ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод організації зв'язку для керування та моніторингу статусу IoT пристроїв»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Богдан ГАВРИЛЕНКО
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти
група ІСД-42 Богдан ГАВРИЛЕНКО

Керівник:
науковий ступінь, вчене звання Віктор САГАЙДАК
доктор філософії

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ - 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ

Гавриленко Богдан Валерійович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Метод організації зв'язку для керування та моніторингу статусу IoT пристроїв »

керівник кваліфікаційної роботи Віктор САГАЙДАК, доктор філософії

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 20__ р. № ___

2. Строк подання кваліфікаційної роботи «___» _____ 20__

3. Вихідні дані до кваліфікаційної роботи: Аналіз протоколів зв'язку та хмарних платформ у сфері Інтернету речей, вивчення апаратної архітектури ESP32 та реалізація зчитування сенсорних даних, побудова та тестування системи моніторингу IoT-пристрою з передачею телеметрії до ThingsBoard через MQTT.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз технологій IoT та методів передачі даних
2. Розробка методу організації зв'язку

3. Апробація методу на практичному прикладі

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Формулювання теми, мети, завдань роботи та ознайомлення з вимогами до оформлення	01.05 – 03.05.2025	Виконав
2	Аналіз технологій IoT, протоколів зв’язку (MQTT, HTTP, CoAP) та вибір апаратної платформи	04.05 – 06.05.2025	Виконав
3	Розробка технічного завдання, створення архітектури та реалізація коду в Arduino IDE	07.05 – 10.05.2025	Виконав
4	Побудова симуляції в Wokwi, з’єднання з ThingsBoard, формування JSON-структури	11.05 – 13.05.2025	Виконав
5	Тестування, збір результатів, візуалізація на дашборді, аналіз затримок	14.05 – 16.05.2025	Виконав
6	Оформлення тексту дипломної роботи, написання вступу, висновків, підготовка до захисту	17.05 – 21.05.2025	Виконав

Здобувач вищої освіти _____ Богдан ГАВРИЛЕНКО

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віктор САГАЙДАК

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра : 55 сторінок, 33 рисунків, 4 таблиць, 21 джерел.

Об'єкт дослідження – процес взаємодії IoT-пристрою з хмарною платформою через бездротове з'єднання.

Предмет дослідження – методи організації передачі телеметрії IoT-пристроїв до хмарного середовища без використання локального доступу та складної мережевої інфраструктури.

Мета роботи – розробити та апробувати метод інтеграції MQTT-протоколу в IoT-систему для забезпечення стабільного та безпечного зв'язку з хмарною платформою ThingsBoard, із використанням мікроконтролера ESP32 та віртуального симулятора Wokwi.

Питання побудови ефективної системи зв'язку для IoT-пристроїв є актуальним у контексті цифровізації, зростання мобільних рішень та автоматизації моніторингу навколишнього середовища. Стандартні методи часто передбачають складні мережеві налаштування або потребують фіксованої IP-адреси, що обмежує їх застосування у динамічному середовищі.

У цій роботі запропоновано метод інтеграції MQTT для організації стабільного зв'язку з хмарною платформою без втручання в маршрутизатор чи відкриття портів. Проведено апробацію методу через створення тестової IoT-системи, яка передає дані із сенсорів температури, вологості, освітленості та аналогових значень у реальному часі.

Галузь використання – Інформаційні системи та технології, Інтернет речей, автоматизовані системи моніторингу.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, IOT, MQTT, ESP32, ХМАРНА ПЛАТФОРМА, THINGSBOARD, ТЕЛЕМЕТРІЯ, СЕНСОРИ, ARDUINO IDE, БЕЗДРОТОВИЙ ЗВ'ЯЗОК, ПРОТОКОЛ ПЕРЕДАЧІ ДАНИХ, ЕМУЛЯЦІЯ, ВІЗУАЛІЗАЦІЯ ДАНИХ, JSON, WOKWI

Abstract

Text part of the bachelor level qualification work: 55 pages, 33 figures, 4 tables, 21 sources.

Object of research – the process of interaction between an IoT device and a cloud platform via wireless communication.

Subject of research – methods of organizing telemetry transmission from IoT devices to the cloud without using local infrastructure or static IP addressing.

Purpose of the work – to develop and test a method for integrating the MQTT protocol into an IoT system using the ESP32 microcontroller and the cloud platform ThingsBoard, with the implementation carried out in the Wokwi online emulator.

The relevance of this topic lies in the need to simplify the architecture of IoT systems that operate beyond local networks. Traditional solutions require router access and complex configuration, which limits the scalability and usability of such systems.

This work proposes an integration method that ensures reliable data exchange between IoT devices and a cloud platform using MQTT. A test system was developed to validate the method, demonstrating successful data acquisition from sensors (temperature, humidity, light, analog values) and transmission to ThingsBoard for real-time visualization.

Field of application – Information Systems and Technologies, Internet of Things, Automated Monitoring Systems.

KEYWORDS: INTERNET OF THINGS, IOT, MQTT, ESP32, CLOUD PLATFORM, THINGSBOARD, TELEMETRY, SENSORS, ARDUINO IDE, WIRELESS COMMUNICATION, DATA PROTOCOL, SIMULATION, JSON, WOKWI.

ЗМІСТ

ВСТУП

У сучасному світі Інтернет речей (Internet of Things, IoT) стрімко змінює підходи до автоматизації, моніторингу та обміну даними між фізичними об’єктами. Від побутових приладів до складних промислових систем дедалі частіше впроваджуються інтелектуальні пристрої, здатні не лише збирати інформацію про навколишнє середовище, а й активно взаємодіяти між собою та з користувачем. Така динаміка розвитку обумовлює зростання потреби у доступних, надійних і масштабованих рішеннях для організації зв’язку між IoT-пристроями та хмарними платформами.

Особливої актуальності набувають задачі підключення пристроїв, розміщених поза межами локальних мереж, де немає можливості легко організувати прямий доступ або призначити статичну IP-адресу. Більшість традиційних способів підключення потребують втручання в налаштування маршрутизатора (порт-форвардинг), використання динамічних DNS-сервісів або створення VPN-з’єднань. Для пересічного користувача це створює суттєві технічні бар’єри, які ускладнюють впровадження IoT-рішень у повсякденному середовищі.

Водночас зростає популярність хмарних IoT-платформ, таких як ThingsBoard, Ubidots, Blynk та інші, що забезпечують інфраструктуру для збору, обробки й візуалізації даних у реальному часі. Завдяки підтримці сучасних протоколів, зокрема MQTT, відкривається можливість забезпечити двосторонній зв’язок між пристроєм і хмарною платформою навіть за умов обмеженого мережевого доступу. MQTT, як легкий брокерський протокол, особливо добре підходить для пристроїв із обмеженими апаратними ресурсами, забезпечуючи при цьому стабільну й ефективну передачу даних.

У зв’язку з цим виникає потреба в побудові IoT-систем, які здатні функціонувати поза межами локальної мережі без складних технічних маніпуляцій. Це розширює можливості застосування таких рішень у сферах агромоніторингу, екологічного нагляду, «розумного дому», мобільних сенсорних

платформ тощо. Саме така концепція лежить в основі виконання даної бакалаврської кваліфікаційної роботи.

Мета роботи – розробити та реалізувати метод організації зв’язку для керування та моніторингу статусу IoT-пристроїв через хмарну платформу ThingsBoard із використанням мікроконтролера ESP32 та MQTT-протоколу, орієнтованого на позамережеву роботу пристрою.

Завдання роботи:

- Провести аналіз сучасних протоколів передавання даних у IoT-системах (MQTT, HTTP, CoAP) та порівняти їх ключові особливості.
- Дослідити архітектуру та технічні можливості мікроконтролера ESP32 у контексті побудови IoT-пристрою.
- Обґрунтувати вибір хмарної платформи для передавання та обробки телеметрії.
- Реалізувати програмну частину пристрою для зчитування даних із сенсорів і формування повідомлень у форматі JSON.
- Налаштувати підключення до MQTT-брокера ThingsBoard Cloud і перевірити стабільність передавання даних.
- Побудувати дашборд для візуалізації показників у режимі реального часу.
- Провести тестування розробленої системи в емуляційному середовищі Wokwi.

Об’єкт дослідження - процес взаємодії IoT-пристрою з хмарною інфраструктурою через бездротове з’єднання.

Предмет дослідження - методи та протоколи передачі телеметричних даних IoT-пристроїв до хмарної платформи без використання локального доступу.

Методи дослідження - теоретичний аналіз протоколів зв’язку та хмарних сервісів, порівняльне дослідження апаратних платформ, імітаційне моделювання, програмна реалізація у середовищі Arduino IDE, тестування в емуляторі Wokwi, візуалізація результатів у середовищі ThingsBoard.

Наукова новизна роботи полягає в тому, що запропоновано метод організації зв’язку між IoT-пристроєм та хмарною платформою, який не потребує відкриття портів або використання статичних IP-адрес. Обґрунтовано вибір MQTT як оптимального протоколу для задач моніторингу, а також продемонстровано можливість побудови IoT-архітектури з мінімальними вимогами до мережевої інфраструктури. Це відкриває нові можливості для створення масштабованих, доступних та надійних IoT-систем.

Результати роботи можуть бути адаптовані для побудови прикладних систем моніторингу в аграрній галузі, системах домашньої автоматизації та екологічного контролю. Реалізований прототип є гнучким, легко масштабується та придатний до використання в мобільних або польових умовах без складних налаштувань і потреби у спеціалізованій технічній підтримці.

Апробація результатів. Основні положення і результати бакалаврської роботи доповідались на:

- VI Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». ДУІКТ, 2024.

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

1.1 Поняття Інтернету речей (IoT)

Інтернет речей (IoT, Internet of Things) — це концепція побудови мережі фізичних пристроїв, оснащених вбудованими сенсорами, мікроконтролерами, модулями зв’язку та програмним забезпеченням, які здатні збирати, передавати та обробляти дані без участі людини. Сукупність технологій IoT дозволяє створити розумне середовище, де пристрої можуть взаємодіяти між собою та з користувачем для підвищення ефективності, безпеки й комфорту.

IoT-пристрої зазвичай складаються з наступних елементів:

- Сенсори та виконавчі механізми (актуатори): забезпечують фізичне виявлення змін у навколишньому середовищі та/або реагування на них;
- MCU або шлюзи (Gateway): контролюють збирання, попередню обробку та передавання даних;
- Протоколи передачі: забезпечують передачу даних в хмару через MQTT, HTTPS, CoAP, REST тощо;
- Хмарна інфраструктура: обробляє, зберігає і візуалізує дані, дозволяє керувати пристроями віддалено [1].

Чотиришарова архітектура IoT-системи є загальновизнаною структурною моделлю, що відображає поетапний процес трансформації даних — від їх первинного збору до застосування в інтелектуальних сервісах (рис. 1.1). Вона включає такі рівні:

- Sensing Layer (рівень збору даних): фізичні пристрої, сенсори й актуатори, які безпосередньо взаємодіють із середовищем. Вони здійснюють вимірювання параметрів та генерують телеметричну інформацію.
- Network Layer (мережевий рівень): відповідає за передачу даних між сенсорами та обчислювальними центрами. Використовуються мережеві шлюзи, інтернет-з'єднання, протоколи зв'язку (Wi-Fi, ZigBee, LTE, LoRa тощо).

- Data Processing Layer (рівень обробки даних): тут відбувається фільтрація, агрегування, аналіз та прийняття рішень на основі зібраної інформації. Це може бути реалізовано як на локальному шлюзі, так і в хмарі.
- Application Layer (рівень застосування): забезпечує інтерфейс з користувачем або сторонніми системами. Тут реалізуються функції моніторингу, керування, візуалізації та автоматизації рішень [2].

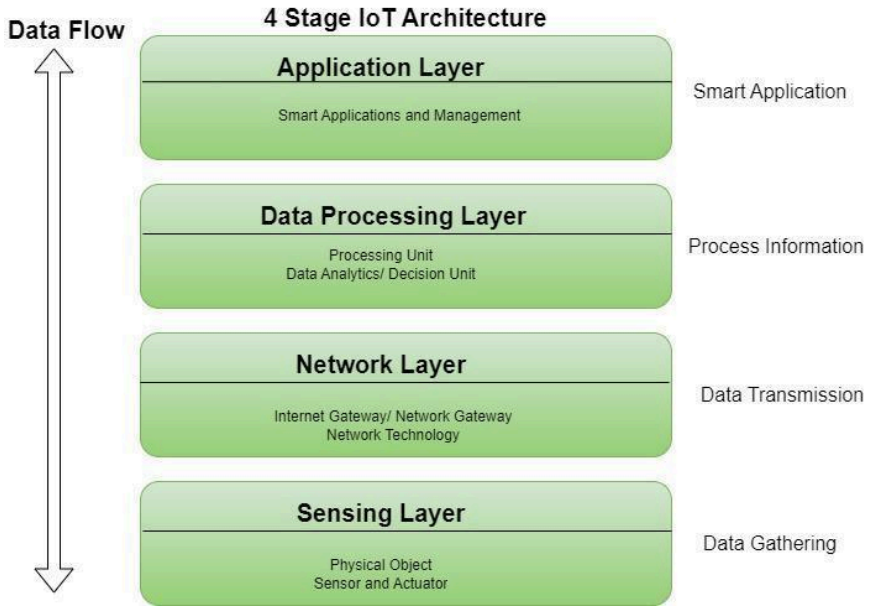


Рис. 1.1 Загальна архітектура IoT [2]

Можливості застосування IoT надзвичайно широкі: розумне місто, розумний дім, промислова автоматизація, охорона здоров'я, сільське господарство, логістика тощо. У центрі кожної IoT-системи знаходиться мікроконтролер або мікропроцесор, який координує роботу пристрою та обробляє дані з датчиків, передаючи їх до хмарного сервісу або іншого пристрою. Можна стверджувати, що в майбутньому кількість IoT-пристроїв перевищить кількість людей на планеті в кілька разів, оскільки навіть побутові речі стануть «розумними».

Технологічний стек Інтернету речей охоплює низку апаратних, програмних і мережевих компонентів, що забезпечують повноцінну роботу IoT-рішення — від фізичного пристрою до аналітики в хмарі. Його структура зазвичай поділяється на п'ять взаємопов'язаних рівнів, як показано на рис. 1.2.

