Trash2 className="h-3.5 w-3.5" />
                          </Button>
                        </div>
                    ))}
```

```
                </div>
              )}
            </TabsContent>
          </Tabs>
        </div>
      );
    }
```

Лістинг Г.3 – Програмна реалізація реактивного компонента «DeviceCard» з інтегрованою валідацією аварійних порогів та гнучким рендерингом телеметрії.

```
    import { useMemo } from "react";
    import {
      Activity, Thermometer, Droplets, Sun, Volume2, Wind, Eye, Flame,
AlertTriangle,
       Zap, Gauge, Battery, Key, ShieldAlert, Radio, Navigation,
Milestone, Timer, Cpu,
      WifiOff
    } from "lucide-react";
    import { Card, CardContent, CardHeader, CardTitle } from
"./ui/card";
    import { Badge } from "./ui/badge";

    interface DeviceCardProps {
      id: string;
      name: string;
      type: string;
      status: "online" | "offline" | "warning";
      lastUpdate?: string;
      [key: string]: any;
    }

    const METRIC_CONFIG_REGISTRY: {
      [key: string]: {
        label: string;
        unit: string;
         icon: React.ComponentType<{ className?: string;[key: string]:
any }>;
        color: string;
      }
    } = {
       "temperature": { label: "Температура", unit: "°C", icon:
Thermometer, color: "text-orange-500" },
       "temp": { label: "Температура", unit: "°C", icon: Thermometer,
color: "text-orange-500" },
       "humidity": { label: "Вологість", unit: "%", icon: Droplets,
color: "text-blue-500" },
       "hum": { label: "Вологість", unit: "%", icon: Droplets, color:
"text-blue-500" },
       "lux": { label: "Освітленість", unit: " Lux", icon: Sun, color:
"text-yellow-500" },
       "noise": { label: "Рівень шуму", unit: " dB", icon: Volume2,
color: "text-purple-500" },
       "aqi": { label: "Індекс якості повітря", unit: "", icon:
Activity, color: "text-green-500" },
       "co": { label: "Чадний газ (CO)", unit: " ppm", icon: Wind,
color: "text-red-400" },
```

```
        "co2": { label: "Вуглекислий газ", unit: " ppm", icon: Wind,
color: "text-green-500" },
        "lpg": { label: "Зріджений газ", unit: " ppm", icon: Flame,
color: "text-amber-500" },
        "methane": { label: "Метан", unit: " ppm", icon: Flame, color:
"text-orange-400" },
        "smoke": { label: "Дим / Загазованість", unit: "%", icon:
AlertTriangle, color: "text-red-500" },
        "uv": { label: "UV-індекс", unit: "", icon: Sun, color:
"text-violet-600" },
        "wind_speed": { label: "Швидкість вітру", unit: " m/s", icon:
Wind, color: "text-teal-500" },
        "wind_dir": { label: "Напрям вітру", unit: "°", icon:
Navigation, color: "text-teal-600" },
        "rain": { label: "Кількість опадів", unit: " mm", icon:
Droplets, color: "text-sky-500" },
        "pm25": { label: "Рівень пилу PM2.5 / PM10", unit: " µg/m³",
icon: Activity, color: "text-stone-500" },
        "radiation": { label: "Радіаційний фон", unit: " µSv/h", icon:
Activity, color: "text-lime-600" },

        "voltage": { label: "Напруга мережі", unit: " V", icon: Zap,
color: "text-yellow-400" },
        "current": { label: "Сила струму", unit: " A", icon: Zap, color:
"text-amber-500" },
        "power": { label: "Потужність", unit: " W", icon: Gauge, color:
"text-red-500" },
        "energy_kwh": { label: "Споживання енергії", unit: " kWh", icon:
Zap, color: "text-emerald-500" },
        "frequency": { label: "Частота мережі", unit: " Hz", icon:
Activity, color: "text-indigo-500" },
        "cos_phi": { label: "Коефіцієнт потужності", unit: "", icon:
Gauge, color: "text-blue-400" },
        "battery_pct": { label: "Заряд батареї", unit: "%", icon:
Battery, color: "text-green-500" },
        "battery_v": { label: "Рівень акумулятора", unit: " V", icon:
Battery, color: "text-teal-500" },
        "battery_temp": { label: "Температура акумулятора", unit: "°C",
icon: Thermometer, color: "text-red-400" },
        "power_status": { label: "Стан живлення", unit: "", icon: Zap,
color: "text-slate-500" },
        "energy_daily": { label: "Енергоспоживання за добу", unit: "
kWh", icon: Zap, color: "text-green-600" },
        "leakage_current": { label: "Струм витоку", unit: " mA", icon:
AlertTriangle, color: "text-orange-600" },
        "relay_status": { label: "Стан реле", unit: "", icon: Zap,
color: "text-yellow-600" },
        "co2_pump": { label: "Стан помпи CO₂", unit: "", icon: Zap,
color: "text-amber-500" },

        "vibration": { label: "Вібрація", unit: " mm/s", icon: Activity,
color: "text-fuchsia-500" },
        "rpm": { label: "Оберти двигуна", unit: " RPM", icon: Gauge,
color: "text-cyan-500" },
        "liquid_level": { label: "Рівень рідини", unit: "%", icon:
Droplets, color: "text-blue-600" },
        "flow_rate": { label: "Витрата рідини", unit: " l/min", icon:
Gauge, color: "text-blue-400" },
```

```
        "conveyor_speed": { label: "Швидкість конвеєра", unit: " m/s",
icon: Milestone, color: "text-stone-600" },
        "valve_position": { label: "Положення клапана", unit: "%", icon:
Gauge, color: "text-emerald-600" },
        "motor_load": { label: "Навантаження двигуна", unit: "%", icon:
Gauge, color: "text-orange-600" },
        "motor_temp": { label: "Температура двигуна", unit: "°C", icon:
Thermometer, color: "text-red-500" },
        "pipe_pressure": { label: "Тиск у трубопроводі", unit: " Bar",
icon: Gauge, color: "text-sky-600" },
        "weight": { label: "Вага / Маса", unit: " kg", icon: Gauge,
color: "text-amber-800" },
        "distance": { label: "Дистанція / Наближення", unit: " cm",
icon: Eye, color: "text-violet-500" },

        "soil_moisture": { label: "Вологість ґрунту", unit: "%", icon:
Droplets, color: "text-amber-700" },
        "ph": { label: "Кислотність середовища", unit: " pH", icon:
Activity, color: "text-lime-500" },
        "ec": { label: "Електропровідність ґрунту", unit: " mS/cm",
icon: Zap, color: "text-yellow-600" },
        "par_light": { label: "Фотосинтетичне світло", unit: " µmol",
icon: Sun, color: "text-amber-500" },

        "rssi": { label: "RSSI сигнал Wi-Fi", unit: " dBm", icon:
Activity, color: "text-green-500" },
        "ping": { label: "Ping / затримка", unit: " ms", icon: Timer,
color: "text-blue-400" },
        "net_quality": { label: "Якість мережі", unit: "%", icon:
Activity, color: "text-sky-500" },
        "uptime": { label: "Uptime пристрою", unit: " s", icon: Timer,
color: "text-slate-600" },
        "ram_usage": { label: "Використання RAM", unit: "%", icon: Cpu,
color: "text-purple-500" },
        "cpu_load": { label: "Завантаження CPU", unit: "%", icon: Cpu,
color: "text-indigo-500" },
        "free_mem": { label: "Вільна пам'ять", unit: " KB", icon: Cpu,
color: "text-zinc-500" },
        "mqtt_status": { label: "Статус MQTT", unit: "", icon: Activity,
color: "text-emerald-500" },
        "last_packet": { label: "Час останнього пакета", unit: "", icon:
Timer, color: "text-neutral-500" },
        "error_count": { label: "Кількість помилок", unit: "", icon:
AlertTriangle, color: "text-red-500" },
        "device_ip": { label: "Локальна IP-адреса", unit: "", icon:
Activity, color: "text-teal-600" },
        "firmware_v": { label: "Версія прошивки", unit: "", icon: Cpu,
color: "text-rose-500" },
        "soc_temp": { label: "Температура чіпа SoC", unit: "°C", icon:
Thermometer, color: "text-orange-600" },

        "door_status": { label: "Відкриття дверей", unit: "", icon: Key,
color: "text-amber-600" },
        "pir_motion": { label: "Рух у приміщенні", unit: "", icon: Eye,
color: "text-rose-500" },
        "gas_smoke_alarm": { label: "Виявлення диму", unit: "", icon:
Flame, color: "text-red-500" },
```