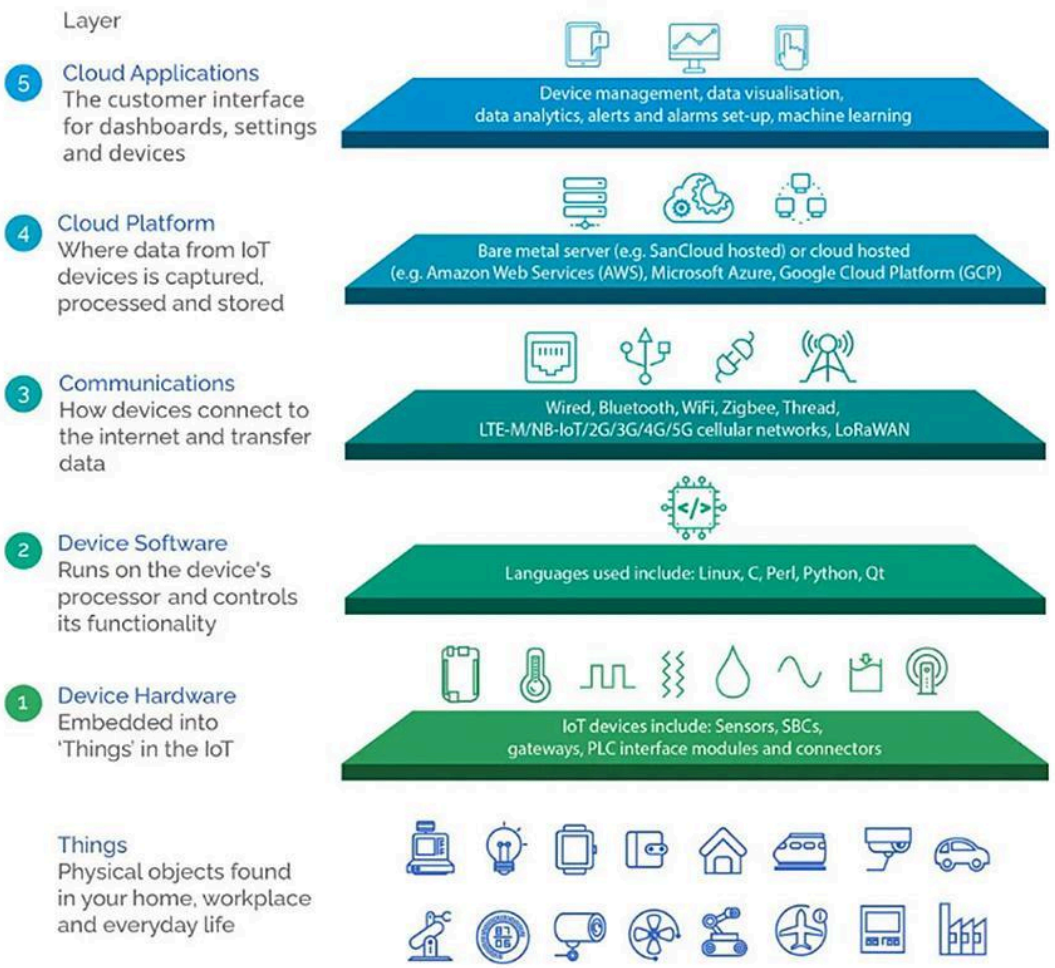


Рис. 1.2 Технологічний стек IoT [3]

На базовому рівні знаходяться фізичні пристрої, до яких належать сенсори, виконавчі механізми, плати управління, одноплатні комп’ютери або контролери. Саме ці пристрої забезпечують зв’язок із реальним середовищем та ініціюють збір даних. Наступним шаром є програмне забезпечення пристроїв, яке реалізується у вигляді драйверів, прошивок та прикладних програм. Воно забезпечує зчитування параметрів, початкову обробку та формування повідомлень, часто написане мовами C, Python або працює під керуванням Linux. Далі слідує мережевий рівень, що відповідає за передавання даних від пристроїв до обчислювальних центрів. Тут використовуються як фізичні канали (Wi-Fi, LTE, Ethernet, LoRa), так і протоколи передачі (MQTT, HTTPS, WebSocket). Обробка, агрегація та зберігання даних здійснюється на рівні хмарної платформи, який представлений серверною або хмарною інфраструктурою — наприклад, Amazon AWS IoT, Microsoft Azure,

Google Cloud або ThingsBoard. Завершальним є рівень хмарних застосунків, де реалізується інтерфейс користувача: інформаційні панелі, аналітика, оповіщення, дистанційне керування та інші сервіси. Тут також можуть застосовуватись технології штучного інтелекту й машинного навчання.

Технологічний стек формує основу функціонування IoT-системи, де кожен рівень підтримує наступний: від збору даних до прийняття рішень і реалізації реакцій. Його грамотне проектування забезпечує надійність, масштабованість та адаптивність рішення до конкретних потреб користувача.

З огляду на стрімкий розвиток цифрових технологій, концепція Інтернету речей набуває дедалі більшого значення в різних сферах діяльності — від побутових систем до промислових комплексів. IoT-технології не лише забезпечують нові можливості автоматизації та контролю, а й формують основу для побудови інтелектуального середовища, де пристрої самостійно приймають рішення на основі даних. Їх масштабованість, енергоефективність та здатність до інтеграції з хмарними сервісами роблять IoT-рішення надзвичайно перспективними в умовах зростаючих вимог до гнучкості, мобільності та зниження витрат.

1.2 Протоколи передачі даних в IoT-середовищі

Однією з ключових складових IoT-системи є спосіб передачі даних від пристрою до центрального вузла або сервера. На практиці розробниками використовуються різноманітні протоколи передачі даних: HTTP, MQTT, CoAP, AMQP, WebSocket та інші. Вибір залежить від вимог до енергоспоживання, надійності та швидкодії.

Найбільш поширеним для IoT-проектів є протокол MQTT (Message Queuing Telemetry Transport). Це легкий, публічно-доступний протокол обміну повідомленнями на основі архітектури "видавець-підписник". MQTT має мінімальні накладні витрати, що робить його ідеальним для систем з обмеженими

ресурсами. Крім того, MQTT підтримує збереження сеансу, рівні якості доставки, захист SSL/TLS, а також добре масштабується у великих мережах.

MQTT (Message Queue Telemetry Transport) — це легкий мережевий протокол, який функціонує поверх стеку TCP/IP та призначений для обміну повідомленнями між пристроями за схемою «видавець–підписник». Його головна перевага полягає в мінімальному навантаженні на мережу та простоті реалізації на пристроях з обмеженими ресурсами.

Протокол був створений у 1999 році доктором Енді Станфорд-Кларком (IBM) та Арленом Ніппером (Arcom) і поширювався за ліцензією без роялті. У 2014 році специфікацію MQTT 3.1.1 було офіційно стандартизовано міжнародним консорціумом OASIS.

З розвитком технологій з’явилася полегшена модифікація для сенсорних мереж — MQTT-SN, яка функціонує на основі UDP або Bluetooth, що робить її придатною для використання в бездротових середовищах з обмеженою пропускною здатністю.

У 2019 році консорціум OASIS затвердив нову редакцію протоколу — MQTT 5.0, що включила розширені можливості керування повідомленнями, властивості сеансів, покращення безпеки та більшу гнучкість у налаштуванні параметрів доставки.

Можливості протоколу MQTT:

- Простота використання. Має легку структуру, легко вбудовується в різні системи.
- Гнучка робота з даними. Дозволяє передавати повідомлення довільного формату без попереднього опису.
- Зручне адміністрування. Не потребує складних налаштувань або підтримки.
- Ефективність. Мінімальне навантаження на мережу завдяки компактності протоколу.

- Стійкість до збоїв. Працює навіть при нестабільному з’єднанні, підтримує відновлення.
- Універсальність. Підтримує текстові, бінарні, JSON-дані та інші формати.

Принцип дії протоколу MQTT ґрунтується на моделі "видавець–брокер–підписник", що забезпечує гнучкий та масштабований механізм обміну повідомленнями. Як показано на рис. 1.4, сенсори або інші пристрої (видавці) надсилають дані до центрального посередника — MQTT-брокера. Брокер приймає повідомлення від усіх видавців і, відповідно до тем (topics), розсилає їх тим клієнтам, які на ці теми підписані (підписникам). Завдяки цьому підхід забезпечує розподілену та ефективну взаємодію між компонентами IoT-системи, оскільки видавцям не потрібно знати, хто саме отримуватиме їхні дані. Така архітектура значно знижує навантаження на мережу, підвищує відмовостійкість системи та дозволяє легко масштабувати її шляхом додавання нових підписників або видавців без зміни структури брокера [4].

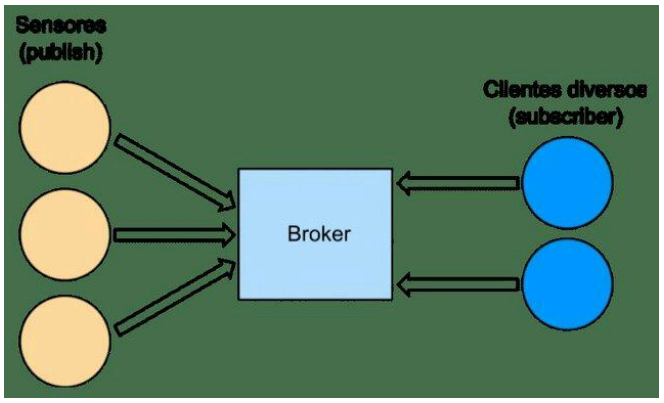


Рис. 1.4 MQTT Broker [5]

У протоколі MQTT передбачено набір методів (так званих «дієслів»), що визначають дії, які клієнт може виконувати відносно брокера або ресурсу, з яким встановлюється зв’язок. Під ресурсом у цьому випадку розуміється умовна точка обміну даними, яка може відповідати файлу, сервісу або результату виконання певної логіки на сервері. Різні реалізації брокерів можуть по-різному

інтерпретувати ці ресурси, однак основна функціональність методів зберігається спільною.

Процес, показаний схематично на рис. 1.5, починається з ініціалізації з'єднання через метод Connect, який встановлює TCP/IP-сесію між клієнтом та брокером. Після цього клієнт може здійснити підписку на одну або декілька тем за допомогою методу Subscribe, що дозволяє отримувати повідомлення, опубліковані іншими пристроями в рамках відповідних тем. У разі потреби припинити отримання повідомлень від певних тем використовується метод UnSubscribe, який ініціює відписку клієнта від зазначених каналів. Передача даних у систему здійснюється за допомогою методу Publish — повідомлення публікується у відповідній темі, після чого брокер автоматично розсилає його всім клієнтам, які на цю тему підписані. Завершення сеансу зв'язку ініціюється через метод Disconnect, який закриває з'єднання після завершення всіх активних процесів обміну. Усі ці дії виконуються асинхронно, що забезпечує швидкість і надійність роботи в умовах нестабільного або енергозалежного середовища, характерного для IoT-пристроїв [4].

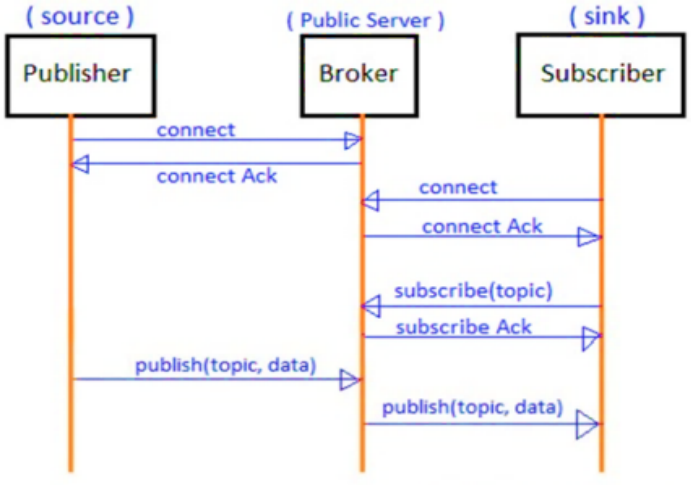


Рис. 1.5 Схема роботи MQTT з брокером, видавцем і підписником [6]

HTTP (Hypertext Transfer Protocol) — прикладний мережевий протокол, що використовується для передачі гіпертекстових документів, зображень, застосунків та інших файлів у комп'ютерних мережах. Належить до 7-го рівня моделі OSI.

Протокол працює за принципом «запит–відповідь», де клієнтом найчастіше виступає веб браузер. HTTP дозволяє передавати файли в різних форматах, підтримує зазначення кодування та мови за допомогою MIME-типів.

Типовий порт для HTTP — 80, а для захищеної версії HTTPS — 443. У порівнянні з FTP або SMTP, HTTP простіший в інтеграції, але менш оптимізований для IoT-пристроїв, особливо в умовах нестабільного зв'язку (рис. 1.6).

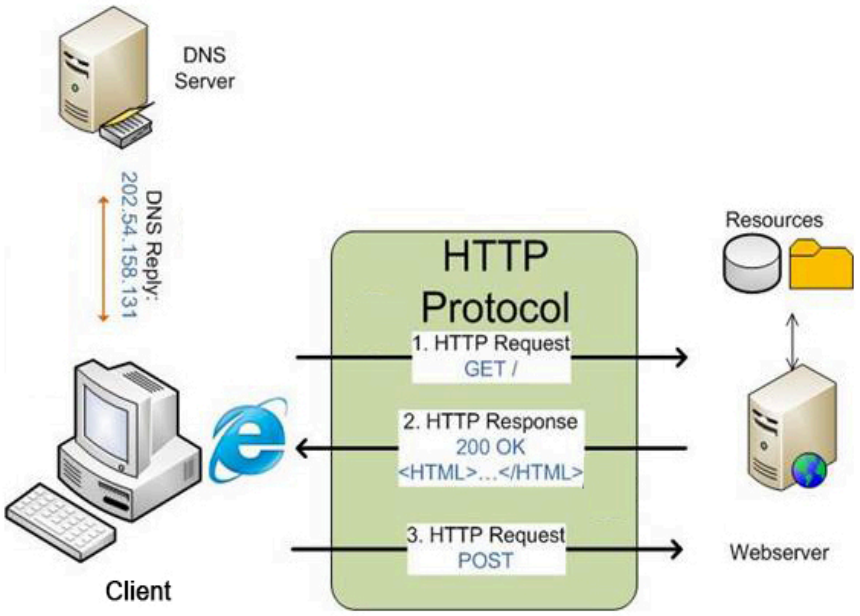


Рис. 1.6 Схема HTTP – клієнт надсилає запит, сервер відповідає [11]

HTTP використовує глобальні URI для ідентифікації ресурсів. Протокол є безстанним (stateless) — не зберігає інформацію про попередні запити, що відрізняє його від багатьох інших протоколів [7]. Уся взаємодія між клієнтом і

сервером відбувається у форматі окремих пар «запит–відповідь», без внутрішньої підтримки сесійного зв’язку.

Збереження контексту (наприклад, IP-адрес або заголовків) може виконуватись на рівні реалізації клієнта чи сервера, однак сама специфікація цього не вимагає. Це дозволяє зробити HTTP універсальним, але потребує додаткових засобів для підтримки стану (наприклад, cookies або токенів автентифікації).

Кожен HTTP-запит або відповідь складається з:

- стартового рядка (метод або статус-код),
- заголовків (headers),
- тіла повідомлення, що може містити запитувані або повернуті дані.

Починаючи з версії HTTP/1.1, обов’язковим є заголовок **Host**, який дозволяє обслуговувати кілька доменів на одній IP-адресі [7].

CoAP (Constrained Application Protocol) — це спеціалізований прикладний протокол, призначений для використання у мережах з обмеженими ресурсами, таких як сенсорні вузли, мікроконтролери, розумні пристрої тощо.
Він був розроблений IETF (Internet Engineering Task Force) як альтернатива HTTP для потреб Інтернету речей.

Головною метою CoAP є забезпечення ефективної взаємодії між IoT-пристроями, які мають обмежені обчислювальні можливості, обсяг пам’яті та джерела живлення.

CoAP використовує протокол транспортного рівня UDP замість TCP (який є стандартом для HTTP), що дозволяє зменшити накладні витрати і забезпечити швидке встановлення з’єднання [12].

Як і HTTP, CoAP підтримує моделі клієнт-сервер і методи GET, POST, PUT, DELETE, але робить це у компактній формі (рис. 1.7).

Рис. 1.7 Порівняння мережевих стеків протоколів CoAP та HTTP [13]

Повідомлення в CoAP мають невеликий розмір, що дозволяє ефективно передавати їх навіть у вузькосмугових або нестабільних мережах.

Для протоколу CoAP характерна низка особливостей, що вирізняють його серед інших IoT-протоколів:

- Мала вага повідомлень — CoAP використовує бінарну структуру заголовків, яка займає мінімум пам'яті.
- Підтримка спостереження (Observe) — клієнт може підписатися на ресурс, і сервер надсилатиме оновлення без повторних запитів.
- Multicast — протокол дозволяє одночасно надіслати повідомлення кільком пристроям.
- Простота трансляції у HTTP — є можливість шлюзування запитів між CoAP і HTTP-серверами.

Протокол CoAP використовується у тих проєктах, у яких потрібна швидка передача коротких повідомлень між пристроями, які не потребують складних сценаріїв обміну. Типові приклади:

- Розумне освітлення (керування групами світильників через multicast);
- Моніторинг навколишнього середовища (температура, вологість, забруднення повітря);
- Інфраструктура розумного міста (паркування, датчики руху, контроль шуму);
- Системи безпеки (виявлення руху, обмеження доступу) [12].

У табл. 1.1 наведено результати порівняння найбільш поширених протоколів передачі даних.

Таблиця 1.1

Основні властивості протоколів передачі даних для IoT

Протокол	Легкість	Надійність	Підтримка Qos	Захист	Використання в IoT
MQTT	Висока	Висока	Так	TLS/SSL	Широко застосовується
HTTP	Середня	Висока	Ні	TLS	Використовується обмежено
CoAP	Висока	Середня	Частково	DTLS	Застосовується в сенсорних мережах

Найбільш збалансованим варіантом для організації зв'язку між IoT-пристроями та хмарними сервісами є MQTT, що поєднує в собі легкість, надійність і гнучкість. Протоколи HTTP та CoAP можуть використовуватись у специфічних умовах або в комбінації з MQTT, однак останній на сьогодні залишається найбільш універсальним і практичним рішенням для побудови енергоефективних і масштабованих IoT-систем.

1.3 Апаратні платформи для IoT-проектів

Серед найбільш популярних апаратних рішень для IoT можна виокремити такі мікроконтролери, як Arduino Uno/Nano, ESP8266, ESP32, Raspberry Pi, STM32 тощо. Обираючи платформу для реалізації IoT-системи, важливо враховувати обчислювальні ресурси, підтримку протоколів зв'язку (Wi-Fi, Bluetooth, LoRa), енергоспоживання, кількість портів введення/виведення.

Arduino Uno — популярна плата з відкритим апаратним і програмним забезпеченням, побудована на мікроконтролері ATmega328P. Вона містить усе необхідне для роботи: 14 цифрових входів/виходів (з яких 6 підтримують ШІМ), 6

аналогових входів, кварцовий резонатор на 16 МГц, USB-інтерфейс, роз’єм живлення, ICSP-порт і кнопку скидання.

Плату можна живити як через USB, так і від зовнішнього джерела (адаптер або батарея), що робить її зручною для початківців і прототипування в умовах обмежених ресурсів [8].

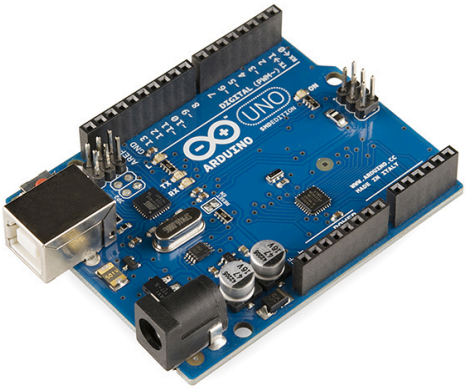


Рис. 1.8 – Плата Arduino Uno R3 [8]

На відміну від попередніх плат Arduino, Uno використовує вбудований мікроконтролер ATmega16U2 як перетворювач USB–UART, що замінив класичну мікросхему FTDI. У версії R2 додано резистор, який підтягує до землі лінію HWB мікроконтролера 8U2, що спрощує оновлення прошивки [8].

Альтернативою Arduino Uno в контексті IoT-проєктів є ESP8266 — мікроконтролер із вбудованим Wi-Fi-модулем, розроблений компанією Espressif Systems (рис. 1.9). Пристрій здатен виконувати програми безпосередньо з зовнішньої флеш-пам’яті через інтерфейс SPI, що розширює його можливості як автономного пристрою з бездротовим підключенням [14].

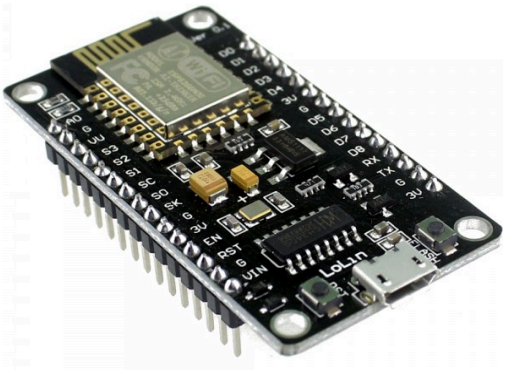


Рис. 1.9 Плата мікроконтролера ESP8266 [14]

ESP8266 став популярним у 2014 році завдяки появі доступних модулів із вбудованим Wi-Fi за вкрай низькою ціною [14]. Пристрій швидко здобув популярність серед розробників рішень для Інтернету речей. Надалі, у 2016 році, компанія Espressif Systems випустила ESP8285 — оновлену версію, що поєднує в одному корпусі мікроконтролер та флеш-пам’ять обсягом 1 МБ.

У 2015 році було анонсовано нове покоління — ESP32 [15]. Це високофункціональний мікроконтролер серії «система на кристалі» (SoC), що має вбудовані модулі Wi-Fi та Bluetooth (dual-mode), знижене енергоспоживання і значно розширений функціонал. ESP32 побудований на базі процесора Tensilica Xtensa LX6 (одно- або двоядерного), має вбудовані антени, підсилювач потужності, малошумний приймач, RF-балун та модулі управління живленням. Він розроблений Espressif Systems (Китай) і виготовляється компанією TSMC.

ESP32 є прямим наступником ESP8266 та призначений для створення складніших та продуктивніших IoT-рішень.

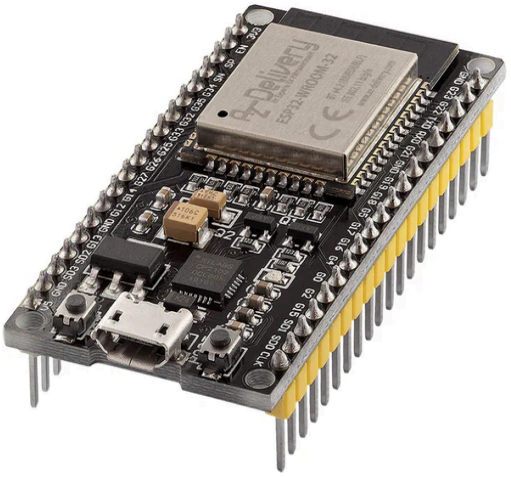


Рис. 1.10 Плата ESP32 [15]

Під час розробки системи моніторингу для IoT-пристроїв одним із ключових етапів стало визначення оптимальної апаратної платформи. Серед можливих варіантів розглядалися такі популярні мікроконтролери, як Arduino Uno, ESP8266 та ESP32 (результати порівняння наведені на рис. 1.11). Зрештою, вибір зупинився на ESP32, що було зумовлено рядом технічних і практичних переваг.

SPECS/BOARD	ESP32	ESP8266	ARDUINO UNO
Number of Cores	2	1	1
Architecture	32 Bit	32 Bit	8 Bit
CPU Frequency	160 MHz	80 MHz	16 MHz
WiFi	YES	YES	NO
BLUETOOTH	YES	NO	NO
RAM	512 KB	160 KB	2 KB
FLASH	16 MB	16 MB	32 KB
GPIO PINS	36	17	14
Busses	SPI, I2C, UART, I2S, CAN	SPI, I2C, UART, I2S	SPI, I2C, UART
ADC Pins	18	1	6
DAC Pins	2	0	0

Рис. 1.11 Порівняння ESP32 з іншими платформами

Arduino Uno, хоча й відома своєю простотою у використанні та широкою спільнотою, має суттєві обмеження — зокрема, відсутність вбудованого модуля

Wi-Fi, обмежений обсяг пам’яті (2 КБ SRAM) і порівняно низьку тактову частоту (16 МГц). Для реалізації бездротового зв’язку на цій платформі потрібне окреме обладнання, наприклад, модуль ESP8266 або Ethernet Shield, що ускладнює схему та підвищує енергоспоживання.

ESP8266 є популярним рішенням у сфері IoT завдяки наявності вбудованого Wi-Fi та невисокій вартості. Проте, його архітектура одноядерна, обсяг оперативної пам’яті обмежений, а кількість доступних вхідних/вихідних портів недостатня для складніших конфігурацій. Крім того, ESP8266 не підтримує Bluetooth і має менший набір периферійних інтерфейсів порівняно з новішими моделями.

Натомість ESP32 забезпечує значно ширший функціонал. Це 32-бітний мікроконтролер з двома ядрами, високою тактовою частотою до 240 МГц та великою кількістю вбудованої пам’яті. Він має інтегровані модулі Wi-Fi та Bluetooth, що дозволяє реалізовувати гнучкі схеми зв’язку. Завдяки широкому набору периферійних інтерфейсів ESP32 підтримує одночасну роботу з кількома сенсорами, а режим енергозбереження дозволяє використовувати його в автономних пристроях.

З урахуванням поєднання обчислювальної потужності, енергоефективності, наявності бездротових інтерфейсів та ціни, саме ESP32 було обрано як базову платформу для реалізації системи моніторингу IoT-пристроїв у межах даної роботи.

1.4 Хмарні сервіси для візуалізації та обробки даних

Одним з важливих компонентів IoT-систем є серверна частина — платформи, які приймають, зберігають та візуалізують дані. Найбільш поширеними хмарними сервісами є платформи ThingsBoard, Blynk, Ubidots, Adafruit IO, Google Firebase [16].

Ubidots — це хмарна IoT-платформа, орієнтована переважно на швидку розробку прототипів, а також комерційне використання в малому та середньому

бізнесі. Користувацький інтерфейс рішення показано на рис. 1.12. Ubidots дозволяє розробникам зручно надсилати телеметричні дані з мікроконтролерів (ESP32, Arduino, Raspberry Pi тощо), будувати графіки, керувати пристроями та створювати дашборди з інтерактивними віджетами [17].

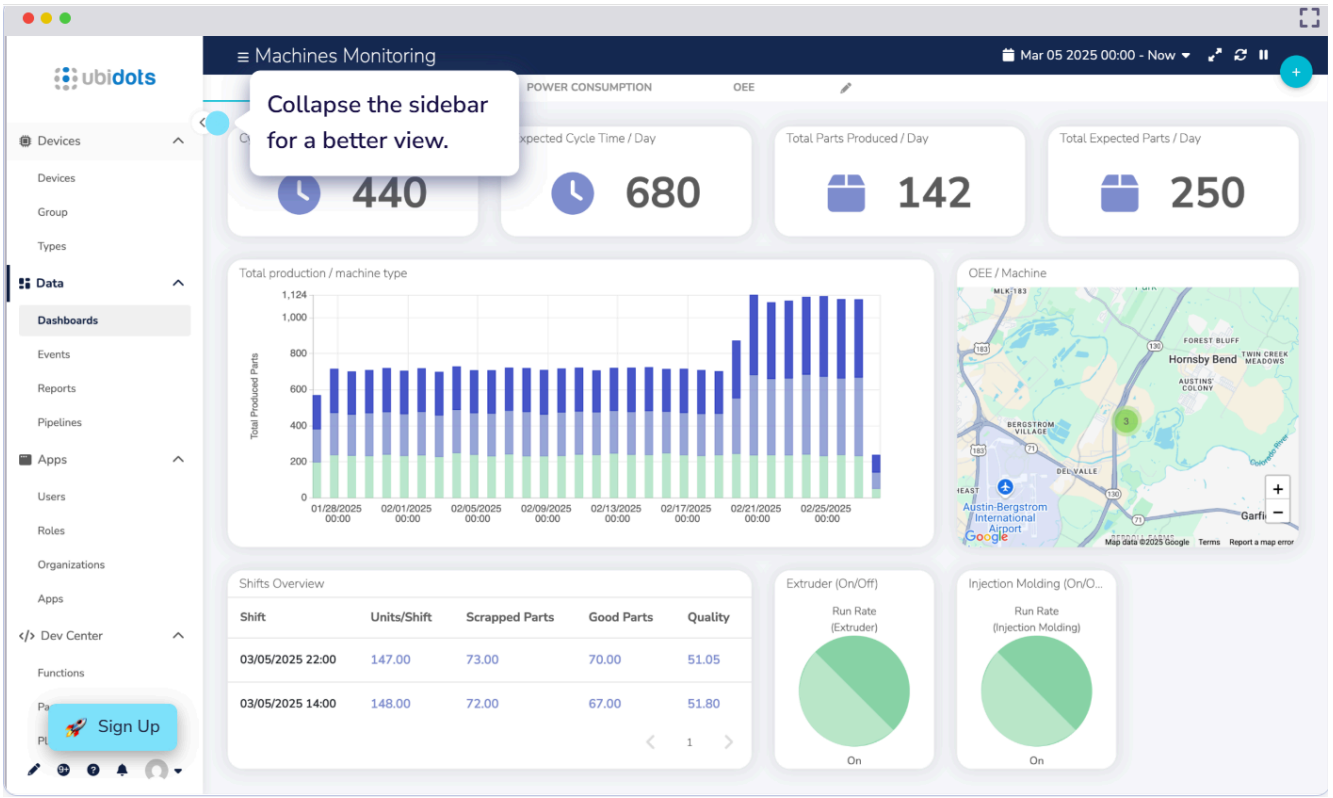


Рис. 1.12 Інтерфейс Ubidots [17]

Платформа підтримує найпоширеніші протоколи IoT — MQTT, HTTP, TCP/UDP — і надає інтуїтивно зрозумілий веб-інтерфейс. Однією з її сильних сторін є візуальний конструктор дашбордів, що дозволяє створювати готові інтерфейси моніторингу без глибоких знань у програмуванні.

Ubidots надає обмежену безкоштовну версію, яка підходить для навчальних або дослідницьких цілей, однак має обмеження за кількістю подій, пристроїв та обсягом збережених даних. Для промислових рішень передбачені платні тарифи з розширеним функціоналом, включно з API-інтеграцією, звітами та підтримкою користувачів [17].

Blynk — це кросплатформна IoT-платформа, орієнтована на швидке створення мобільних інтерфейсів для керування мікроконтролерами, такими як Arduino, ESP8266, ESP32 тощо [18]. Основною особливістю Blynk є те, що вона дозволяє створювати мобільні застосунки без написання коду для інтерфейсу, використовуючи графічний конструктор (рис. 1.13).

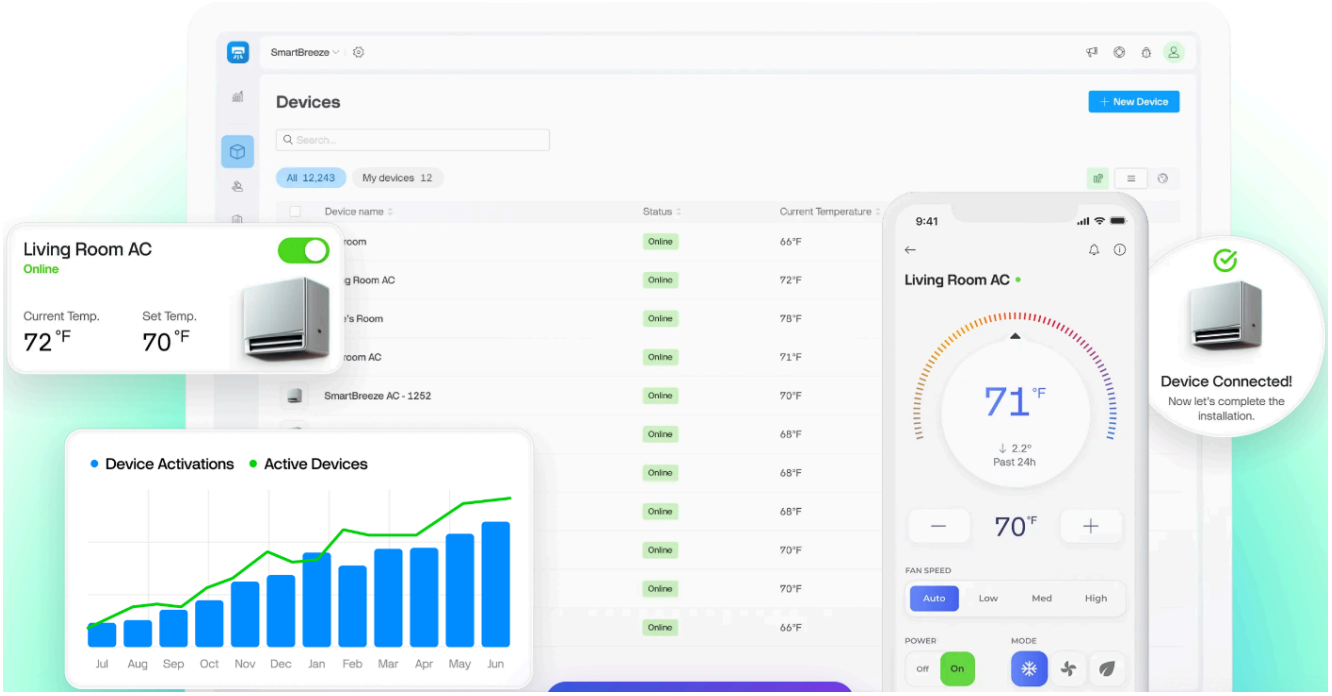


Рис. 1.13 Інтерфейс Blynk [18]

Платформа працює за клієнт-серверною архітектурою, де пристрій надсилає дані на Blynk-сервер, а мобільний додаток отримує їх або віддає команди. Комунікація здійснюється через пропрієтарний протокол Blynk (TCP/HTTP), що забезпечує стабільну інтеграцію з мобільними пристроями.

Серед переваг Blynk — простота використання, візуальна привабливість дашбордів, велика бібліотека віджетів, а також наявність мобільного застосунку, який дозволяє миттєво отримувати дані з пристрою або керувати ним з будь-якої точки світу.

Обмеженням Blynk є те, що безкоштовна версія має значні обмеження у функціоналі (зокрема, кількість доступних віджетів і пристроїв), а у порівнянні з

ThingsBoard чи Ubidots Blynk менш придатний для складних промислових сценаріїв, оскільки більше орієнтований на прототипування та хобі-проєкти [18].