```
        "water_leak": { label: "Виявлення витоку води", unit: "", icon:
Droplets, color: "text-blue-500" },
        "alarm_state": { label: "Стан сигналізації", unit: "", icon:
ShieldAlert, color: "text-red-600" },
        "lock_state": { label: "Стан замка", unit: "", icon: Key, color:
"text-zinc-600" },
        "rfid_nfc": { label: "Ідентифікація RFID/NFC", unit: "", icon:
Radio, color: "text-cyan-600" },
        "presence": { label: "Наявність людини", unit: "", icon: Eye,
color: "text-emerald-500" }
      };

      function formatValue(key: string, value: any, label?: string):
string {
        if (value === undefined || value === null) return "---";
        if (typeof value === "string") return value;

        if (key === "gps_coords") return String(value);
        if (key === "door_status" || key === "lock_state") return value
? "Відкрито" : "Зачинено";
        if (key === "pir_motion" || key === "presence") return value ?
"Є рух" : "Немає";
        if (key === "alarm_state") return value ? "Тривога" : "Норма";
          if (key === "relay_status" || key === "co2_pump" ||
label?.includes("помпи") || label?.includes("реле")) {
            if (value === 100 || value === true || value === 1) return
"УВІМКН";
          if (value === 0 || value === false) return "ВИМКН";
        }
        if (key === "gas_smoke_alarm" || key === "water_leak") return
value ? "Виявлено!" : "Норма";
          if (key === "mqtt_status") return value ? "З'єднано" :
"Відключено";
        if (key === "power_status") return value === 1 ? "Мережа" :
"Батарея";

        const num = Number(value);
        if (Number.isInteger(num)) return `${num}`;
        return `${num.toFixed(1)}`;
      }

      export function DeviceCard({ id, name, type, status, lastUpdate =
"щойно", ...props }: DeviceCardProps) {

        const computedStatus = useMemo(() => {
          if (status === "offline") return "offline";

                                        const       savedThresholds       =
localStorage.getItem("scada_thresholds");
          if (!savedThresholds) return status;

          const thresholdLimits = JSON.parse(savedThresholds);
          let hasViolation = false;

          Object.entries(props).forEach(([key, value]) => {
            const lookupKey = key === "temp" ? "temperature" : key ===
"hum" ? "humidity" : key;
```

```
            if (thresholdLimits[lookupKey] && typeof value === "number")
{
              const minLimit = Number(thresholdLimits[lookupKey].min);
              const maxLimit = Number(thresholdLimits[lookupKey].max);

              if (value < minLimit || value > maxLimit) {
                hasViolation = true;
              }
            }
          });

          return hasViolation ? "warning" : "online";
        }, [status, props]);

        const getStatusColor = () => {
          switch (computedStatus) {
            case "online": return "bg-green-500";
            case "warning": return "bg-yellow-500";
            case "offline": return "bg-red-500";
          }
        };

        const getStatusBadge = () => {
          switch (computedStatus) {
                  case "online": return <Badge className="bg-green-500
hover:bg-green-600">Онлайн</Badge>;
                case "warning": return <Badge className="bg-yellow-500
hover:bg-yellow-600 animate-pulse">Попередження</Badge>;
                                  case "offline": return <Badge
variant="destructive">Офлайн</Badge>;
          }
        };

        const cleanTypeDisplay = useMemo(() => {
          if (!type || type.includes("/") || type.includes("sensors")) {
            const lowerName = name.toLowerCase();
             if (lowerName.includes("co2") || lowerName.includes("газ"))
return "Вуглекислий газ";
            return "Температура & Вологість";
          }
          return type;
        }, [type, name]);

        const metricEntries = useMemo(() => {
          const rawMap: Record<string, any> = {};
          const lowerType = (type || "").toLowerCase();
          const lowerName = (name || "").toLowerCase();
          const lowerCleanType = cleanTypeDisplay.toLowerCase();

          Object.entries(props).forEach(([key, value]) => {
            if (
              key !== "categoryId" &&
              key !== "metrics" &&
              key !== "topic" &&
              key !== "activeMetric" &&
              key !== "lastPacketTimestamp" &&
              value !== undefined &&
              value !== null
```

```
            ) {
              rawMap[key] = value;
            }
          });

              const impliesPureGas = lowerType.includes("co2") ||
lowerCleanType.includes("вуглекислий") || lowerName.includes("co2");
            const hasLiveClimateData = rawMap.temp !== undefined ||
rawMap.temperature !== undefined;

          const filteredEntries = Object.entries(rawMap).filter(([key])
=> {
            if (impliesPureGas && !hasLiveClimateData) {
              return key !== "temperature" && key !== "temp" && key !==
"humidity" && key !== "hum";
            }
            return true;
          });

          return filteredEntries
            .map(([key, value]) => {
              let currentKey = key;
               if (key === "voltage" && (lowerType.includes("помпа") ||
lowerName.includes("помпа") || lowerType.includes("реле"))) {
                currentKey = "co2_pump";
              }

              const config = METRIC_CONFIG_REGISTRY[currentKey];
              return {
                key,
                currentKey,
                value,
                label: config?.label ?? key,
                unit: config?.unit ?? "",
                IconComponent: config?.icon ?? Activity,
                colorClass: config?.color ?? "text-muted-foreground"
              };
            })
            .slice(0, 6);
        }, [props, type, name, cleanTypeDisplay]);

        const isOffline = computedStatus === "offline";

        return (
            <Card className="relative overflow-hidden transition-shadow
hover:shadow-md">
                <div className={`absolute top-0 left-0 right-0 h-1
${getStatusColor()}`} />

                  <CardHeader className="flex flex-row items-center
justify-between space-y-0 pb-2 pt-4">
                   <CardTitle className="text-sm font-semibold truncate
pr-2">{name}</CardTitle>
              {getStatusBadge()}
            </CardHeader>

            <CardContent>
```

```
                        <div  className="text-xs  text-muted-foreground
mb-4">{cleanTypeDisplay}</div>

              {isOffline ? (
                        <div  className="flex  items-center  gap-2  py-4
text-muted-foreground">
                  <WifiOff className="h-5 w-5 text-red-500" />
                  <div>
                                                    <div   className="text-xs
text-muted-foreground">З'єднання</div>
                                  <div  className="text-sm  font-semibold
text-red-500">Втрачено</div>
                  </div>
                </div>
              ) : (
                <div className="grid grid-cols-2 gap-x-4 gap-y-3">
                  {metricEntries.map((metric) => {
                    const Icon = metric.IconComponent;

                                                  const   savedThresholds   =
localStorage.getItem("scada_thresholds");
                    let isViolated = false;
                    if (savedThresholds) {
                      const limits = JSON.parse(savedThresholds);
                      const val = Number(metric.value);
                      if (limits[metric.currentKey]) {
                                                          const   minL   =
Number(limits[metric.currentKey].min);
                                                          const   maxL   =
Number(limits[metric.currentKey].max);
                        if (val < minL || val > maxL) isViolated = true;
                      }
                    }

                    return (
                      <div key={metric.key} className="flex items-center
gap-2 min-w-0">
                                      <Icon  className={`h-4  w-4  shrink-0
transition-colors   ${isViolated   ?   "text-red-500   animate-pulse"   :
metric.colorClass
                        }`} />
                        <div className="min-w-0">
                                  <div  className={`text-[10px]  truncate
leading-none   mb-0.5   ${isViolated   ?   "text-red-500   font-bold"   :
"text-muted-foreground"
                          }`}>
                          {metric.label}
                        </div>
                                    <div  className={`text-sm  font-bold
leading-none ${isViolated ? "text-red-600" : ""}`}>
                                        {formatValue(metric.currentKey,
metric.value, metric.label)}
                          <span className="text-xs font-normal ml-0.5
text-muted-foreground">
                            {metric.unit}
                          </span>
                        </div>
                      </div>
```

```
                    </div>
                  );
                })}
              </div>
            )}

              <div className="text-[10px] text-muted-foreground mt-4
pt-3 border-t border-border/50">
                Оновлено: {lastUpdate}
              </div>
            </CardContent>
          </Card>
        );
      }
```

Лістинг Г.4 – Програмна реалізація центрального координатора інтерфейсу «Dashboard» з асинхронною буферизацією трендів часових рядів.