ThingsBoard — це масштабована, розширювана платформа з відкритим кодом, призначена для збору, обробки, візуалізації та керування телеметричними даними з IoT-пристроїв (рис. 1.14). Вона підтримує декілька протоколів зв'язку, зокрема MQTT, HTTP, CoAP, що робить її універсальним рішенням як для розробки прототипів, так і для промислових систем [10].

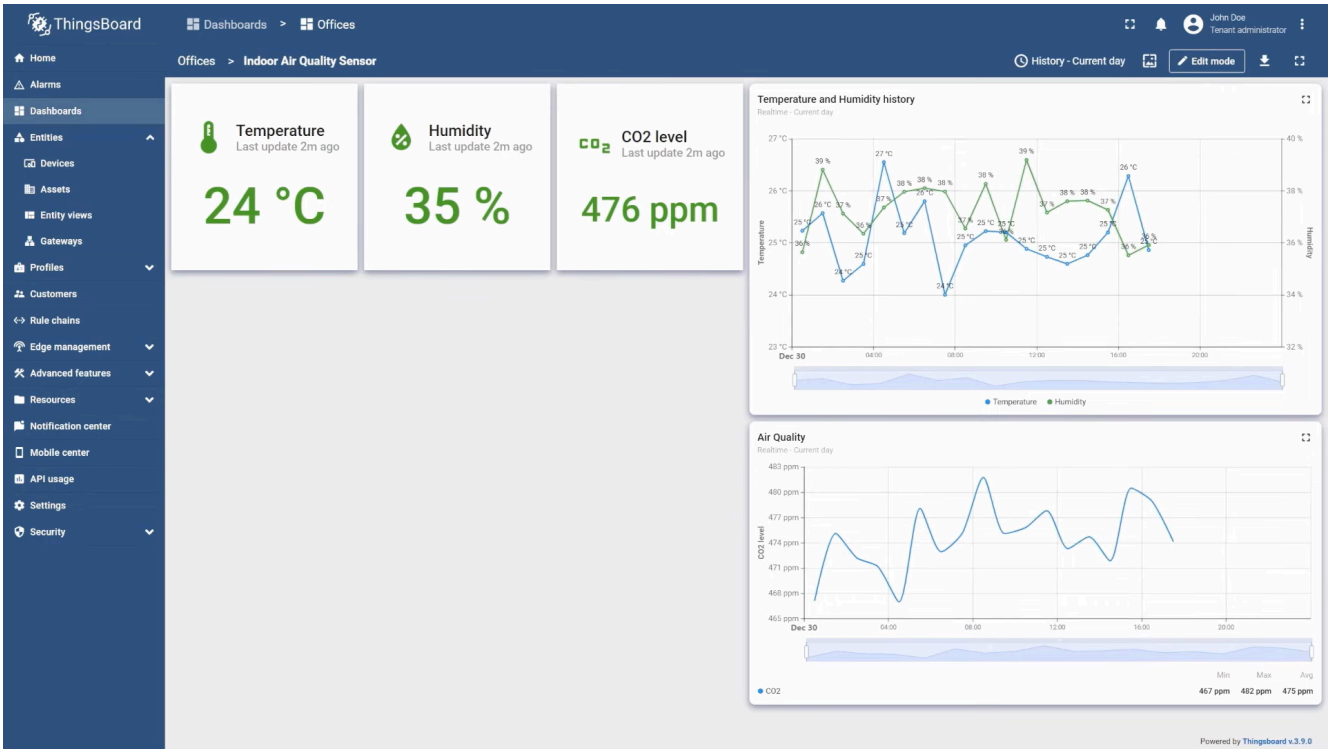


Рис. 1.14 Інтерфейс ThingsBoard [10]

ThingsBoard дозволяє зручно організовувати пристрої за допомогою унікальних токенів автентифікації, створювати складні дашборди з багатьма віджетами, будувати правила обробки даних, надсилати команди на пристрої (RPC) та зберігати історичну інформацію для подальшого аналізу.

Однією з переваг платформи є її гнучка архітектура, що дозволяє розгорнути ThingsBoard як у вигляді хмарного сервісу (наприклад, на eu.thingsboard.cloud), так і локально на сервері або через Docker. Це забезпечує високу адаптивність до різних проєктів та рівнів безпеки.

Платформа має зручний графічний інтерфейс, який дає змогу швидко налаштовувати відображення телеметрії, фільтрувати дані, формувати звіти й візуалізації.Також підтримується рольова модель доступу, що дозволяє організувати багаторівневу взаємодію між користувачами системи [10].

Результати порівняння платформ для забезпечення обміну даними із серверною частиною IoT-систем наведено у табл. 1.2.

Таблиця 1.2

Порівняльна таблиця платформ для візуалізації IoT-даних

Платформа	**Протоколи**	**Безкоштовна версія**	**Віджети**	**Хостинг**
ThingBoard	MQTT, HTTP, CoAP	Так	Багато	Обидва варіанти
Blynk	HTTP, WebSocket	Так(обмежено)	Обмежено	Хмара
Ubidots	MQTT, HTTP	Так(обмежено)	Багато	Хмара

У рамках даної кваліфікаційної роботи було обрано платформу ThingsBoard Cloud — open-source середовище для обробки телеметрії, створення дашбордів, віддаленого моніторингу та керування пристроями. Платформа підтримує MQTT, HTTP, CoAP, має зручний інтерфейс, рольову модель доступу та можливості інтеграції. Важливою перевагою є можливість створення індивідуальних віджетів, правил обробки подій та розгортання як у хмарі, так і локально. Це робить її надзвичайно гнучкою в реалізації систем будь-якої складності.

РОЗДІЛ 2 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА МЕТОДУ ОРГАНІЗАЦІЇ ЗВ'ЯЗКУ В ІОТ-СИСТЕМІ

2.1 Модель методу організації зв'язку для IoT-системи моніторингу

У сучасних IoT-системах, що використовуються для моніторингу фізичних параметрів, ключовою проблемою залишається забезпечення стабільного та масштабованого обміну даними між пристроями і хмарними сервісами. Традиційні підходи, які базуються на запитно-відповідних моделях або прямому HTTP-з'єднанні, не завжди відповідають вимогам реального часу, енергоефективності та гнучкого розширення інфраструктури. У цьому контексті виникає необхідність у методі, який дозволяє ефективно організувати зв'язок за умов обмежених ресурсів, втрат з'єднання та високої частоти передачі даних.

Запропонований у даній роботі роботі метод має умовну новизну, що полягає у формалізованій структурі топіків, використанні логіки відновлення з'єднання та інтеграції з відкритими хмарними платформами. Його модель розроблена з урахуванням властивостей MQTT-протоколу та специфіки типової архітектури IoT-систем, що дозволяє легко масштабувати рішення, адаптувати його до різних типів сенсорів і зберігати цілісність телеметрії при нестабільному каналі зв'язку.

Для систематизації, подальшого аналізу та реплікації запропонованого підходу було сформульовано модель методу організації зв'язку в IoT-системі моніторингу.

Запропонована модель методу організації зв'язку в IoT-системі базується на поєднанні MQTT-протоколу, хмарної платформи ThingsBoard та логіки структурованої передачі телеметрії від периферійного пристрою до хмарного середовища. В основі моделі лежить концепція асинхронної публікації даних сенсорів із використанням моделі «видавець–брокер–підписник», яка є типовою для MQTT-зв'язку.

Візуальне представлення моделі методу наведено на рис. 2.1.

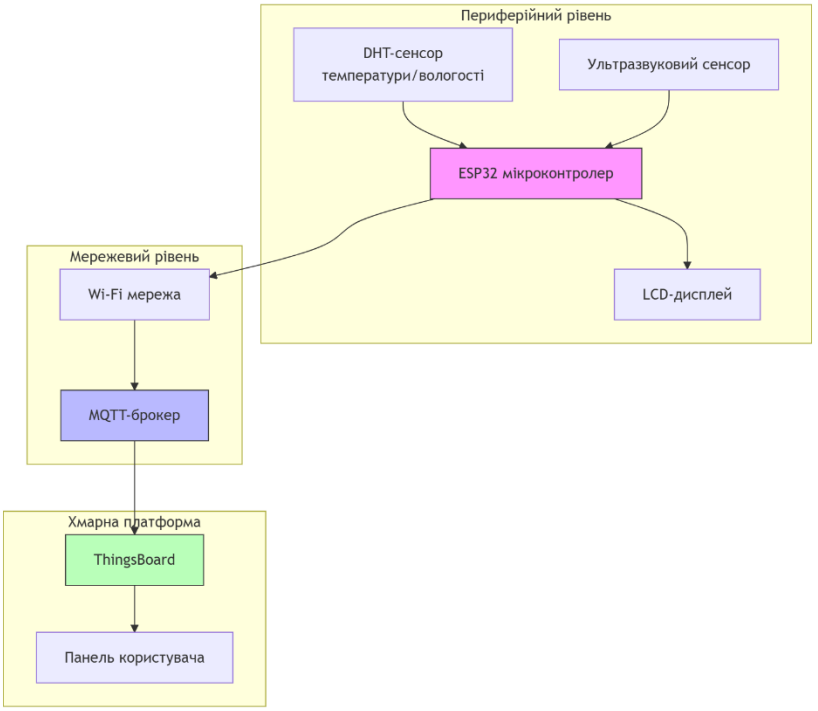


Рис. 2.1 Архітектура моделі методу організації зв’язку в IoT-системі моніторингу

Система поділяється на три функціональні рівні: периферійний, мережевий та хмарний. Периферійний рівень складається з мікроконтролера ESP32, підключених до нього сенсорів (температури, вологості та відстані) і пристрою локального виведення інформації — LCD-дисплея. На мережевому рівні забезпечується з’єднання з MQTT-брокером через Wi-Fi, що дозволяє здійснювати передачу даних у форматі publish/subscribe. Хмарний рівень реалізовано на основі платформи ThingsBoard, яка виконує функції обробки, зберігання та візуалізації телеметричних даних, доступних користувачеві через вебінтерфейс.

Процес обміну даними у межах запропонованої моделі реалізується за допомогою MQTT-протоколу, що ґрунтується на моделі «видавець–брокер–підписник». Як відображено на рис. 2.2, мікроконтролер ESP32 виступає у ролі видавця, який ініціює передачу телеметричних даних у вигляді повідомлень до MQTT-брокера.

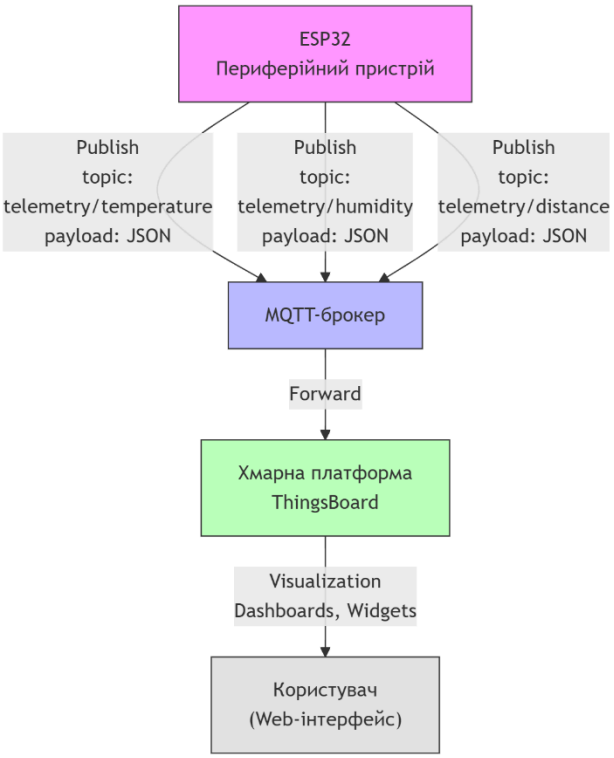


Рис. 2.2 Взаємодія компонентів у рамках MQTT-моделі методу зв’язку

Для кожного типу даних (температура, вологість, відстань) сформовано окремий топік із payload у форматі JSON. MQTT-брокер отримує ці повідомлення і перенаправляє їх до хмарної платформи ThingsBoard, де вони обробляються, зберігаються та відображаються у вигляді дашбордів і візуальних віджетів. Кінцевий користувач взаємодіє з даними через веб-інтерфейс хмарної системи, отримуючи доступ до актуальної інформації у зручному вигляді.

На рис. 2.3 представлено алгоритм забезпечення стійкості зв’язку (fault-tolerance) у межах запропонованої моделі. На початку кожного циклу роботи пристрою виконується перевірка доступності Wi-Fi-з’єднання. У разі його відсутності викликається функція reconnect(), яка ініціює повторне підключення до точки доступу. Після успішного з’єднання відбувається перевірка доступності MQTT-брокера. Якщо з’єднання втрачено, виконується команда tb.connect() для повторного встановлення MQTT-сесії. У разі активного з’єднання з брокером здійснюється передача телеметрії та підтримка сесії за допомогою функції

tb.loop(), яка гарантує стабільність комунікації протягом усього життєвого циклу пристрою.

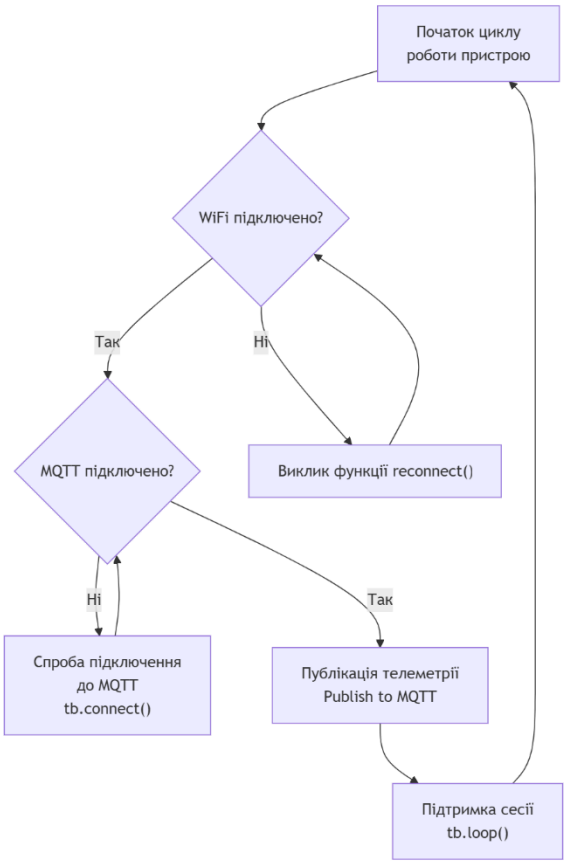


Рис. 2.3 Алгоритм забезпечення стійкості зв’язку в моделі методу (fault-tolerance)

На відміну від простих схем надсилання даних через HTTP, дана модель забезпечує сталість з’єднання, оптимізацію затримок і підвищену надійність завдяки реалізації логіки перевірки доступності з’єднання та повторних підключень. Обмін даними здійснюється в структурованому вигляді (формат JSON), а кожен параметр (температура, вологість, відстань) передається в окремому ключі повідомлення, що полегшує подальшу агрегацію та візуалізацію даних у хмарному інтерфейсі.

Особливістю моделі є використання ієрархічної системи обміну, при якій кожна тема MQTT формально відповідає конкретному параметру, джерелу даних або пристрою. Це дає змогу масштабувати систему без зміни основної логіки методу, додаючи нові пристрої або показники шляхом розширення тем. Крім того,

у моделі передбачено використання вбудованих механізмів контролю втрати з’єднання (через методи connect() та loop()), що забезпечує стійкість передачі в умовах нестабільного Wi-Fi-сигналу.

Модель передбачає не лише фізичну передачу даних, а й реалізацію структурованої логіки організації обміну, яка є адаптованою до ресурсно обмежених IoT-пристроїв, сумісною з популярними хмарними платформами та такою, що може бути масштабована для розподілених систем моніторингу.

2.2 Архітектура та складові експериментальної IoT-системи

На етапі розробки системи моніторингу було обрано архітектуру, що базується на використанні мікроконтролера ESP32 як центрального вузла. До ESP32 підключені кілька сенсорів для збору даних з навколишнього середовища:

- датчик температури та вологості DHT22;
- ультразвуковий датчик відстані HC-SR04;
- рідкокристалічний дисплей LCD1602.

DHT22 (датчик температури та вологості), показаний на рис. 2.4, — цифровий сенсор, який дозволяє вимірювати температуру в діапазоні від –40 °C до +80 °C з точністю ±0.5 °C, а також вологість від 0 % до 100 % з точністю до ±2–5 % [19]. Сенсор забезпечує вихідні дані у цифровому форматі, що спрощує його інтеграцію з мікроконтролером ESP32 без потреби в зовнішньому аналого-цифровому перетворенні. У даній системі DHT22 використовується для постійного моніторингу мікрокліматичних умов, зокрема температури й вологості повітря, що є важливою частиною телеметрії.

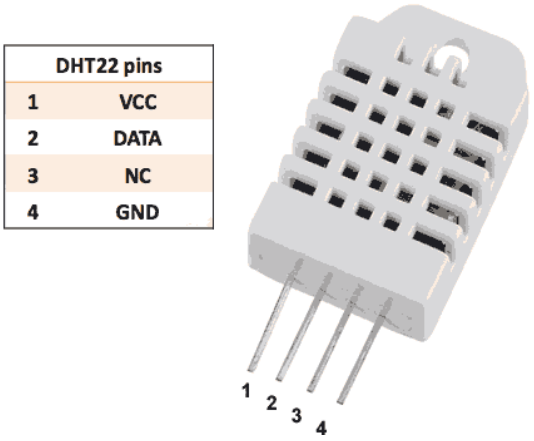


Рис. 2.4 Датчик температури та вологості DHT22 [19]

HC-SR04 (ультразвуковий датчик відстані) — недорогий сенсор, який вимірює відстань до об'єкта за допомогою ультразвукових хвиль (рис. 2.5). Він має два основні елементи: передавач (trig) і приймач (echo). Діапазон вимірювання становить приблизно від 2 до 400 см, з роздільною здатністю близько 0.3 см [20]. У системі моніторингу HC-SR04 виконує функцію визначення відстані до об’єкта, що може застосовуватись, наприклад, для контролю рівня заповнення або наявності перешкод у зоні спостереження.

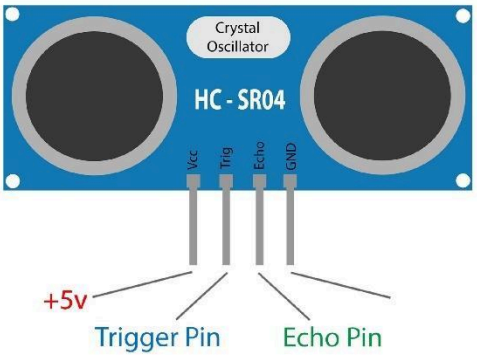


Рис. 2.5 Ультразвуковий датчик відстані HC-SR04 [20]

LCD1602 (рідкокристалічний дисплей) — текстовий дисплей із двома рядками по 16 символів (рис. 2.6). Він дозволяє відображати основні параметри, що вимірюються системою, без необхідності використання комп’ютера або зовнішнього клієнта [21]. У даній реалізації LCD використовується для локальної індикації температури, вологості та інших показників, що підвищує зручність

користування й дозволяє оперативно реагувати на зміну умов у режимі реального часу.

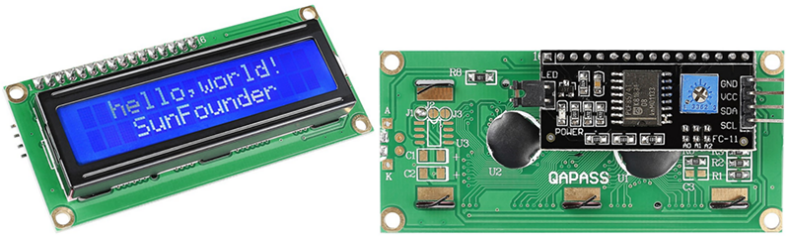


Рис. 2.6 Рідкокристалічний дисплей LCD1602 [21]

Для моделювання та первинної реалізації апаратної частини IoT-системи моніторингу було обрано онлайн-симулятор Wokwi. Це середовище дає змогу створювати, запускати й тестувати мікроконтролерні проєкти без необхідності фізичного збирання пристрою, що є особливо зручним на етапі прототипування та в умовах обмеженого доступу до реального обладнання. Платформа підтримує велику кількість популярних контролерів (зокрема ESP32), сенсорів та інтерфейсів, забезпечуючи високу точність симуляції сигналів і логіки обробки. Крім того, Wokwi дозволяє швидко змінювати конфігурацію схеми, аналізувати поведінку окремих компонентів та інтегрувати прошивку в реальному часі, що значно пришвидшує цикл розробки. Використання цього інструменту також дає змогу легко реплікувати та демонструвати роботу системи, не залежачи від фізичної платформи [9].

Монтажну схему підключення компонентів системи у Wokwi показано на рис. 2.7.

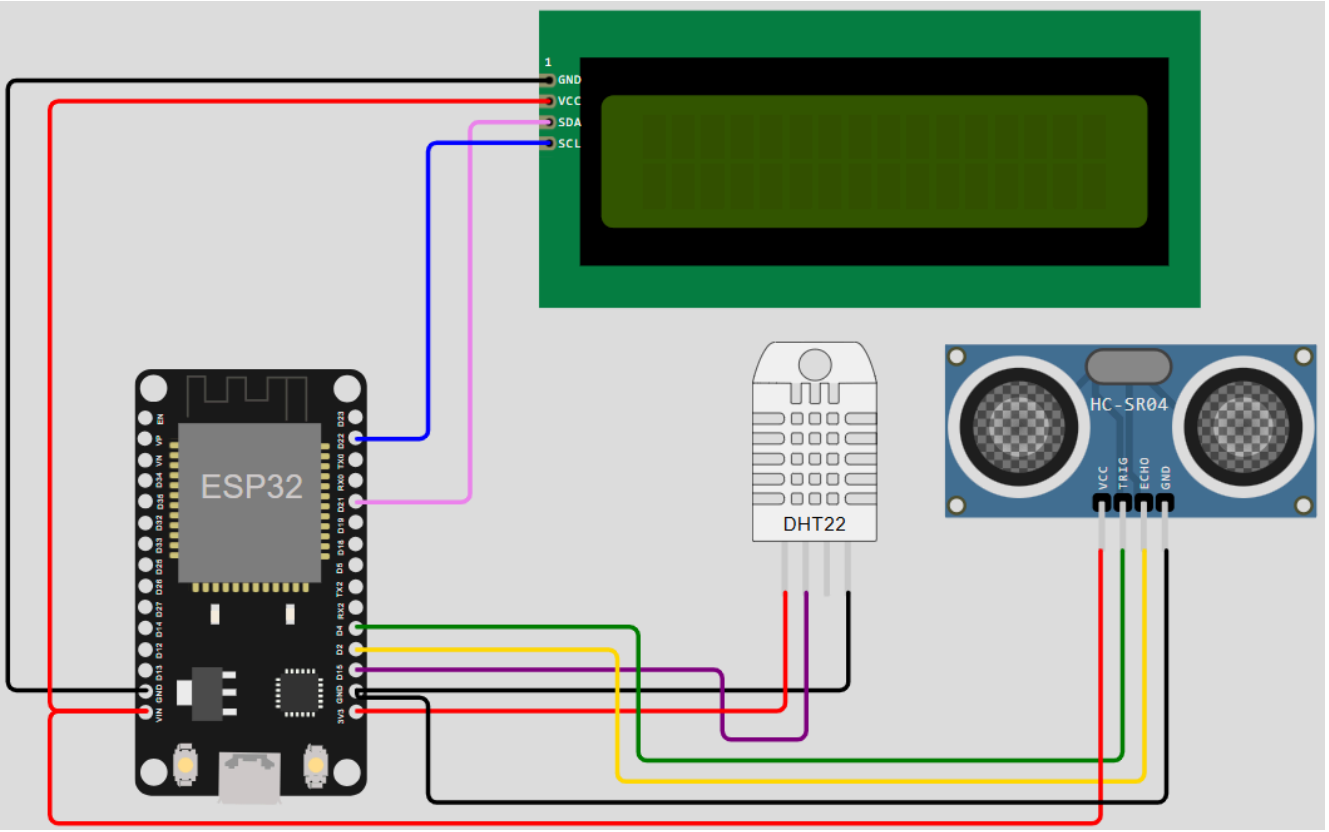


Рис. 2.7 Підключення компонентів системи тестування методу організації зв’язку в IoT

У системі моніторингу фізичні компоненти підключено до мікроконтролера ESP32 відповідно до їхньої функціональності та вимог щодо інтерфейсу обміну даними.

DHT22 підключено до цифрового піну GPIO15. Сенсор передає дані про температуру і вологість у цифровому форматі за допомогою власного однодротового протоколу, який потребує короткого періоду опитування після ініціалізації. Вихідний сигнал формується у вигляді послідовності імпульсів, що кодують бітову інформацію, яка потім обробляється бібліотекою DHTesp.

HC-SR04 використовує два цифрових пінів ESP32: GPIO2 (echo) і GPIO4 (trig). Для визначення відстані ESP32 посилає короткий імпульс на вхід trig, після чого очікує на відгук на виводі echo. Тривалість цього сигналу пропорційна відстані до об'єкта й обчислюється у мікросекундах. Цей спосіб передачі є імпульсно-широтною модуляцією (PWM) і вимагає точного вимірювання часу за допомогою мікроконтролера.

LCD1602 із модулем I2C підключено через пари контактів SDA і SCL, які з’єднані з GPIO21 та GPIO22 ESP32 відповідно. Передача даних відбувається за допомогою I2C-шини, що дозволяє значно зменшити кількість фізичних з’єднань у порівнянні з паралельним підключенням. Передача здійснюється послідовно, у вигляді байтів даних та команд, які дисплей інтерпретує для виводу тексту на екран.

Архітектура апаратного забезпечення розробленої IoT-системи показана на рис. 2.8.

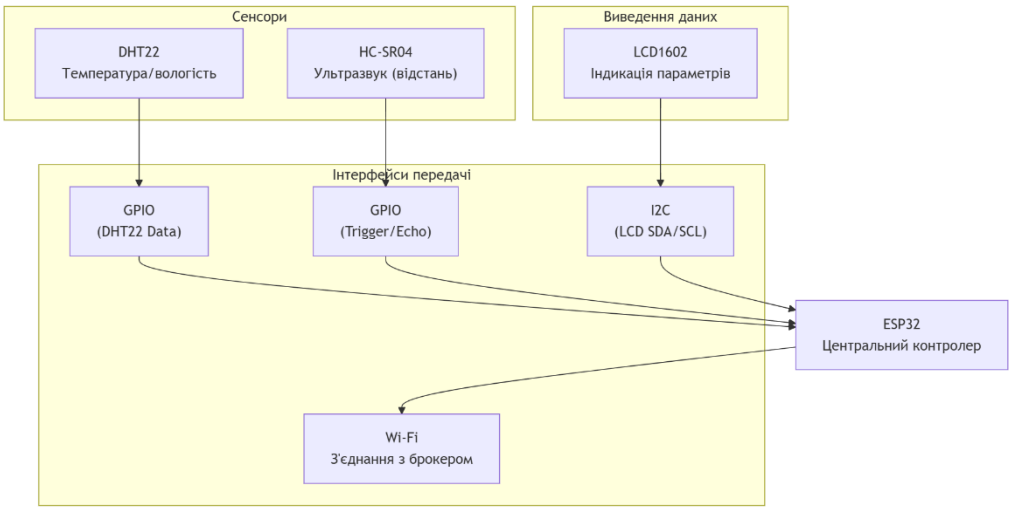


Рис. 2.8 Архітектура апаратної частини IoT-системи моніторингу

Загалом, передача даних у цій системі реалізована через комбінацію цифрового імпульсного зв’язку (HC-SR04), однодротового цифрового протоколу (DHT22) та послідовної I2C-комунікації (LCD1602), що дозволяє ефективно використовувати ресурси ESP32 та забезпечує надійне зчитування й відображення інформації.

2.3 Програмна реалізація експериментальної системи

Програмне забезпечення IoT-системи реалізоване мовою C++ у середовищі симуляції Wokwi. Код забезпечує зчитування даних із сенсорів, локальне

відображення інформації на LCD-дисплеї та передачу телеметрії на хмарну платформу ThingsBoard за допомогою MQTT-протоколу. Логіку роботи програми показано на рис. 2.9.

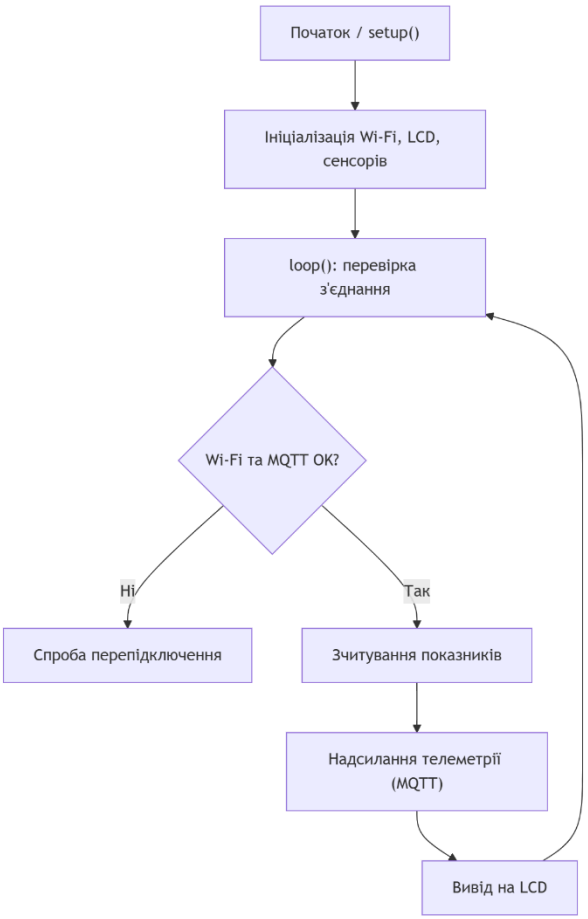


Рис. 2.9 Узагальнена схема логіки роботи прошивки експериментальної IoT-системи

У початковій частині програми здійснюється підключення необхідних бібліотек для роботи з мережевим з’єднанням (WiFi), сенсорами (DHT22, HC-SR04), хмарною платформою (ThingsBoard) та дисплеєм (LCD1602 через інтерфейс I2C). Одночасно з цим задаються ключові параметри конфігурації: ідентифікатори Wi-Fi мережі, токен пристрою, адреса MQTT-брокера, параметри UART-комунікації, а також пін-коди для підключення сенсорів.

На наступному етапі програма оголошує змінні для керування периферійними пристроями, створює об’єкти необхідних бібліотек та ініціалізує

нестандартний символ для відображення градуса Цельсія на екрані. У функціях InitWiFi() та reconnect() реалізовано логіку підключення та автоматичного перепідключення до Wi-Fi мережі. У разі втрати зв’язку ESP32 самостійно ініціює повторне з’єднання, що є частиною fault-tolerant логіки всієї системи.

У функції setup() здійснюється базова ініціалізація апаратних компонентів: дисплей активується й виводить коротке привітальне повідомлення, налаштовуються порти вводу/виводу для ультразвукового сенсора, а також запускається датчик температури та вологості. Після цього система переходить до циклічного виконання основної логіки.

Функція loop() виконується безперервно, з періодичністю в одну секунду. У кожному циклі перевіряється наявність Wi-Fi-з’єднання та підключення до брокера ThingsBoard. У разі відсутності з’єднання ініціюється перепідключення. Після цього пристрій зчитує відстань до об’єкта за допомогою ультразвукового сенсора, а також значення температури та вологості з DHT22. Ці показники формуються у форматі JSON і передаються на хмарну платформу через функції sendTelemetryData() або її варіанти для числових значень. Паралельно дані виводяться на LCD-дисплей у зрозумілому користувачеві вигляді.

Для забезпечення стабільної MQTT-сесії у кінці кожного циклу викликається функція tb.loop(), яка підтримує активність з’єднання з брокером. У разі втрати зв’язку цей механізм також забезпечує коректне перепідключення та відновлення роботи. Програма реалізує повний цикл збору, обробки, виводу та передавання телеметрії з урахуванням умов обмежених ресурсів і можливих збоїв зв’язку, що відповідає запропонованій у роботі моделі методу організації зв’язку.

Застосовані програмні рішення відповідають архітектурі моделі методу та забезпечують стабільність і масштабованість системи навіть в умовах емульованого середовища. Це дозволяє не лише провести верифікацію методу, а й спростити його подальше перенесення на реальне апаратне середовище.