```
      import { useState, useEffect, useMemo, useRef } from "react";
      import { useApp, Device } from "../components/AppContext";
      import { DeviceCard } from "../components/DeviceCard";
      import { DataChart } from "../components/DataChart";
      import { StatsCard } from "../components/StatsCard";
      import { AlertsPanel } from "../components/AlertsPanel";
      import { Activity, Server, ShieldCheck } from "lucide-react";
      import { Tabs, TabsContent, TabsList, TabsTrigger } from
"../components/ui/tabs";
      import { Button } from "../components/ui/button";
      import { Card } from "../components/ui/card";

      import { db } from "../../services/firebase";
      import { ref, onValue, query, limitToLast } from
"firebase/database";

      interface Alert {
        id: string;
        type: "info" | "warning" | "error" | "success";
        message: string;
        timestamp: string;
        device?: string;
      }

      interface ChartPoint {
        time: string;
        [metricKey: string]: string | number;
      }

      interface DetectedMetric {
        key: string;
        label: string;
        unit: string;
      }

      const STRICT_METRIC_MAP: { [key: string]: DetectedMetric } = {
         "температура": { key: "temp", label: "Температура", unit: "°C"
},
        "вологість": { key: "hum", label: "Вологість", unit: "%" },
         "вуглекислий": { key: "co2", label: "Вуглекислий газ", unit:
"ppm" },
```

```
        "напруга": { key: "voltage", label: "Напруга мережі", unit: "V"
},
         "стан помпи co₂": { key: "voltage", label: "Стан помпи CO₂",
unit: "% (ON/OFF)" },
        "сила струму": { key: "current", label: "Сила струму", unit: "A"
},
        "потужність": { key: "power", label: "Потужність", unit: "W" },
        "освітленість": { key: "lux", label: "Освітленість", unit: "Lux"
},
        "рівень шуму": { key: "noise", label: "Рівень шуму", unit: "dB"
}
      };

      export function Dashboard() {
        const { devices, status: mqttStatus } = useApp();
        const [chartData, setChartData] = useState<ChartPoint[]>([]);
                  const      [firebaseAlerts,      setFirebaseAlerts]      =
useState<Alert[]>([]);
                 const      [selectedDevices,      setSelectedDevices]      =
useState<string[]>([]);
        const [activeTab, setActiveTab] = useState<string>("overview");

         const latestTelemetryBuffer = useRef<{ [key: string]: number
}>({});
        const chartDataRef = useRef<ChartPoint[]>([]);

            const    [currentMetricFilter,    setCurrentMetricFilter]    =
useState<string>(() => {
          return localStorage.getItem("scada_selected_metric") || "";
        });

        useEffect(() => {
          const savedTab = localStorage.getItem("scada_active_tab");
          if (savedTab) setActiveTab(savedTab);
        }, []);

        const handleTabChange = (newTab: string) => {
          setActiveTab(newTab);
          localStorage.setItem("scada_active_tab", newTab);
        };

        const detectedMetrics = useMemo(() => {
          if (!devices || devices.length === 0) return [];
          const metricsMap: { [key: string]: DetectedMetric } = {};

          devices.forEach((device: Device) => {
            const typeStr = (device.type || "").toLowerCase();
            Object.keys(STRICT_METRIC_MAP).forEach((keyword) => {
              if (typeStr.includes(keyword)) {
                const config = STRICT_METRIC_MAP[keyword];
                if (!metricsMap[config.key]) {
                  metricsMap[config.key] = config;
                }
              }
            });
          });

          return Object.values(metricsMap);
```

```
        }, [devices]);

        useEffect(() => {
          if (detectedMetrics.length > 0) {
                                              const      savedFilter      =
localStorage.getItem("scada_selected_metric") || "";
                const  isValid  =  detectedMetrics.some(m  =>  m.key  ===
savedFilter);

            if (savedFilter && isValid) {
              setCurrentMetricFilter(savedFilter);
            } else {
              const defaultMetric = detectedMetrics[0].key;
              setCurrentMetricFilter(defaultMetric);
                           localStorage.setItem("scada_selected_metric",
defaultMetric);
            }
          } else {
            setCurrentMetricFilter("");
          }
        }, [detectedMetrics]);

        const handleMetricChange = (newMetric: string) => {
          setCurrentMetricFilter(newMetric);
          localStorage.setItem("scada_selected_metric", newMetric);
        };

        useEffect(() => {
           if  (devices  &&  devices.length  >  0  &&  selectedDevices.length
=== 0) {
            setSelectedDevices(devices.map((d: Device) => d.id));
          }
        }, [devices]);

        const filteredDevicesByMetric = useMemo(() => {
          if (!currentMetricFilter || !devices) return devices;

          return devices.filter((device: Device) => {
            const typeStr = (device.type || "").toLowerCase();
            let hasMetric = false;
            Object.keys(STRICT_METRIC_MAP).forEach((keyword) => {
              if (STRICT_METRIC_MAP[keyword].key === currentMetricFilter
&& typeStr.includes(keyword)) {
                hasMetric = true;
              }
            });
            return hasMetric;
          });
        }, [devices, currentMetricFilter]);

        const activeOnlineDevicesByMetric = useMemo(() => {
           return filteredDevicesByMetric.filter((d: Device) => d.status
!== "offline");
        }, [filteredDevicesByMetric]);

        const criticalDevices = useMemo(() => {
          return filteredDevicesByMetric.filter(
```

```
                (d: Device) => d.status === "warning" || d.status === "offline"
          );
        }, [filteredDevicesByMetric]);

        const toggleDeviceSelection = (deviceId: string) => {
          setSelectedDevices((prev) =>
                prev.includes(deviceId) ? prev.filter((id) => id !== deviceId) : [...prev, deviceId]
          );
        };

        useEffect(() => {
          if (!devices || devices.length === 0) return;

          const initialPointsMap: { [time: string]: ChartPoint } = {};

          const unsubscribes = devices.map((device: Device) => {
             const historyQuery = query(ref(db, `history/${device.id}`), limitToLast(15));

            return onValue(historyQuery, (snapshot) => {
              const data = snapshot.val();
              if (!data) return;

                 const logsArray = Object.values(data).sort((a: any, b: any) => a.timestamp - b.timestamp);

              if (chartDataRef.current.length === 0) {
                logsArray.forEach((log: any) => {
                  if (!log) return;

                        const logTimeStr = new Date(log.timestamp || Date.now()).toLocaleTimeString("uk-UA", {
                          hour: "2-digit", minute: "2-digit", second: "2-digit"
                  });

                  if (!initialPointsMap[logTimeStr]) {
                    initialPointsMap[logTimeStr] = { time: logTimeStr };
                  }

                  if (device.status !== "offline") {
                         const t = log.temp !== undefined ? log.temp : log.temperature;
                          const h = log.hum !== undefined ? log.hum : log.humidity;

                                                  if (t !== undefined) initialPointsMap[logTimeStr][`${device.id}_temp`] = t;
                                                  if (h !== undefined) initialPointsMap[logTimeStr][`${device.id}_hum`] = h;
                                                if (log.co2 !== undefined) initialPointsMap[logTimeStr][`${device.id}_co2`] = log.co2;
                                              if (log.voltage !== undefined) initialPointsMap[logTimeStr][`${device.id}_voltage`] = log.voltage;
                  }
                });
```

```
                                                const    sortedHistory    =
Object.values(initialPointsMap).slice(-15);
                chartDataRef.current = sortedHistory;
                setChartData(sortedHistory);
              }

              const latestLog: any = logsArray[logsArray.length - 1];
              if (!latestLog || device.status === "offline") return;

                                    if    (latestLog.temp    !==    undefined)
latestTelemetryBuffer.current[`${device.id}_temp`] = latestLog.temp;
                               if   (latestLog.temperature   !==   undefined)
latestTelemetryBuffer.current[`${device.id}_temp`]                          =
latestLog.temperature;
                                    if    (latestLog.hum    !==    undefined)
latestTelemetryBuffer.current[`${device.id}_hum`] = latestLog.hum;
                                 if    (latestLog.humidity   !==   undefined)
latestTelemetryBuffer.current[`${device.id}_hum`] = latestLog.humidity;
                                    if    (latestLog.co2    !==    undefined)
latestTelemetryBuffer.current[`${device.id}_co2`] = latestLog.co2;
                                   if    (latestLog.voltage   !==   undefined)
latestTelemetryBuffer.current[`${device.id}_voltage`]                       =
latestLog.voltage;
            });
          });

          return () => unsubscribes.forEach((unsub) => unsub());
        }, [devices]);

        useEffect(() => {
          const tickInterval = setInterval(() => {
                                      const     currentTimeStr     =     new
Date().toLocaleTimeString("uk-UA", {
              hour: "2-digit", minute: "2-digit", second: "2-digit"
            });

            devices.forEach((d: Device) => {
              if (d.status === "offline") {
                delete latestTelemetryBuffer.current[`${d.id}_temp`];
                delete latestTelemetryBuffer.current[`${d.id}_hum`];
                delete latestTelemetryBuffer.current[`${d.id}_co2`];
                delete latestTelemetryBuffer.current[`${d.id}_voltage`];
              }
            });

            const newPoint: ChartPoint = { time: currentTimeStr };
             Object.keys(latestTelemetryBuffer.current).forEach((key) =>
{
              newPoint[key] = latestTelemetryBuffer.current[key];
            });

                      const   updatedHistory   =   [...chartDataRef.current,
newPoint].slice(-15);
            chartDataRef.current = updatedHistory;
            setChartData(updatedHistory);
          }, 2000);
```

```
          return () => clearInterval(tickInterval);
        }, [devices]);

        useEffect(() => {
              const unsubscribe = onValue(query(ref(db, "alerts"),
limitToLast(20)), (snapshot) => {
            const data = snapshot.val();
            if (data) {
              const formatted = Object.keys(data).map((key) => {
                const dbType = data[key].type;
                 const validTypes: Alert["type"][] = ["info", "warning",
"error", "success"];
                                    const finalType: Alert["type"] =
validTypes.includes(dbType) ? dbType : "info";

                return {
                  id: key,
                  type: finalType,
                  message: data[key].message,
                  device: data[key].device,
                                                        timestamp: new
Date(data[key].timestamp).toLocaleTimeString("uk-UA", { hour:
"2-digit", minute: "2-digit" })
                };
              });
              setFirebaseAlerts(formatted.reverse());
            }
          });
          return () => unsubscribe();
        }, []);


        const combinedAlerts = useMemo<Alert[]>(() => {
          const activeHardwareAlerts: Alert[] = [];
           const currentTime = new Date().toLocaleTimeString("uk-UA", {
hour: "2-digit", minute: "2-digit" });

                                    const savedConfigs =
localStorage.getItem("scada_notification_configs");
                    const notificationConfigs = savedConfigs ?
JSON.parse(savedConfigs) : {
            globalNotifications: true,
            offlineAlerts: true,
            highValueAlerts: true
          };

          if (!notificationConfigs.globalNotifications) {
            return [{
              id: "system-muted",
              type: "info" as const,
                message: "Сповіщення вимкнено користувачем у глобальних
налаштуваннях.",
              device: "Система SCADA",
              timestamp: currentTime
            }];
          }
```

```
                                    const savedThresholds =
localStorage.getItem("scada_thresholds");
                    const thresholdLimits = savedThresholds ?
JSON.parse(savedThresholds) : null;

          devices.forEach((d: Device) => {
                                if (d.status === "offline" &&
notificationConfigs.offlineAlerts) {
              activeHardwareAlerts.push({
                id: `live-offline-${d.id}`,
                type: "error",
                message: "З'єднання втрачено",
                device: d.name,
                    timestamp: d.lastUpdate && d.lastUpdate !== "немає
даних" ? d.lastUpdate : "щойно"
              });
            }

                                if (d.status === "warning" &&
notificationConfigs.highValueAlerts) {
                let alertMessage = "Виявлено відхилення технологічних
параметрів";

              if (thresholdLimits) {
                Object.entries(d).forEach(([key, value]) => {
                  if (thresholdLimits[key] && typeof value === "number")
{
                    const minLimit = Number(thresholdLimits[key].min);
                    const maxLimit = Number(thresholdLimits[key].max);

                    let UkrainianLabel = key;
                    Object.keys(STRICT_METRIC_MAP).forEach(k => {
                                if (STRICT_METRIC_MAP[k].key === key)
UkrainianLabel = STRICT_METRIC_MAP[k].label;
                    });

                    if (value > maxLimit) {
                      alertMessage = `Перевищено ліміт`;
                    } else if (value < minLimit) {
                      alertMessage = `Показник нижче норми`;
                    }
                  }
                });
              }

              activeHardwareAlerts.push({
                id: `live-warning-${d.id}`,
                type: "warning",
                message: alertMessage,
                device: d.name,
                timestamp: "щойно"
              });
            }
          });

          const allowedFirebaseAlerts = firebaseAlerts.filter(alert => {
                                if (alert.type === "warning" &&
!notificationConfigs.highValueAlerts) return false;
```

```
            return true;
          });

                      const    allAlerts    =    [...activeHardwareAlerts,
...allowedFirebaseAlerts];

          if (allAlerts.length === 0) {
            return [{
              id: "system-clean",
              type: "success" as const,
                    message:  "Усі  системи  працюють  у  штатному  режимі.
Інцидентів не зафіксовано.",
              device: "Система SCADA",
              timestamp: currentTime
            }];
          }

          return tyrannyUniqueAlerts(allAlerts);
        }, [devices, firebaseAlerts]);

        function tyrannyUniqueAlerts(arr: Alert[]): Alert[] {
          const seen = new Set();
          return arr.filter(item => {
            const duplicate = seen.has(item.id);
            seen.add(item.id);
            return !duplicate;
          });
        }

        const getDynamicDataKeys = () => {
          if (!currentMetricFilter) return [];
          return devices
            .filter((d: Device) =>
              selectedDevices.includes(d.id) &&
              d.status !== "offline" &&
              activeOnlineDevicesByMetric.some(fd => fd.id === d.id)
            )
            .map((d: Device, idx: number) => {
                      const  colors  =  ["#3b82f6",  "#10b981",  "#ef4444",
"#f59e0b", "#6366f1", "#06b6d4", "#a855f7"];
              return {
                key: `${d.id}_${currentMetricFilter}`,
                color: colors[idx % colors.length],
                name: d.name
              };
            });
        };

         const currentMetricObject = detectedMetrics.find(m => m.key ===
currentMetricFilter);
          const  onlineDevicesCount  =  filteredDevicesByMetric.filter((d:
Device) => d.status === "online" || d.status === "warning").length;
         const  offlineDevicesCount  =  filteredDevicesByMetric.filter((d:
Device) => d.status === "offline").length;

        const renderDeviceFilters = () => {
          return (
            <div className="flex flex-wrap gap-2 w-full">
```

```
            {activeOnlineDevicesByMetric.map((device: Device) => {
              const isSelected = selectedDevices.includes(device.id);
              return (
                <Button
                  key={device.id}
                  variant={isSelected ? "default" : "outline"}
                  size="sm"
                  onClick={() => toggleDeviceSelection(device.id)}
                   className="text-xs h-8 px-3 font-semibold shadow-sm
transition-all"
                >
                                <span  className={`w-2  h-2  rounded-full  mr-2
${device.status   ===   "warning"   ?   "bg-amber-500   animate-ping"   :
"bg-green-500"
                    }`} />
                  {device.name}
                </Button>
              );
            })}
          </div>
        );
      };

      return (
            <div  className="container  mx-auto  p-4  md:p-6  space-y-6
overflow-x-hidden">

            <div  className="flex  flex-col  sm:flex-row  sm:items-center
justify-between gap-4 border-b pb-4">
            <div>
                        <h1  className="text-2xl  md:text-3xl  font-bold
tracking-tight">Універсальна SCADA-Панель</h1>
                                  <p   className="text-xs   md:text-sm
text-muted-foreground">Реальний    час    •    Стан    MQTT:    <span
className="text-primary font-semibold">{mqttStatus}</span></p>
            </div>

                  <div  className="flex  items-center  justify-between
sm:justify-start  gap-2  border  bg-muted/40  p-2  rounded-xl  shadow-sm
w-full sm:w-auto">
              <span className="text-xs font-bold text-muted-foreground
px-1 whitespace-nowrap">Параметр:</span>
              {detectedMetrics.length === 0 ? (
                 <span className="text-xs font-semibold text-amber-600
px-2 animate-pulse">Сканування...</span>
              ) : (
                <select
                     className="p-1.5 border rounded-lg bg-background
text-xs    font-bold    shadow-sm    focus:outline-none    focus:ring-1
focus:ring-primary w-full sm:min-w-[160px] cursor-pointer"
                  value={currentMetricFilter}
                  onChange={(e) => handleMetricChange(e.target.value)}
                >
                  {detectedMetrics.map((m) => (
                                                    <option   key={m.key}
value={m.key}>{m.label}</option>
                  ))}
                </select>
```

```
                )}
              </div>
            </div>

                    <div  className="grid  grid-cols-1  sm:grid-cols-2
lg:grid-cols-3 gap-4">
              <StatsCard
                title="Кількість пристроїв в поточній категорії"
                value={filteredDevicesByMetric.length}
                icon={Server}
                            description={`${onlineDevicesCount}  active,
${offlineDevicesCount} offline`}
              />
                                  <StatsCard   title="Режим   аналітики"
value={currentMetricObject ? currentMetricObject.label : "Scanning..."}
icon={Activity} />
            </div>