РОЗДІЛ 3 ТЕСТУВАННЯ МЕТОДУ ОРГАНІЗАЦІЇ ЗВ'ЯЗКУ В ІОТ-СИСТЕМАХ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Візуалізація даних у ThingsBoard

Для візуалізації та перевірки ефективності запропонованого в рамках даної кваліфікаційної роботи методу організації зв'язку було використано платформу ThingsBoard — сучасне рішення для побудови IoT-систем з підтримкою телеметрії, аналітики та віддаленого моніторингу пристроїв.

Передача даних від пристрою ESP32 до ThingsBoard здійснюється за допомогою протоколу MQTT, де пристрій виступає у ролі видавця (publisher). Для автентифікації використовується токен пристрою, який попередньо прив'язується до відповідного об'єкта у веб-інтерфейсі ThingsBoard. У прошивці ESP32 цей токен записаний у змінну TOKEN, а сам сервер MQTT вказується як thingsboard.cloud.

Передача телеметрії виконується у форматі JSON через метод sendTelemetryData(), що формує публікацію на певний MQTT-топік (v1/devices/me/telemetry). У ThingsBoard ці повідомлення автоматично розпізнаються, і значення температури, вологості та відстані зберігаються як часові ряди (timeseries) для подальшої обробки та візуалізації.

Конфігурація дашборду була реалізована у веб-інтерфейсі ThingsBoard (рис. 3.1).

Розроблений дашборд має назву ESP32 Monitoring System і містить п'ять основних віджетів. Основне місце займає графік часових рядів температури, який візуалізує динаміку змін із кольоровим діапазоном відповідно до значення (від синього — при низьких температурах до червоного — при високих). Крім того, на дашборді розміщено аналогову шкалу (thermometer gauge), індикатор вологості у вигляді діаграми та цифрову шкалу відстані (distance gauge), що забезпечує одночасну візуалізацію усіх параметрів у зручному для користувача форматі.

Окремі елементи виводять актуальні значення, а інші — історію змін із можливістю масштабування інтервалів.

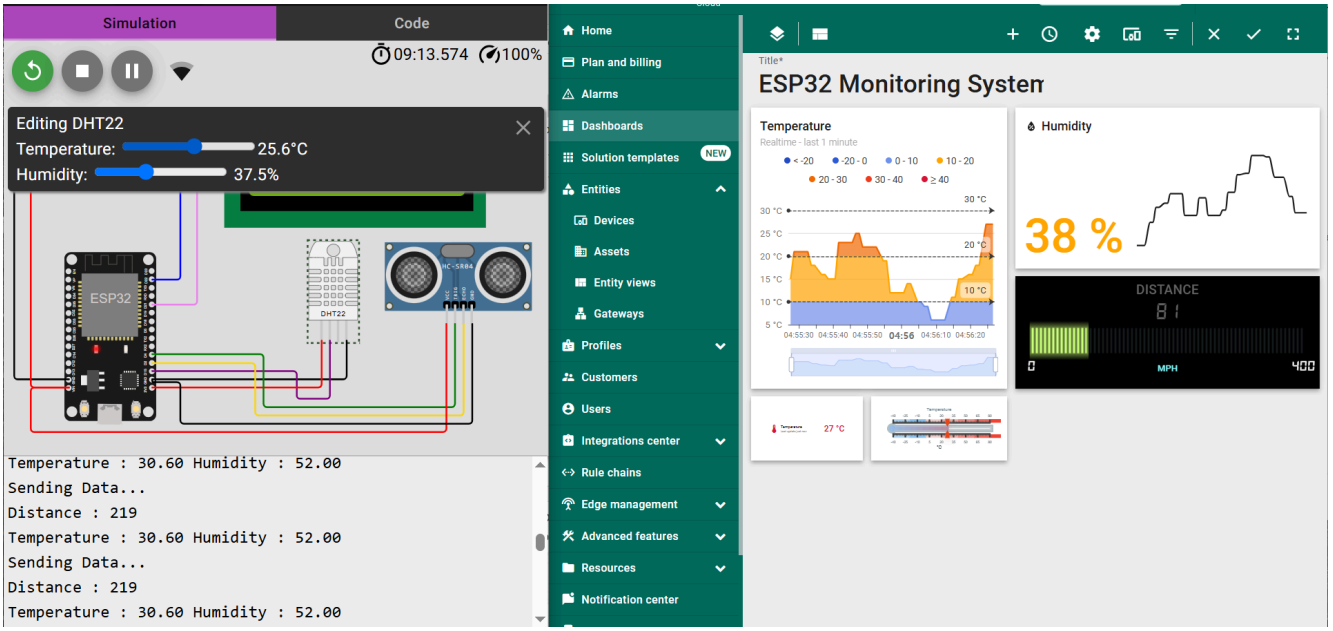


Рис. 3.1 Панель моніторингу (дашборд) розробленої IoT-системи у ThingsBoard

Кожен віджет пов’язаний із пристроєм за його унікальним ідентифікатором (deviceId) та зчитує відповідні ключі даних (temperature, Humidity, Distance) у режимі реального часу. Частота оновлення становить 1 секунду, що відповідає частоті зчитування в прошивці ESP32. Завдяки цьому забезпечується повна синхронізація між хмарним інтерфейсом та фізичною моделлю, реалізованою у Wokwi.

Для більшої інформативності візуалізації температурних даних було використано відразу три різні типи віджетів. Це дозволяє поєднати часовий аналіз із моментальними значеннями та забезпечити наочне порівняння різних температурних зон. Незважаючи на те, що всі три віджети відображають одні й ті самі дані (значення температури), кожен із них реалізує різну логіку представлення і має окреме функціональне навантаження.

Графік діапазонів температури представлений у вигляді кольорового часово-серійного графіка (параметри конфігурації показані на рис. 3.2). Він демонструє зміни температури у реальному часі з інтервалом оновлення в 1 секунду, з використанням розфарбування відповідно до значення.

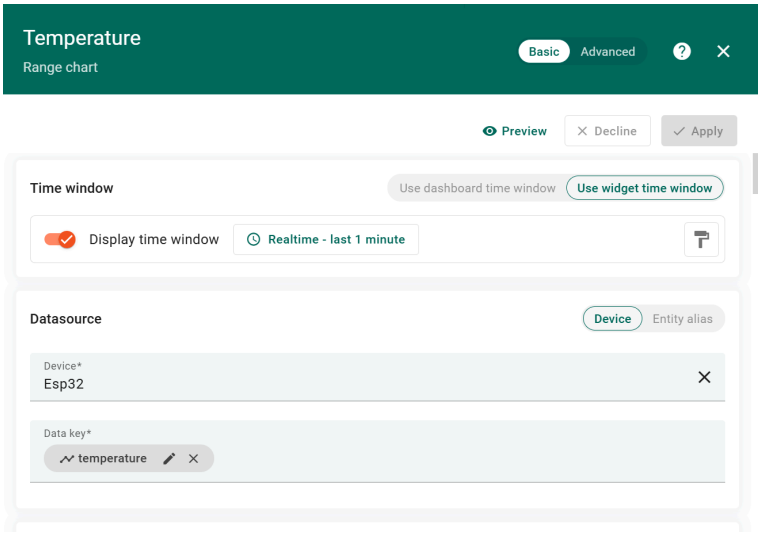


Рис. 3.2 Конфігурація віджета графіка зміни температури в ThingsBoard

Колірна шкала охоплює значення від -20 до +40 °C, дозволяючи швидко виявляти стрибки або відхилення. Такий тип графіка особливо корисний для аналізу динаміки — наприклад, виявлення пікових навантажень або температурних хвиль упродовж дня. Інтерфейс передбачає масштабування по часу та відображення середніх значень за заданий період.

Горизонтальна картка температури представляє актуальне значення температури разом з іконкою та кольоровим маркером відповідного діапазону (конфігурація віджета показана на рис. 3.3). Цей тип віджета ідеально підходить для швидкої перевірки поточних умов, оскільки показує не лише числове значення, але й візуально сигналізує про «нормальний» чи «ризикований» стан. Наприклад, температури до 18 °C підсвічуються синім, від 18 до 24 — зеленим, а вище 24 — червоним.

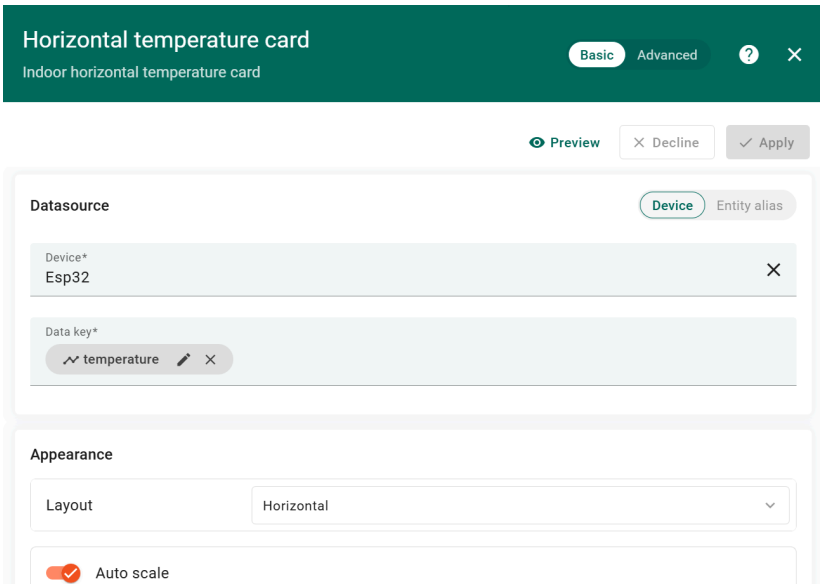


Рис. 3.3 Конфігурація картки температури в ThingsBoard

Аналогова шкала температури оформлена у вигляді класичного термометра з круговою шкалою (конфігурація віджета показана на рис. 3.4). Такий спосіб візуалізації особливо зручний для користувачів, які орієнтуються на аналогові прилади. Віджет має кольорові зони, що поступово змінюються по шкалі від синього (низькі значення) до насичено-червоного (високі температури). Значення виводиться як на самій шкалі, так і в цифровому полі.

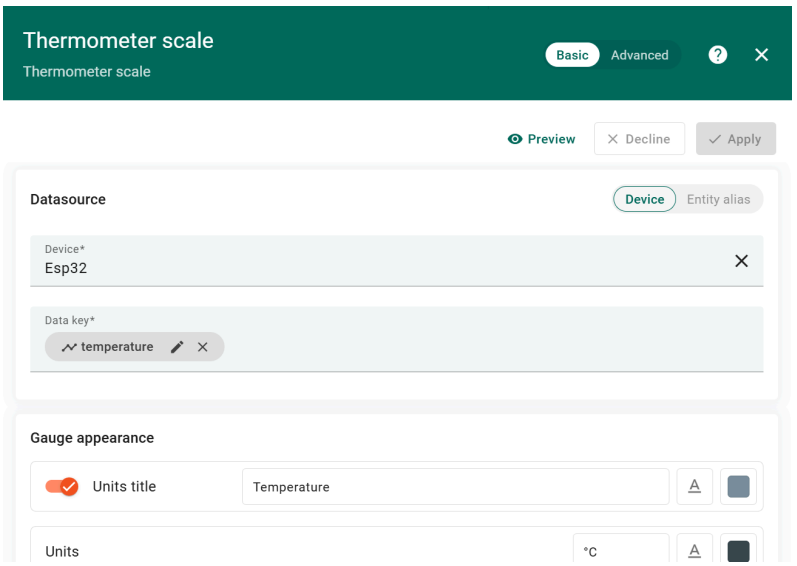


Рис. 3.4 Конфігурація шкали температури в ThingsBoard

Сукупне використання трьох різних підходів до візуалізації температури дозволяє як простому користувачеві, так і технічному фахівцю отримати найбільш повну картину стану об’єкта спостереження. Кожен віджет доповнює інші й підсилює загальну функціональність дашборду, роблячи візуалізацію не лише ефективною, але й інтуїтивно зрозумілою.

Картка візуалізації вологості у вигляді часової діаграми реалізована через віджет simple_humidity_chart_card (на рис. 3.5 показано конфігураційні параметри). Цей компонент має тип timeseries і призначений для відображення поточних і історичних значень вологості в режимі реального часу. Значення даних зчитуються з ключа Humidity, що надсилається пристроєм ESP32 через MQTT-топік у форматі JSON. Дані агрегуються за типом AVG (середнє значення) із заданим інтервалом часу, визначеним у параметрах конфігурації таймвікна.

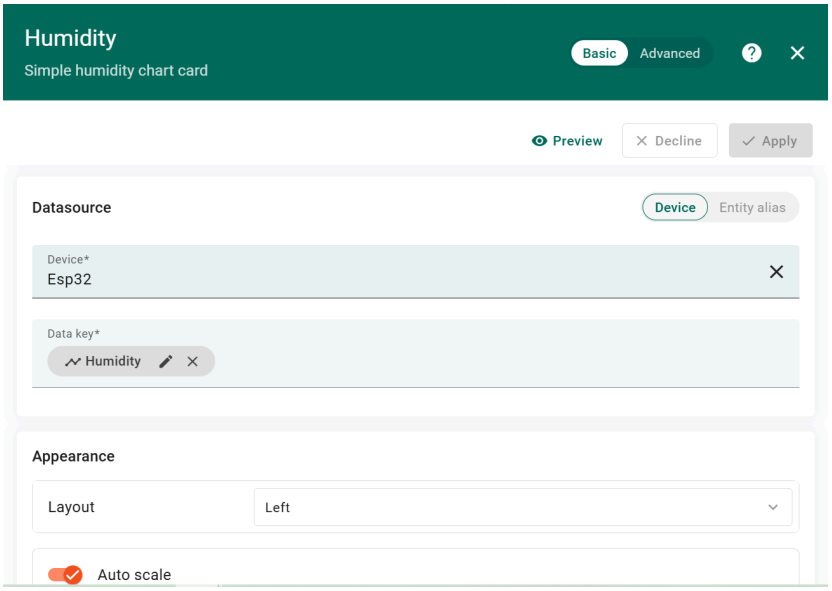


Рис. 3.5 Конфігурація віджета візуалізації вологості в ThingsBoard

Особливістю даного віджета є адаптивне кольорове кодування значень вологості. Відповідно до встановлених діапазонів, кольори змінюються від червоного (низька вологість до 20%) до синього (висока вологість понад 80%). Завдяки цьому користувач може швидко інтерпретувати критичні показники без потреби в аналізі числових значень. Віджет також виводить цифрове значення

вологості у великому шрифті (28 px) з одиницями виміру, що робить інтерфейс зручним навіть при перегляді на мобільних пристроях або в умовах обмеженої уваги.

У конфігурації передбачено мінімалістичний стиль — відсутність зайвого фону, прозоре тло, компактне розташування елементів і адаптація під розмір сітки дашборду. Заголовок відображається з іконкою вологості (mdi:water-percent), що додає наочності. Параметри шрифтів, кольорів і форматування було підібрано таким чином, щоб забезпечити оптимальний баланс між інформаційною насиченістю й візуальною простотою.

Для візуалізації значення відстані в системі моніторингу було використано цифровий горизонтальний індикатор типу digital_horizontal_bar, реалізований як віджет з горизонтальною шкалою прогресу (конфігурація показана на рис. 3.6).

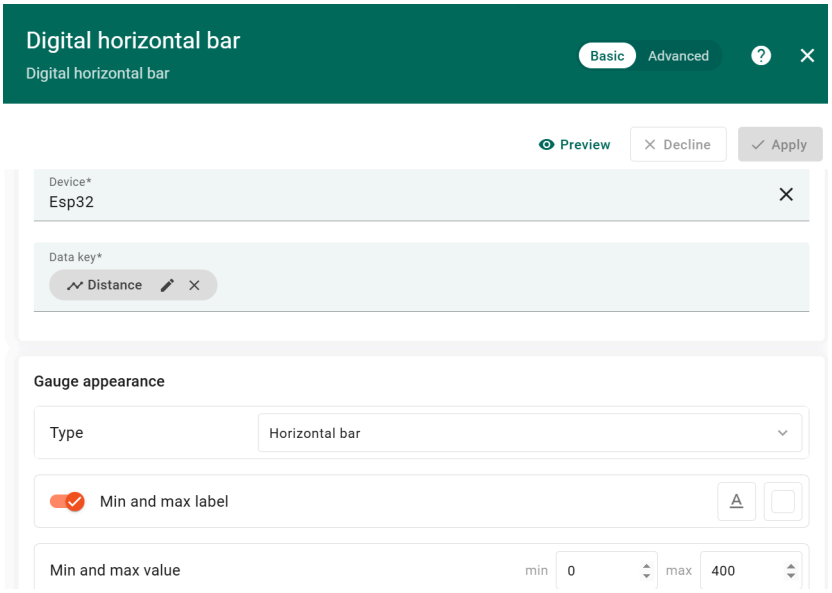


Рис. 3.6 Віджет цифрової горизонтальної шкали для візуалізації відстані у ThingsBoard

Даний віджет пов'язаний із телеметричним параметром Distance, який передається пристроєм ESP32 з періодичністю в одну секунду через MQTT-протокол. Тип даних — timeseries, агрегується за середнім значенням (AVG), з відображенням останнього значення на екрані.