                <div  className="grid  grid-cols-1  lg:grid-cols-3  gap-6
items-start w-full">

                 <div  className="lg:col-span-2  space-y-6  w-full  min-w-0
overflow-hidden">
                {detectedMetrics.length === 0 ? (
                                      <Card  className="p-10  text-center
text-muted-foreground border-dashed text-sm">
                    В базі даних не знайдено зареєстрованих пристроїв з
активованими метриками.
                  </Card>
                ) : (
                                                <Tabs   value={activeTab}
onValueChange={handleTabChange} className="w-full">

                    <div className="w-full overflow-x-auto pb-1">
                           <TabsList  className="grid grid-cols-3 w-full
min-w-[360px]">
                        <TabsTrigger value="overview" className="text-xs
md:text-sm">Огляд подій</TabsTrigger>
                            <TabsTrigger value="chart" className="text-xs
md:text-sm">Графік трендів</TabsTrigger>
                            <TabsTrigger value="cards" className="text-xs
md:text-sm">Картки                                                  модулів
({filteredDevicesByMetric.length})</TabsTrigger>
                      </TabsList>
                    </div>

                          <TabsContent  value="overview"  className="mt-4
outline-none w-full">
                    {criticalDevices.length === 0 ? (
                              <Card className="p-6 md:p-8 flex flex-col
items-center  justify-center  text-center  space-y-3  border-green-500/30
bg-green-500/5 rounded-2xl">
                                        <div  className="p-3  rounded-full
bg-green-500/10 text-green-500">
                                  <ShieldCheck className="h-8 w-8 md:h-10
md:w-10" />
                          </div>
```

```
                            <h3 className="text-lg md:text-xl font-bold
text-green-600">Діагностика: Системи в нормі</h3>
                                        <p  className="text-xs  md:text-sm
text-muted-foreground  max-w-md">Усі  модулі  цієї  категорії  функціонують
стабільно.</p>
                        </Card>
                      ) : (
                        <div className="grid grid-cols-1 sm:grid-cols-2
gap-4 w-full">
                          {criticalDevices.map((device: Device) => (
                            <DeviceCard key={device.id} {...device} />
                          ))}
                        </div>
                      )}
                    </TabsContent>

                            <TabsContent  value="chart"  className="mt-4
outline-none w-full">
                      <Card className="p-4 sm:p-6 rounded-2xl shadow-sm
w-full overflow-hidden border">

                          <div className="flex flex-col gap-4 border-b
pb-4 w-full">
                        <div className="w-full">
                                  <h3  className="text-base  md:text-lg
font-bold">Динаміка наживо: {currentMetricObject?.label}</h3>
                                <p  className="text-[11px]  md:text-xs
text-muted-foreground">Аналітика              потоків              даних
({currentMetricObject?.unit || "---"})</p>
                        </div>
                                              <div  className="w-full
py-0.5">{renderDeviceFilters()}</div>
                        </div>

                                  <div  className="w-full  mt-4  h-[380px]
xs:h-[420px]  sm:h-[460px]  md:h-[500px]  block  relative  overflow-hidden">
                        <div className="w-full h-full min-w-[280px]">
                          <DataChart
                               title={`${currentMetricObject?.label ||
""} (${currentMetricObject?.unit || ""})`}
                            data={chartData}
                            dataKeys={getDynamicDataKeys()}
                            type="line"
                          />
                        </div>
                      </div>
                    </Card>
                  </TabsContent>

                            <TabsContent  value="cards"  className="mt-4
outline-none w-full">
                      <div className="grid grid-cols-1 sm:grid-cols-2
gap-4 w-full">
                      {filteredDevicesByMetric.map((device: Device) =>
(
                          <DeviceCard key={device.id} {...device} />
                        ))}
                    </div>
```

```
                    </TabsContent>
                  </Tabs>
                )}
              </div>

                <div className="w-full lg:sticky lg:top-6 mt-4 lg:mt-0
pt-2">
                <AlertsPanel alerts={combinedAlerts} />
              </div>

            </div>
          </div>
        );
      }
```

Лістинг Г.5 – Програмна реалізація модуля глобального адміністрування «Settings» із механізмами інтелектуальної валідації та локального збереження конфігурацій SCADA.

```
      import { useState } from "react";
      import { Card, CardContent, CardDescription, CardHeader, CardTitle
} from "../components/ui/card";
      import { Label } from "../components/ui/label";
      import { Input } from "../components/ui/input";
      import { Switch } from "../components/ui/switch";
      import { Button } from "../components/ui/button";
      import   {   Select,   SelectContent,   SelectItem,   SelectTrigger,
SelectValue } from "../components/ui/select";
      import { Separator } from "../components/ui/separator";
      import  {  Bell,  Thermometer,  Save,  Sliders,  LayoutGrid  }  from
"lucide-react";
      import { toast } from "sonner";

      const CONFIGURABLE_CATEGORIES = [
        {
          id: "eco",
          title: "Екологічні та кліматичні",
          metrics: [
             { key: "temp", label: "Температура", defMin: 18, defMax: 26
},
            { key: "hum", label: "Вологість", defMin: 30, defMax: 60 },
              { key: "lux", label: "Освітленість", defMin: 100, defMax:
5000 },
             { key: "noise", label: "Рівень шуму", defMin: 0, defMax: 75
},
              { key: "aqi", label: "Індекс якості повітря", defMin: 0,
defMax: 100 },
             { key: "co", label: "Концентрація чадного газу", defMin: 0,
defMax: 5 },
            { key: "co2", label: "Вуглекислий газ", defMin: 400, defMax:
1200 },
            { key: "lpg", label: "Зріджений газ", defMin: 0, defMax: 200
},
            { key: "methane", label: "Метан", defMin: 0, defMax: 500 },
              { key: "smoke", label: "Дим / Загазованість", defMin: 0,
defMax: 10 },
            { key: "uv", label: "UV-індекс", defMin: 0, defMax: 5 },
              { key: "wind_speed", label: "Швидкість вітру", defMin: 0,
defMax: 15 },
```

```
            { key: "wind_dir", label: "Напрям вітру", defMin: 0, defMax:
360 },
            { key: "rain", label: "Кількість опадів", defMin: 0, defMax:
50 },
            { key: "pm25", label: "Рівень пилу PM2.5 / PM10", defMin: 0,
defMax: 35 },
              { key: "radiation", label: "Радіаційний фон", defMin: 0,
defMax: 0.3 },
          ]
        },
        {
          id: "energy",
          title: "Енергетика та електроніка",
          metrics: [
               { key: "voltage", label: "Напруга мережі", defMin: 200,
defMax: 240 },
              { key: "current", label: "Сила струму", defMin: 0, defMax:
16 },
            { key: "power", label: "Потужність", defMin: 0, defMax: 3500
},
            { key: "energy_kwh", label: "Споживання енергії", defMin: 0,
defMax: 1000 },
             { key: "frequency", label: "Частота мережі", defMin: 49.5,
defMax: 50.5 },
              { key: "cos_phi", label: "Коефіцієнт потужності", defMin:
0.8, defMax: 1.0 },
              { key: "battery_pct", label: "Заряд батареї", defMin: 20,
defMax: 100 },
               { key: "battery_v", label: "Рівень акумулятора", defMin:
10.5, defMax: 14.2 },
               { key: "battery_temp", label: "Температура акумулятора",
defMin: 15, defMax: 45 },
              { key: "power_status", label: "Стан живлення", defMin: 0,
defMax: 1 },
              { key: "energy_daily", label: "Енергоспоживання за добу",
defMin: 0, defMax: 15 },
             { key: "leakage_current", label: "Струм витоку", defMin: 0,
defMax: 30 },
                { key: "relay_status", label: "Стан реле", defMin: 0,
defMax: 1 },
          ]
        },
        {
          id: "industrial",
          title: "Промислові сенсори (Industry 4.0)",
          metrics: [
              { key: "vibration", label: "Вібрація", defMin: 0, defMax:
4.5 },
              { key: "rpm", label: "Оберти двигуна", defMin: 0, defMax:
3000 },
              { key: "liquid_level", label: "Рівень рідини", defMin: 10,
defMax: 90 },
               { key: "flow_rate", label: "Витрата рідини", defMin: 0,
defMax: 100 },
                { key: "conveyor_speed", label: "Швидкість конвеєра",
defMin: 0, defMax: 2 },
            { key: "valve_position", label: "Положення клапана", defMin:
0, defMax: 100 },
```

```
            { key: "motor_load", label: "Навантаження двигуна", defMin:
0, defMax: 85 },
             { key: "motor_temp", label: "Температура двигуна", defMin:
20, defMax: 80 },
              { key: "pipe_pressure", label: "Тиск у трубопроводі",
defMin: 1, defMax: 6 },
             { key: "weight", label: "Вага / Маса", defMin: 0, defMax:
500 },
            { key: "distance", label: "Дистанція / Наближення", defMin:
5, defMax: 200 },
          ]
        },
        {
          id: "agro",
          title: "Розумний агро-сектор (Smart Agro)",
          metrics: [
            { key: "soil_moisture", label: "Вологість ґрунту", defMin:
40, defMax: 80 },
             { key: "ph", label: "Кислотність середовища", defMin: 5.5,
defMax: 7.5 },
              { key: "ec", label: "Електропровідність ґрунту", defMin:
0.5, defMax: 2.5 },
            { key: "par_light", label: "Фотосинтетичне світло", defMin:
200, defMax: 1000 },
          ]
        },
        {
          id: "network",
          title: "Мережа та стан пристрою",
          metrics: [
             { key: "rssi", label: "RSSI сигнал Wi-Fi", defMin: -85,
defMax: -30 },
            { key: "ping", label: "Ping / затримка", defMin: 1, defMax:
150 },
             { key: "net_quality", label: "Якість мережі", defMin: 50,
defMax: 100 },
              { key: "uptime", label: "Uptime пристрою", defMin: 0,
defMax: 604800 },
             { key: "ram_usage", label: "Використання RAM", defMin: 0,
defMax: 85 },
              { key: "cpu_load", label: "Завантаження CPU", defMin: 0,
defMax: 75 },
             { key: "free_mem", label: "Вільна пам'ять", defMin: 4000,
defMax: 32000 },
              { key: "mqtt_status", label: "Статус MQTT", defMin: 1,
defMax: 1 },
              { key: "last_packet", label: "Час останнього пакета",
defMin: 0, defMax: 10 },
            { key: "error_count", label: "Кількість помилок", defMin: 0,
defMax: 5 },
             { key: "device_ip", label: "Локальна IP-адреса", defMin: 0,
defMax: 0 },
             { key: "firmware_v", label: "Версія прошивки", defMin: 0,
defMax: 0 },
             { key: "soc_temp", label: "Температура чіпа SoC", defMin:
30, defMax: 70 },
          ]
        },
```

```
        {
          id: "security",
          title: "Безпека та доступ",
          metrics: [
             { key: "door_status", label: "Відкриття дверей", defMin: 0,
defMax: 1 },
              { key: "pir_motion", label: "Рух у приміщенні", defMin: 0,
defMax: 1 },
              { key: "gas_smoke_alarm", label: "Виявлення диму", defMin:
0, defMax: 0 },
             { key: "water_leak", label: "Виявлення витоку води", defMin:
0, defMax: 0 },
             { key: "alarm_state", label: "Стан сигналізації", defMin: 0,
defMax: 1 },
             { key: "lock_state", label: "Стан замка", defMin: 0, defMax:
1 },
              { key: "rfid_nfc", label: "Ідентифікація RFID/NFC", defMin:
0, defMax: 1 },
               { key: "presence", label: "Наявність людини", defMin: 0,
defMax: 1 },
          ]
        },
        {
          id: "gps",
          title: "GPS та транспортний моніторинг",
          metrics: [
              { key: "gps_coords", label: "Координати GPS", defMin: 0,
defMax: 0 },
             { key: "speed", label: "Швидкість руху", defMin: 0, defMax:
110 },
             { key: "altitude", label: "Висота над рівнем моря", defMin:
0, defMax: 2000 },
              { key: "heading", label: "Напрям руху", defMin: 0, defMax:
360 },
              { key: "fuel_level", label: "Рівень пального", defMin: 15,
defMax: 100 },
               { key: "engine_temp", label: "Температура двигуна авто",
defMin: 70, defMax: 105 },
                { key: "mileage", label: "Пробіг / Одометр", defMin: 0,
defMax: 999999 },
          ]
        }
      ];

      export function Settings() {
            const   [activeCategoryFilter,   setActiveCategoryFilter]   =
useState<string>("eco");

            const   [notificationConfigs,   setNotificationConfigs]   =
useState<Record<string, boolean>>(() => {
                                                  const         saved         =
localStorage.getItem("scada_notification_configs");
          if (saved) return JSON.parse(saved);
          return {
            globalNotifications: true,
            offlineAlerts: true,
            highValueAlerts: true
          };
```

```
        });

         const [thresholds, setThresholds] = useState<Record<string, {
min: string; max: string }>>(() => {
          const saved = localStorage.getItem("scada_thresholds");
          if (saved) return JSON.parse(saved);

           const initial: Record<string, { min: string; max: string }> =
{};
          CONFIGURABLE_CATEGORIES.forEach(cat => {
            cat.metrics.forEach(m => {
                      initial[m.key] = { min: String(m.defMin), max:
String(m.defMax) };
            });
          });
          return initial;
        });

        const handleSwitchChange = (configKey: string, checked: boolean)
=> {
          setNotificationConfigs(prev => {
            if (configKey === "globalNotifications") {
                           if (checked && !prev.offlineAlerts &&
!prev.highValueAlerts) {
                return {
                  globalNotifications: true,
                  offlineAlerts: true,
                  highValueAlerts: true
                };
              }
              return {
                globalNotifications: checked,
                offlineAlerts: checked ? prev.offlineAlerts : false,
                highValueAlerts: checked ? prev.highValueAlerts : false,
              };
            }

            if (checked && !prev.globalNotifications) {
              return {
                ...prev,
                globalNotifications: true,
                [configKey]: true
              };
            }

            if (!checked) {
                    const otherKey = configKey === "offlineAlerts" ?
"highValueAlerts" : "offlineAlerts";
              const isOtherAlsoDisabled = !prev[otherKey];

              if (isOtherAlsoDisabled) {
                return {
                  ...prev,
                  globalNotifications: false,
                  [configKey]: false
                };
              }
            }
```

```
            return {
              ...prev,
              [configKey]: checked
            };
          });
        };

        const handleThresholdChange = (metricKey: string, field: "min" |
"max", value: string) => {
          setThresholds(prev => ({
            ...prev,
            [metricKey]: {
              ...prev[metricKey],
              [field]: value
            }
          }));
        };

        const handleSave = () => {
          let hasValidationError = false;
          Object.keys(thresholds).forEach(key => {
            const min = Number(thresholds[key].min);
            const max = Number(thresholds[key].max);
            if (min > max) hasValidationError = true;
          });

          if (hasValidationError) {
            toast.error("Помилка конфігурації: Мінімальний поріг не може
бути більшим за максимальний!");
            return;
          }

                                    localStorage.setItem("scada_thresholds",
JSON.stringify(thresholds));
                        localStorage.setItem("scada_notification_configs",
JSON.stringify(notificationConfigs));
           toast.success("Всі технологічні пороги та стани перемикачів
збережено успішно!");
        };

         const currentCategory = CONFIGURABLE_CATEGORIES.find(c => c.id
=== activeCategoryFilter);

        return (
              <div  className="container  mx-auto  p-4  md:p-6  space-y-6
max-w-4xl overflow-x-hidden">
            <div>
                          <h1  className="text-2xl  md:text-3xl  font-bold
tracking-tight">Налаштування</h1>
                  <p  className="text-sm  text-muted-foreground">Керування
технологічними лімітами та конфігурацією SCADA</p>
            </div>

            <Card className="shadow-sm rounded-2xl">
              <CardHeader>
                  <CardTitle  className="flex  items-center  gap-2  text-lg
font-bold">
```

```
                <Bell className="h-5 w-5 text-primary" />
                Маршрутизація сповіщень
              </CardTitle>
                        <CardDescription>Керування потоками системних
алертів</CardDescription>
            </CardHeader>
            <CardContent className="space-y-4">
              <div className="flex items-center justify-between">
                <div className="space-y-0.5 pr-2">
                    <Label className="text-sm font-semibold">Увімкнути
сповіщення</Label>
                                                    <p   className="text-xs
text-muted-foreground">Глобальний комутатор пуш-повідомлень подій</p>
                </div>
                <Switch
                  checked={notificationConfigs.globalNotifications}
                                          onCheckedChange={(checked)   =>
handleSwitchChange("globalNotifications", checked)}
                />
              </div>

              <Separator className="my-2" />

              <div className="flex items-center justify-between">
                <div className="space-y-0.5 pr-2">
                   <Label className="text-sm font-semibold">Сповіщення
про офлайн пристрої</Label>
                                                    <p   className="text-xs
text-muted-foreground">Миттєвий       триггер       при       спрацюванні
Watchdog-таймера</p>
                </div>
                <Switch
                  checked={notificationConfigs.offlineAlerts}
                                          onCheckedChange={(checked)   =>
handleSwitchChange("offlineAlerts", checked)}
                />
              </div>