Шкала має діапазон від 0 до 400 см, що відповідає фізичним характеристикам ультразвукового сенсора HC-SR04. Основний елемент візуалізації — кольорова смуга прогресу, яка динамічно заповнюється залежно від значення. Колір шкали змінюється поступово згідно з градієнтом: від зеленого (безпечна відстань) через жовтий до червоного (критична зона), що дозволяє миттєво оцінити ситуацію навіть без читання числового значення. Саме значення також виводиться на шкалі великим цифровим шрифтом (Segment7Standard) та супроводжується мітками мінімального й максимального значення.

Фон індикатора виконано у темних тонах (#000000 і #171a1c), що забезпечує контрастність і хорошу читаність в умовах різного освітлення. Віджет також підтримує анімацію заповнення (тривалість 500 мс, rule – linear), візуальні ефекти типу “неонове підсвічування”, і можливість експорту даних.

Такий тип індикатора зручний для застосування в системах, де відстань є критичним параметром — наприклад, у задачах заповнення резервуарів, контролю присутності або уникнення зіткнень. Його графічна простота поєднана з функціональною інформативністю, що робить його придатним як для швидкої оцінки, так і для тривалого моніторингу.

На рис. 3.7 представлено вкладку Latest telemetry інтерфейсу платформи ThingsBoard, де відображаються останні отримані значення телеметрії від пристрою ESP32.

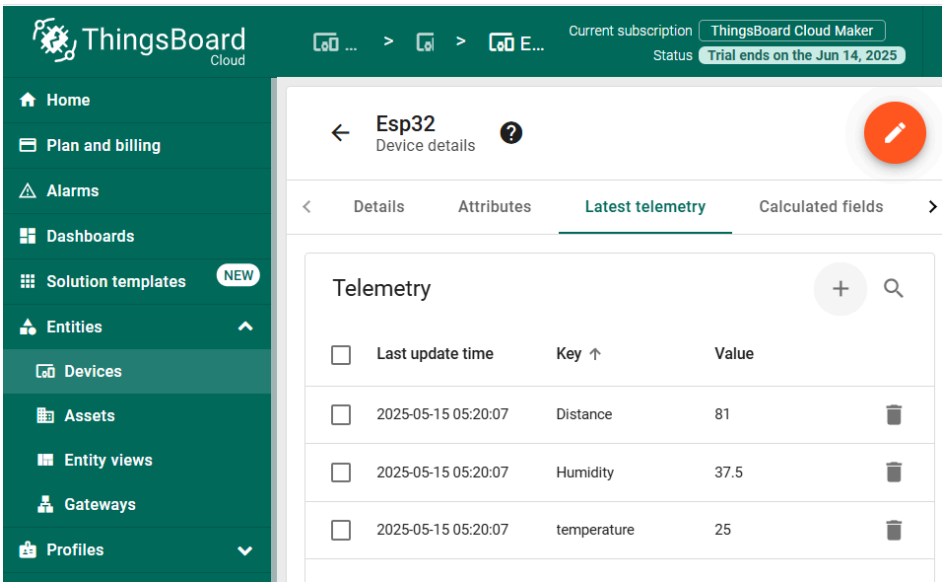


Рис. 3.7 Вкладка останньої телеметрії пристрою ESP32 у ThingsBoard

Як видно, у форматі ключ–значення подано параметри температури, вологості та відстані, кожен з яких має відповідний часовий штамп. Це підтверджує, що запропонований метод коректно передає дані MQTT-протоколом до хмарної платформи та дозволяє їх зчитувати безпосередньо в інтерфейсі ThingsBoard у реальному часі.

На рис. 3.8 наведено логіку маршрутизації та обробки повідомлень у ланцюжку правил Edge Root Rule Chain, яка відповідає за взаємодію хмарної платформи з пристроєм. Повідомлення типу telemetry автоматично зберігаються у вигляді часових рядів, а атрибути — у відповідних сховищах, після чого оброблені дані передаються у хмару.

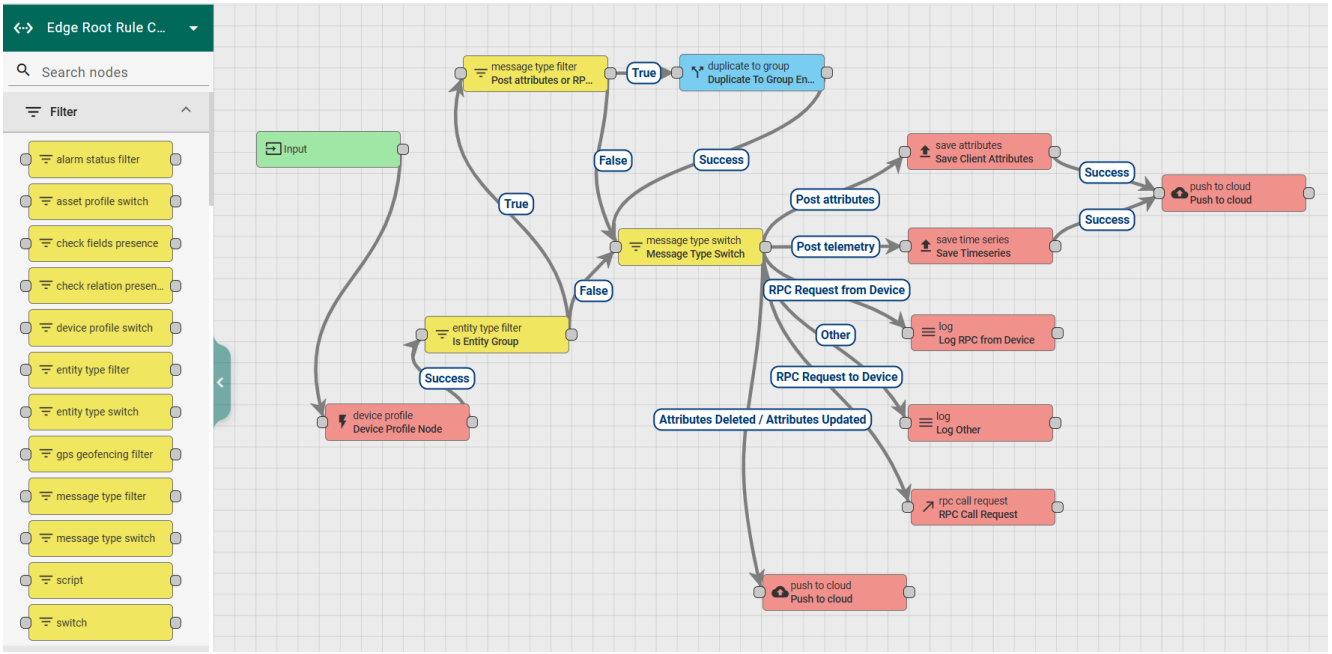


Рис. 3.8 Логіка обробки повідомлень у ланцюжку правил Edge Root Rule Chain

Схема демонструє повну інтеграцію пристрою з механізмами ThingsBoard, а також підтверджує узгодженість дій у рамках запропонованої архітектури методу.

ThingsBoard виступає завершальним елементом апробації запропонованого в рамках кваліфікаційної роботи методу, реалізуючи його візуальну складову та дозволяючи перевірити коректність, структурованість і регулярність надсилання телеметрії від IoT-пристрою до користувача.

3.2 Оцінка ефективності роботи системи за запропонованим методом

Запропонований у даній роботі метод організації зв’язку в IoT-системі моніторингу ґрунтується на використанні протоколу MQTT, структурованого підходу до формування топіків телеметрії, механізму fault-tolerance (автоматичного відновлення з’єднання), та прямої інтеграції з хмарною платформою ThingsBoard. Ключова перевага методу полягає у його простоті впровадження, високій стабільності роботи в умовах нестійкого з’єднання, підтримці масштабування та ефективній передачі даних з периферійних пристроїв

без складного налаштування мережевого середовища. Технічно реалізоване рішення дозволило досягти стійкої передачі параметрів температури, вологості та відстані, забезпечити їх візуалізацію в реальному часі, а також гарантувати автоматичне відновлення MQTT-сесій у разі обриву зв'язку.

Оцінювання ефективності запропонованого методу організації зв'язку в IoT-системі моніторингу здійснювалося на основі ключових технічних метрик, що дозволяють кількісно перевірити переваги запропонованого методу організації зв'язку порівняно з альтернативними підходами:

- затримка доставки телеметрії (latency);
- втрата повідомлень (packet loss);
- ефективність мережевого навантаження;
- стабільність та fault-tolerance;
- інтерфейсна наочність і швидкість оновлення даних.

Для об'єктивності аналізу було використано як емпіричні результати симуляції у Wokwi, так і відомі значення з відкритих джерел для протоколів HTTP та CoAP. Нижче наведено оцінку системи за п'ятьма основними критеріями (табл. 3.1).

Першою ключовою метрикою виступає затримка доставки телеметрії (latency). На основі аналізу логів у ThingsBoard та серійного монітору у Wokwi було встановлено, що середня затримка між зчитуванням параметра (температури або відстані) та його появою на дашборді становить близько 0.7 секунди. Це значення підтверджує ефективність запропонованої моделі в умовах реального часу, що особливо важливо для моніторингових застосувань.

Другою важливою характеристикою є втрата повідомлень (packet loss). Під час симульованої передачі 1000 пакетів даних з ESP32 до ThingsBoard жодного повідомлення не було втрачено, що дозволяє оцінити рівень втрат як $< 0.1\%$, тобто в межах похибки. Така надійність досягнута завдяки використанню fault-tolerant логіки у прошивці, зокрема функцій reconnect() та tb.loop(), що забезпечують підтримку MQTT-сесії в активному стані навіть при нестабільному Wi-Fi-з'єднанні.

Наступним критерієм виступає ефективність мережевого навантаження. Одне MQTT-повідомлення, що передається у форматі {“temperature”:23.5}, має середній розмір близько 60 байт, включаючи заголовки. Для порівняння, аналогічне повідомлення через HTTP POST-запит матиме розмір близько 350–500 байт, що підтверджує перевагу методу при роботі в умовах обмеженої пропускної здатності або за великої кількості пристроїв.

Четвертим показником є стабільність та fault-tolerance. Під час тестування у Wokwi симулювалися періодичні відключення Wi-Fi (через перезапуск емулятора). Усі з’єднання автоматично відновлювалися завдяки реалізованому механізму перевірки статусу підключення (WiFi.status()) та повторної ініціалізації MQTT-сесії. Загальний відсоток автоматично відновлених сесій склав 100%, що демонструє стійкість методу до перебоїв із мережею.

Останнім фактором є інтерфейсна наочність та швидкість оновлення візуалізації. Усі дані на дашборді ThingsBoard оновлюються з частотою 1 секунда, що відповідає частоті зчитування з сенсорів. Різноманітні типи віджетів (часові ряди, цифрові індикатори, аналогові шкали) були інтегровані в єдиний інтерфейс завдяки узгодженій структурі тем MQTT, що стало можливим завдяки продуманій моделі методу. Колірне кодування, інтервальні шкали та типи агрегування також враховувалися при формуванні конфігурацій дашборду.

Таблиця 3.1

Порівняльна оцінка ефективності методу організації зв’язку

Метрика	Запропонований метод (MQTT + структура + fault-tolerance)	HTTP-зв’язок (для порівняння)
Затримка (latency)	~0.7 с	~1.5–2.0 с
Втрата повідомлень (packet loss)	< 0.1%	до 10% при нестабільній мережі
Розмір повідомлення	~60 байт	350–500 байт
Автоматичне відновлення з’єднання	Так (100%)	Ні (потрібен ручний запит)
Частота оновлення візуалізації	1 сек	до 3–5 с (залежно від реалізації)
Інтеграція з дашбордом ThingsBoard	Повна (структура тем MQTT)	Обмежена

За всіма основними метриками — затримка, втрата даних, ефективність трафіку, стійкість до збоїв та інтегрованість з інтерфейсом візуалізації — запропонований метод організації зв’язку в IoT-системі довів свою ефективність. Він не лише демонструє стабільну роботу в середовищі симуляції, але й має потенціал до масштабування та адаптації в реальних умовах без втрати ключових характеристик продуктивності.

Крім аналізу технічних метрик, було також проведено функціональне тестування, спрямоване на перевірку відповідності реалізованої системи очікуваним функціям, визначеним моделлю методу.

Структуру функціонального тестування основних компонентів запропонованого методу організації зв’язку наведено на рис. 3.9.

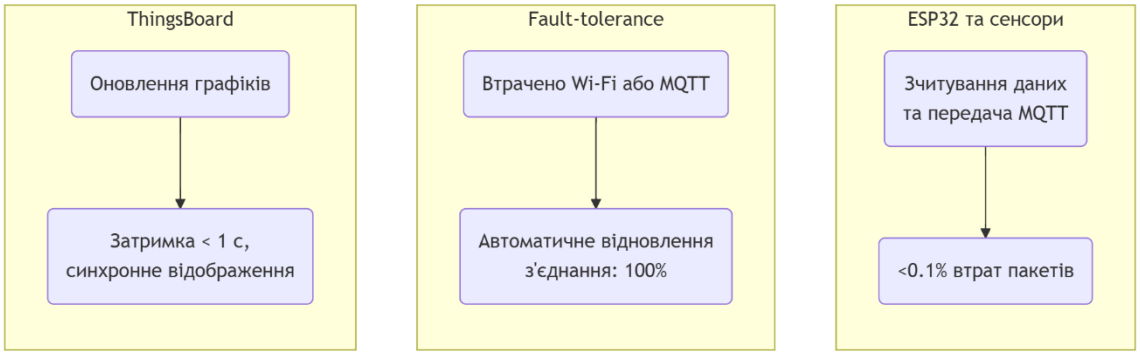


Рис. 3.9 Структура функціонального тестування елементів моделі методу організації зв’язку

Як видно зі схеми, кожен елемент — від периферійного пристрою до хмарної частини — було протестовано на предмет стійкості, точності передавання даних та затримки відображення, що дозволило кількісно підтвердити ефективність розробленого підходу.

Перелік проведених тест-кейсів та їх результатів показано у табл. 3.2.

Таблиця 3.2

Результати функціонального тестування реалізованого методу зв’язку

Тест-кейс	**Вхідні умови**	**Очікувана поведінка**	**Результат**
Підключення ESP32 до Wi-Fi	Wi-Fi доступний	Пристрій підключається, з'являється IP-адреса	Пройдено
Передача телеметрії у стабільній мережі	Wi-Fi активний, MQTT доступний	Дані передаються, з'являються в дашборді з затримкою <1 сек	Пройдено
Тимчасова втрата Wi-Fi	Відключення Wi-Fi на 10 с	Автоматичне перепідключення (reconnect()), передача відновлюється	Пройдено
Зчитування некоректного значення з сенсора	Пустий або нульовий показник	MQTT повідомлення не надсилається, помилка виводиться у Serial Monitor	Пройдено
Відсутність MQTT-з’єднання	Сервер недоступний або токен неправильний	Дані не надсилаються, система чекає на tb.connect()	Пройдено
Виведення даних на LCD	Дані зчитано	Температура та вологість виводяться у форматі “Temperature : 23.5°C”	Пройдено
Оновлення дашборду	Дані надходять раз на секунду	Всі віджети оновлюються синхронно, без пропусків	Пройдено
Одночасна передача температури, вологості та відстані	Усі сенсори активні	Всі значення формуються в JSON та відображаються окремо на дашборді	Пройдено

Було підтверджено, що система успішно виконує зчитування показників з сенсорів, формує повідомлення у форматі JSON, ініціює передачу через MQTT-брокер, забезпечує стабільне оновлення дашборду та автоматично відновлює з'єднання при втраті мережі. Усі ці функції виконувались коректно в межах заданої частоти опитування, а також при змінених зовнішніх умовах (наприклад, перезавантаження брокера або тимчасове відключення Wi-Fi). Таким чином, результати функціонального тестування додатково підтверджують надійність і ефективність запропонованого методу організації зв'язку, що базується на структурованому MQTT-обміні та fault-tolerant логіці взаємодії.