              <div className="flex items-center justify-between">
                <div className="space-y-0.5 pr-2">
                                                <Label   className="text-sm
font-semibold">Попередження про високі показники</Label>
                                                    <p   className="text-xs
text-muted-foreground">Активація   тривоги   при   перетині   встановлених
меж</p>
                </div>
                <Switch
                  checked={notificationConfigs.highValueAlerts}
                                          onCheckedChange={(checked)   =>
handleSwitchChange("highValueAlerts", checked)}
                />
              </div>
            </CardContent>
          </Card>

          <Card className="shadow-sm rounded-2xl border-primary/20">
            <CardHeader className="pb-4">
```

```
                  <CardTitle className="flex items-center gap-2 text-lg
font-bold">
                  <Thermometer className="h-5 w-5 text-orange-500" />
                  Промисловий конфігуратор меж сповіщень
                </CardTitle>
                  <CardDescription>Встановіть граничні Min/Max значення
для автоматичного генератора інцидентів</CardDescription>
              </CardHeader>
              <CardContent className="space-y-6">
                <div className="space-y-2">
                          <Label  className="text-xs  font-bold  uppercase
tracking-wider text-muted-foreground flex items-center gap-1">
                    <LayoutGrid className="w-3 h-3" /> Оберіть категорію
заліза для налаштування:
                  </Label>
                                      <Select  value={activeCategoryFilter}
onValueChange={setActiveCategoryFilter}>
                    <SelectTrigger className="rounded-xl font-semibold
bg-muted/40">
                        <SelectValue placeholder="Оберіть технологічну
категорію" />
                    </SelectTrigger>
                    <SelectContent>
                      {CONFIGURABLE_CATEGORIES.map(cat => (
                                  <SelectItem  key={cat.id}  value={cat.id}
className="font-medium">{cat.title}</SelectItem>
                      ))}
                    </SelectContent>
                  </Select>
                </div>

                <Separator className="my-2" />

                {currentCategory && (
                  <div className="space-y-6 animate-fadeIn">
                      <h3 className="text-sm font-bold text-primary flex
items-center gap-1.5 uppercase tracking-wider">
                          <Sliders  className="w-3.5  h-3.5" /> Секція:
{currentCategory.title}
                    </h3>

                        <div  className="grid  grid-cols-1  md:grid-cols-2
gap-x-6 gap-y-4">
                    {currentCategory.metrics.map((metric) => {
                       const currentValues = thresholds[metric.key] ||
{ min: "0", max: "0" };
                        const isInvalid = Number(currentValues.min) >
Number(currentValues.max);

                      return (
                        <div
                          key={metric.key}
                                  className={`space-y-2  p-3  rounded-xl
bg-muted/30  border  transition-colors  ${isInvalid  ?  "border-red-500/50
bg-red-500/5" : "border-muted/50"
                                    }`}
                        >
```

```
                                    <Label className={`text-xs font-bold
truncate block mb-1 ${isInvalid ? "text-red-600" :
"text-foreground"}`}>
                                        {metric.label} (`{metric.key}`) {isInvalid
&& "(Min > Max)"}
                                    </Label>

                                    <div className="grid grid-cols-2 gap-2">
                                      <div className="space-y-1">
                                          <span className="text-[9px] uppercase
tracking-wider font-bold text-muted-foreground">Поріг Min</span>
                                        <Input
                                          type="number"
                                          placeholder="Min"
                                          value={currentValues.min}
                                                                onChange={(e) =>
handleThresholdChange(metric.key, "min", e.target.value)}
                                                                className={`h-8 text-xs font-bold
rounded-lg bg-background ${isInvalid ? "border-red-500
focus-visible:ring-red-500" : ""}`}
                                          />
                                      </div>
                                      <div className="space-y-1">
                                          <span className="text-[9px] uppercase
tracking-wider font-bold text-muted-foreground">Поріг Max</span>
                                        <Input
                                          type="number"
                                          placeholder="Max"
                                          value={currentValues.max}
                                                                onChange={(e) =>
handleThresholdChange(metric.key, "max", e.target.value)}
                                                                className={`h-8 text-xs font-bold
rounded-lg bg-background ${isInvalid ? "border-red-500
focus-visible:ring-red-500" : ""}`}
                                          />
                                      </div>
                                    </div>
                                  );
                                })}
                              </div>
                            </div>
                          )}
                        </CardContent>
                      </Card>

                      <div className="flex justify-end pt-2">
                          <Button onClick={handleSave} className="gap-2 rounded-xl
h-10 px-5 font-bold shadow-md shadow-primary/10 transition-transform
active:scale-95">
                            <Save className="h-4 w-4" />
                            Зберегти налаштування SCADA
                          </Button>
                      </div>
                    </div>
                );
            }
```

Лістинг Г.6 – Програмна реалізація глобального сервіс-провайдера «AppProvider» із інтегрованими модулями MQTT-клієнта та мережевого Watchdog-таймера.

```
import React, { createContext, useContext, useState, useEffect, useRef } from "react";
import mqtt, { MqttClient } from "mqtt";
import { db } from "../../services/firebase";
import { ref, onValue, set } from "firebase/database";

import { METRIC_MAP } from "../data/metricCategories";

export interface Device {
  id: string;
  name: string;
  type: string;
  status: "online" | "offline" | "warning";
  lastUpdate: string;
  topic?: string;
  categoryId?: string;
  lastPacketTimestamp?: number;
  [key: string]: any;
}

interface AppContextType {
  devices: Device[];
  status: string;
   registerNewDevice: (id: string, name: string, topic: string, type: string) => Promise<void>;
  deleteDevice: (id: string) => Promise<void>;
}

const AppContext = createContext<AppContextType | undefined>(undefined);

export function AppProvider({ children }: { children: React.ReactNode }) {
  const [devices, setDevices] = useState<Device[]>([]);
  const [status, setStatus] = useState("Connecting...");
  const mqttClientRef = useRef<MqttClient | null>(null);

  useEffect(() => {
    const devicesDbRef = ref(db, 'devices');
     const unsubscribeFirebase = onValue(devicesDbRef, (snapshot) => {
      const data = snapshot.val();
      let deviceList: Device[] = [];

      if (data) {
        deviceList = Object.keys(data).map(key => {
          const item = data[key];
              const { id, name, type, status, topic, lastUpdate, categoryId, ...restFields } = item;

          const dynamicMetrics: Record<string, any> = {};
           Object.entries(restFields).forEach(([metricKey, value]) => {
```

```
                    if (METRIC_MAP[metricKey] !== undefined || metricKey
=== "temp" || metricKey === "hum") {
                      dynamicMetrics[metricKey] = value;
                    }
                  });

                  return {
                    id: key,
                    name: name || "Невідомий пристрій",
                    type: type || "Кліматичний датчик",
                    status: status || "offline",
                    topic: topic || "sensors/data",
                    categoryId: categoryId || "eco",
                    lastUpdate: lastUpdate || "немає даних",
                    lastPacketTimestamp: status === "online" || status ===
"warning" ? Date.now() : 0,
                    ...dynamicMetrics
                  };
                });
              } else {
                const defaultDevice: Device = {
                  id: "ESP32_Wokwi",
                  name: "Реальний ESP32 (Wokwi)",
                  type: "Температура & Вологість",
                  status: "offline",
                  categoryId: "eco",
                  lastUpdate: "немає даних",
                  topic: "sensors/data",
                  lastPacketTimestamp: 0
                };
                set(ref(db, 'devices/ESP32_Wokwi'), defaultDevice);
                deviceList = [defaultDevice];
              }

              setDevices(prevDevices =>
                deviceList.map(newDev => {
                        const existing = prevDevices.find(p => p.id ===
newDev.id);
                  if (existing) {
                       const { id, name, type, status, topic, lastUpdate,
categoryId, ...newMetrics } = newDev;
                      const { id: _, name: __, type: ___, status: ____,
topic: _____, lastUpdate: ______, categoryId: _______,
...existingMetrics } = existing;

                    return {
                      ...newDev,
                      ...existingMetrics,
                      ...newMetrics,
                                  status: existing.status !== "offline" ?
existing.status : newDev.status,
                       lastUpdate: existing.lastUpdate !== "немає даних" ?
existing.lastUpdate : newDev.lastUpdate,
                      lastPacketTimestamp: existing.lastPacketTimestamp ||
newDev.lastPacketTimestamp
                    };
                  }
                  return newDev;
```

```
            })
          );
        });

        return () => unsubscribeFirebase();
      }, []);

      useEffect(() => {
        const client = mqtt.connect('wss://broker.emqx.io:8084/mqtt',
{
                                clientId: 'react-global-ui-' +
Math.random().toString(16).substring(2, 8),
          keepalive: 60,
          reconnectPeriod: 1000
        });
        mqttClientRef.current = client;

        client.on('connect', () => {
          setStatus('Connected to Broker');
              console.log("[ФОНОВИЙ РОБОТ] Мережа активна. Оновлюємо
підписки на топіки...");
          devices.forEach(device => {
            if (device.topic) client.subscribe(device.topic);
          });
        });

        client.on('message', (topic, message) => {
          try {
            const msgStr = message.toString();

            if (msgStr === "offline" || msgStr === "disconnected") {
                  console.warn(`[MQTT LWT] Брокер зафіксував аварійне
відключення топіка: ${topic}`);
              setDevices(prevDevices =>
                  prevDevices.map(device => device.topic === topic ? {
...device, status: "offline", lastUpdate: "щойно" } : device)
              );
              return;
            }

            const parsedData = JSON.parse(msgStr);

            if (parsedData.status === "offline") {
              setDevices(prevDevices =>
                  prevDevices.map(device => (device.topic === topic ||
device.id === parsedData.device) ? { ...device, status: "offline",
lastUpdate: "щойно" } : device)
              );
              return;
            }

                                    const savedThresholds =
localStorage.getItem("scada_thresholds");
                        const thresholdLimits = savedThresholds ?
JSON.parse(savedThresholds) : null;

            const incomingMetrics: Record<string, any> = {};
            let isAnyMetricViolated = false;
```

```
            Object.entries(parsedData).forEach(([key, value]) => {
              if (METRIC_MAP[key] !== undefined) incomingMetrics[key]
= value;
                if (key === "temp") incomingMetrics["temperature"] =
value;
              if (key === "hum") incomingMetrics["humidity"] = value;

               const lookupKey = key === "temp" ? "temperature" : key
=== "hum" ? "humidity" : key;

                if (thresholdLimits && thresholdLimits[lookupKey] &&
typeof value === "number") {
                                                    const    minLimit    =
Number(thresholdLimits[lookupKey].min);
                                                    const    maxLimit    =
Number(thresholdLimits[lookupKey].max);

                if (value < minLimit || value > maxLimit) {
                  isAnyMetricViolated = true;
                }
              }
            });

            setDevices(prevDevices =>
              prevDevices.map(device => {
                        if (device.topic === topic || device.id ===
parsedData.device) {
                  return {
                    ...device,
                            status: isAnyMetricViolated ? "warning" :
"online",
                    lastUpdate: "щойно",
                    lastPacketTimestamp: Date.now(),
                    ...incomingMetrics
                  };
                }
                return device;
              })
            );
          } catch (error) {
              console.error("Помилка парсингу фонового повідомлення
телеметрії:", error);
            }
          });

          return () => {
            if (client) client.end();
          };
        }, [devices.length]);

        useEffect(() => {
          const watchdogTimer = setInterval(() => {
            const TIMEOUT_LIMIT = 12000;
            const now = Date.now();

            setDevices(prevDevices => {
              let hasChanges = false;
```

```
            const updatedDevices = prevDevices.map(device => {
                                if (device.status !== "offline" &&
device.lastPacketTimestamp) {
                                    const timeSinceLastPacket = now -
device.lastPacketTimestamp;

                if (timeSinceLastPacket > TIMEOUT_LIMIT) {
                                    console.error(`[WATCHDOG TIMEOUT] Пристрій
[${device.name}] мовчить ${Math.round(timeSinceLastPacket / 1000)}с.
Переводимо в OFFLINE.`);
                  hasChanges = true;
                  return {
                    ...device,
                    status: "offline" as const,
                    lastUpdate: "втрачено зв'язок"
                  };
                }
              }
              return device;
            });

            return hasChanges ? updatedDevices : prevDevices;
          });
        }, 4000);

        return () => clearInterval(watchdogTimer);
      }, []);

       const registerNewDevice = async (id: string, name: string,
topic: string, type: string) => {
        let categoryId = "eco";
        const lowerType = type.toLowerCase();

         if (lowerType.includes("ват") || lowerType.includes("енерг")
|| lowerType.includes("живл")) categoryId = "energy";
                        else if (lowerType.includes("двиг") ||
lowerType.includes("пром") || lowerType.includes("тиск")) categoryId =
"industrial";
                        else if (lowerType.includes("агро") ||
lowerType.includes("ґрунт") || lowerType.includes("теплиц")) categoryId
= "agro";
                       else if (lowerType.includes("мереж") ||
lowerType.includes("шлюз")) categoryId = "network";
                        else if (lowerType.includes("охр") ||
lowerType.includes("безпек") || lowerType.includes("замок")) categoryId
= "security";
                        else if (lowerType.includes("gps") ||
lowerType.includes("трек") || lowerType.includes("авто")) categoryId =
"gps";

        const newDeviceData = {
          name,
          type,
          topic,
          categoryId,
          status: "offline",
          lastUpdate: "щойно зареєстровано"
```

```
        };
        await set(ref(db, `devices/${id}`), newDeviceData);
      };

      const deleteDevice = async (id: string) => {
        try {
          await set(ref(db, `devices/${id}`), null);
          await set(ref(db, `history/${id}`), null);
           console.log(`[DESTRUCTOR] Пристрій ${id} успішно видалено з
хмари Firebase`);
        } catch (error) {
          console.error("Помилка видалення пристрою:", error);
        }
      };

      return (
                  <AppContext.Provider   value={{   devices,   status,
registerNewDevice, deleteDevice }}>
          {children}
        </AppContext.Provider>
      );
    }

    export const useApp = () => {
      const context = useContext(AppContext);
       if  (!context)  throw  new  Error("useApp  must  be  used  within
AppProvider");
      return context;
    };
```

Лістинг Г.7 – Програмна реалізація компонента «AlertsPanel» для динамічного відображення та кольорографічного кодування технологічних інцидентів SCADA.

```
    import  {  Card,  CardContent,  CardHeader,  CardTitle  }  from
"./ui/card";
    import { Alert, AlertDescription } from "./ui/alert";
    import { ScrollArea } from "./ui/scroll-area";

    interface AlertItem {
      id: string;
      type: "info" | "warning" | "error" | "success";
      message: string;
      timestamp: string;
      device?: string;
    }

    interface AlertsPanelProps {
      alerts: AlertItem[];
    }

    export function AlertsPanel({ alerts }: AlertsPanelProps) {

      const getAlertClass = (type: string) => {
        switch (type) {
          case "warning":
             return "border-yellow-300 bg-yellow-50/70 text-yellow-900
dark:border-yellow-500/30 dark:bg-yellow-500/10 dark:text-yellow-200";
          case "error":
```

```
                    return "border-red-300 bg-red-50/70 text-red-900
dark:border-red-500/30 dark:bg-red-500/10 dark:text-red-200";
            case "success":
                 return "border-green-300 bg-green-50/70 text-green-900
dark:border-green-500/30 dark:bg-green-500/10 dark:text-green-200";
            default:
                  return "border-blue-300 bg-blue-50/70 text-blue-900
dark:border-blue-500/30 dark:bg-blue-500/10 dark:text-blue-200";
          }
        };

        const getSubtextClass = (type: string) => {
          switch (type) {
                        case "warning": return "text-yellow-700/80
dark:text-yellow-400/70";
            case "error": return "text-red-700/80 dark:text-red-400/70";
                         case "success": return "text-green-700/80
dark:text-green-400/70";
            default: return "text-blue-700/80 dark:text-blue-400/70";
          }
        };

        return (
          <Card className="h-full flex flex-col justify-between">
            <CardHeader className="pb-3">
                           <CardTitle className="text-lg font-bold
tracking-tight">Журнал подій SCADA</CardTitle>
            </CardHeader>
            <CardContent className="flex-1">
              <ScrollArea className="h-[430px] pr-2">
                <div className="space-y-3">
                  {alerts.map((alert) => (
                    <Alert
                      key={alert.id}
                       className={`transition-all duration-300 shadow-sm
border rounded-xl p-4 ${getAlertClass(alert.type)}`}
                    >
                      <AlertDescription>
                        <div className="flex flex-col space-y-1.5">
                                <div className="text-base font-semibold
leading-snug tracking-tight">
                            {alert.message}
                          </div>

                                    <div className={`text-xs font-medium
${getSubtextClass(alert.type)}`}>
                                                {alert.device &&
<span>{alert.device}</span>}
                                              {alert.device && <span
className="mx-1.5">•</span>}
                            <span>{alert.timestamp}</span>
                          </div>
                        </div>
                      </AlertDescription>
                    </Alert>
                  ))}
                </div>
              </ScrollArea>
```

```
        </CardContent>
      </Card>
    );
}
```