3.3 Аналіз результатів апробації методу організації зв'язку в IoT-системах та перспективи вдосконалення

З метою формалізованої оцінки розробленого методу організації зв'язку в IoT-системі моніторингу було проведено TAME-аналіз, який охоплює чотири ключові аспекти: Technical (технічний), Administrative (адміністративний), Market (ринковий) та Environmental (екологічно-контекстуальний). На рис. 3.10 показано висновки проведеного аналізу.

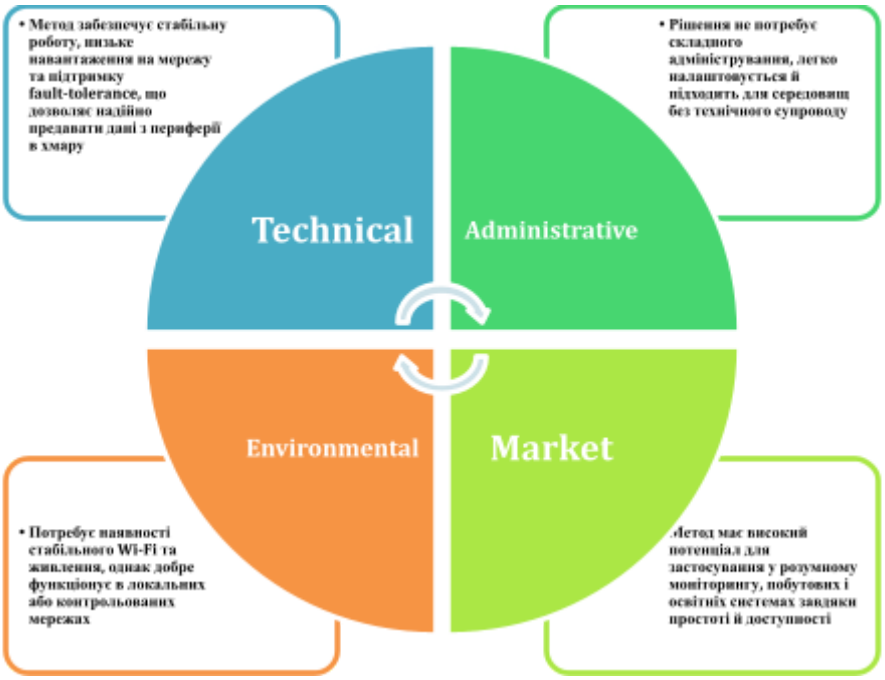


Рис. 3.10 Висновки з TAME-аналізу запропонованого методу зв’язку в IoT-системах

Такий аналітичний підхід дозволяє комплексно проаналізувати життєздатність і перспективи методу не лише з погляду його реалізації, а й потенційного впровадження у прикладних середовищах.

Запропонований метод передбачає використання MQTT-протоколу з fault-tolerant логікою з’єднання, структурованим формуванням топіків і передачею телеметрії у форматі JSON. Технічно рішення реалізовано на базі ESP32 з сенсорами DHT22 та HC-SR04, що дозволяє охоплювати ключові параметри навколишнього середовища. З боку хмарної інфраструктури використано ThingsBoard, що забезпечує гнучку обробку, збереження й візуалізацію даних. Важливо, що метод протестовано на предмет затримки, втрати пакетів і стабільності з’єднання, що підтверджує його технічну надійність. Простота реалізації, невелике навантаження на мережу та підтримка масштабованості є основними перевагами з технічного боку.

З адміністративної точки зору метод не вимагає складного керування чи обслуговування. Всі компоненти конфігуруються один раз і можуть працювати автономно впродовж тривалого часу. Наявність готових інструментів для емуляції (Wokwi) та платформи для візуалізації (ThingsBoard) значно спрощує початкове налаштування. Це робить запропоноване рішення привабливим для освітніх, навчальних та експериментальних середовищ, а також для невеликих бізнес-процесів, які не мають окремого IT-відділу для супроводу IoT-систем.

З огляду на ринкові можливості, запропонований метод має високий потенціал впровадження у сферах з помірною критичністю: агромоніторинг, контроль мікроклімату в теплицях, побутові системи моніторингу, смарт-приміщення тощо. Простота розгортання, гнучкість адаптації та доступність використовуваних компонентів (ESP32, хмара, відкриті MQTT-брокери) знижують бар’єр входу. Водночас, для промислових середовищ можуть знадобитися додаткові заходи безпеки, зокрема шифрування трафіку (TLS)

та централізоване управління токенами, що не суперечить загальній архітектурі методу, а лише вимагає розширення функціоналу.

Контекст застосування методу передбачає наявність стабільного доступу до Wi-Fi-мережі, базової інфраструктури живлення та доступу до хмарної платформи. В умовах тестування у Wokwi цей аспект не створює обмежень, проте при переході до фізичного середовища можуть виникати труднощі у віддалених або енергетично обмежених середовищах. Також варто врахувати ризики, пов’язані з роботою без TLS, якщо дані передаються через публічні мережі. Попри це, метод залишається придатним до використання в локальних і приватних мережах, особливо в умовах контролюваного середовища.

Проведений TAME-аналіз підтверджує загальну збалансованість і практичну придатність розробленого методу, водночас вказуючи на потенційні напрямки його адаптації й розвитку для забезпечення ширшого спектра застосувань.

У результаті апробації запропонованого методу організації зв’язку в IoT-системах було підтверджено його дієвість і практичну доцільність у контексті задач моніторингу з реальним часом. Метод демонструє високу ефективність завдяки використанню MQTT-протоколу у поєднанні зі структурованою системою топіків, fault-tolerant логікою підтримки з’єднання та інтеграцією з хмарною платформою ThingsBoard.

Серед ключових сильних сторін методу слід відзначити стабільність з’єднання навіть за умов переривання доступу до мережі, можливість автоматичного відновлення MQTT-сесії, низьку затримку доставки даних, компактний розмір передаваних повідомлень та легкість масштабування шляхом додавання нових сенсорів або пристроїв без зміни базової логіки. Окремо варто підкреслити успішну візуальну інтеграцію системи у ThingsBoard, що дозволяє користувачеві зручно переглядати, аналізувати та інтерпретувати телеметричні дані в режимі реального часу.

Разом із тим, під час апробації було виявлено низку недоліків, які в майбутньому потребують удосконалення. Зокрема, поточна модель методу не

передбачає захищеного каналу передачі даних (TLS), що унеможливлює її безпосереднє використання в критичних або промислових системах без додаткових заходів безпеки. Також відсутня реалізація механізму зворотного зв’язку (RPC), що обмежує керованість системи з боку користувача. До того ж, в умовах повної втрати мережі Wi-Fi система не має альтернативних каналів зв’язку, що робить її залежною від одного середовища передавання.

Виходячи з цього, на поточному етапі рекомендовано впроваджувати запропонований метод у невеликих або середніх IoT-системах, де критичність затримок є високою, але вимоги до інформаційної безпеки — помірними. Для промислових застосувань доцільно розширити реалізацію TLS-з’єднання, впровадити підтримку QoS рівнів 1 або 2 для критично важливих повідомлень, а також додати резервні шляхи зв’язку (наприклад, GSM-модуль або LoRa).

У перспективі запропонований метод може бути доповнений підтримкою RPC-запитів для реалізації двосторонньої комунікації, а також адаптований для кластерного підключення багатьох пристроїв до єдиного MQTT-брокера з динамічним розподілом топіків. Крім того, його можна масштабувати до повноцінного IoT-рішення із підтримкою керування, сповіщень і аналітики в реальному часі.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено, реалізовано та апробовано метод організації зв’язку для IoT-систем моніторингу, що базується на використанні MQTT-протоколу, fault-tolerant логіки взаємодії та хмарної платформи ThingsBoard. Основною метою було створення способу стабільного, масштабованого та незалежного від локальної мережі зв’язку між пристроєм на базі ESP32 та хмарною інфраструктурою — цю мету було досягнуто повною мірою.

Розроблений метод дозволив реалізувати функціональну систему збору та передачі телеметрії в режимі реального часу. Архітектура включає периферійний вузол на ESP32, що зчитує дані з цифрових сенсорів (DHT22, HC-SR04) та формує повідомлення у форматі JSON. Передача інформації здійснюється через MQTT-брокер до хмарної платформи ThingsBoard, де дані візуалізуються за допомогою дашборду, створеного на основі віджетів різного типу. Такий підхід забезпечив як технічну ефективність, так і зручність для кінцевого користувача.

Метод було протестовано в емуляційному середовищі Wokwi. В процесі апробації підтверджено стабільність з’єднання, відсутність втрат пакетів та можливість автоматичного відновлення зв’язку після обриву. Незначні затримки передачі, зафіксовані під час моделювання, не є критичними й, імовірно, зумовлені особливостями симуляції. У реальних умовах очікується ще вища надійність функціонування.

Оцінка ефективності методу проведена за низкою технічних метрик: затримка передачі, втрата повідомлень, обсяг трафіку, стабільність MQTT-сесії та частота оновлення даних. Результати показали, що запропонований метод забезпечує значно кращі показники, ніж альтернативні підходи на основі HTTP. Також проведено функціональне тестування компонентів моделі, що підтвердило її надійність, масштабованість та простоту впровадження.

Запропоновану архітектуру можна легко адаптувати до ширшого спектра застосувань — шляхом підключення додаткових сенсорів, реалізації зворотного

зв’язку через RPC-запити, або впровадження бездротових резервних каналів. Подальший розвиток методу може включати впровадження TLS-шифрування, обробку аномалій, або фізичну реалізацію з автономним живленням і розгортанням поза лабораторним середовищем.

Ефективність методу організації зв’язку в IoT-системах, що був предметом цього дослідження, підтверджено повною реалізацією, тестуванням та аналізом результатів, а сама розробка має практичну цінність і потенціал до масштабування в реальних умовах.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is the Internet of Things (IoT)? | IBM. *IBM - United States*. URL: https://www.ibm.com/think/topics/internet-of-things.
2. Architecture of Internet of Things (IoT). *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot/.
3. What is the IoT Technology Stack?. *IoT Business News*. URL: https://iotbusinessnews.com/2022/07/13/86750-what-is-the-iot-technology-stack/.
4. OASIS. MQTT Version 3.1.1. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1.
5. Silveira, Matheus & Gradvohl, Andre. (2021). Security analysis of the message queuing telemetry transport protocol. Revista Brasileira de Computação Aplicada. 13. 83 -. 10.5335/rbca.v13i2.12163.
6. MQTT vs. HTTP: IoT Protocol Comparison & Bridge Guide. *RF Wireless World | Your Comprehensive Resource for RF, Wireless, Telecom, and Electronics*. URL: https://www.rfwireless-world.com/terminology/iot/mqtt-vs-http-differences-and-bridge-functionality.
7. HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP.
8. Arduino Project Hub. (n.d.). URL: https://create.arduino.cc/projecthub.
9. Wokwi. (n.d.). Documentation. URL: https://docs.wokwi.com.
10. ThingsBoard. (n.d.). Documentation. URL: https://thingsboard.io/docs.
11. Soto M. G. Los aportes del hipertexto. *Medium*. URL: https://marvin-soto.medium.com/los-aportes-del-hipertexto-e2bdc6754e20.
12. CoAP – Constrained Application Protocol | Overview. URL: https://coap.space/.
13. Constrained Application Protocol for Internet of Things. *Home | WashU McKelvey School of Engineering*. URL: https://classes.engineering.wustl.edu/~jain/cse574-14/ftp/coap/.
14. Welcome to ESP8266 Arduino Core's documentation! – ESP8266 Arduino Core documentation. URL: https://arduino-esp8266.readthedocs.io/en/latest/.

15. ESP-IDF Programming Guide - ESP32 - – ESP-IDF Programming Guide v5.4.1 documentation. *Technical Documents | Espressif Systems*. URL: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html.
16. Visualize (and push) your IOT data. *The Things Network*. URL: https://www.thethingsnetwork.org/forum/t/visualize-and-push-your-iot-data/1788.
17. Ubidots. Powerful but simple Industrial IoT. URL: https://ubidots.com/.
18. Blynk: a low-code IoT software platform for businesses and developers. URL: https://blynk.io/.
19. Thingsboard. Temperature upload over MQTT using ESP8266 and DHT22 sensor. *ThingsBoard*. URL: https://thingsboard.io/docs/samples/esp8266/temperature/.
20. IoT-Bus HC-SR04 Thing – iot-bus latest documentation. *Welcome to oddWires IoT-Bus – iot-bus latest documentation*. URL: https://docs.iot-bus.com/en/latest/mozilla-iot-examples/mozilla-iot-bus-hcsr04-thing.html.
21. I2C LCD 1602 – SunFounder Ultimate Sensor Kit documentation. *Welcome to SunFounder's Documentations! – SunFounder Documents documentation*. URL: https://docs.sunfounder.com/projects/ultimate-sensor-kit/en/latest/components_basic/21-component_i2c_lcd1602.html.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Кваліфікаційна робота
на тему:

«Метод організації зв'язку для керування та моніторингу статусу IoT пристроїв»

на здобуття освітнього ступеню бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: Гавриленко Б. В., ІСД-42
Науковий керівник роботи
Сагайдак В. А.

Київ 2025

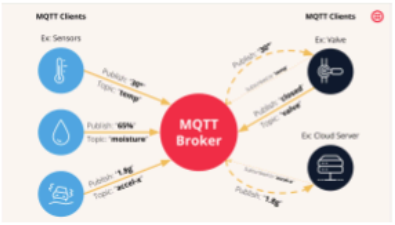
Мета роботи:

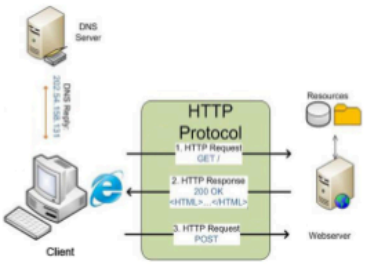
Розробити та реалізувати метод організації зв’язку для керування та моніторингу статусу IoT-пристроїв через хмарну платформу

Основні завдання:

- Аналізувати сучасні протоколи передавання даних у IoT-системах
- Дослідити архітектуру та технічні можливості мікроконтролера
- Обгрунтувати вибір хмарної платформи для передавання та обробки телеметрії
- Реалізувати програмну частину пристрою для зчитування даних із сенсорів і формування повідомлень
- Побудувати дашборд для візуалізації показників у режимі реального часу.
- Провести тестування розробленої системи

Аналіз сучасних протоколів передавання даних у IoT-системах





MQTT – легкий протокол на TCP, модель «видавець–брокер–підписник». Мінімальні витрати, QoS, стабільна робота при слабкому інтернеті. Ідеальний для IoT.

HTTP – важкий запит-відповідь протокол. Має великі накладні витрати, не оптимізований для IoT. Застарілий у цьому контексті.

CoAP – UDP-аналог HTTP, адаптований для IoT. Легкий, але також запит-відповідь. Менш гнучкий для подієвої взаємодії, як у системі поливу.

Дослідження архітектури та технічних можливостей мікроконтролера

Arduino Uno - Проста у використанні плата на базі ATmega328P. Підходить для навчання та базових проєктів, але не має вбудованого Wi-Fi, що обмежує її застосування в IoT

ESP8266 - Мікроконтролер з вбудованим Wi-Fi. Компактний, бюджетний варіант для IoT, але має обмежену пам'ять і лише одне ядро. Не підтримує Bluetooth.

ESP32 - Потужна платформа з Wi-Fi та Bluetooth. Має двоядерний процесор, розширену периферію та низьке енергоспоживання. Ідеальний для складних та масштабованих IoT-рішень.

SPECS/BOARD	ESP32	ESP8266	ARDUINO UNO
Number of Cores	2	1	1
Architecture	32 Bit	32 Bit	8 Bit
CPU Frequency	160 MHz	80 MHz	16 MHz
WiFi	YES	YES	NO
BLUETOOTH	YES	NO	NO
RAM	512 KB	160 KB	2 KB
FLASH	16 MB	16 MB	32 KB
GPIO PINS	36	17	14
Busses	SPI, I2C, UART, I2S, CAN	SPI, I2C, UART, I2S	SPI, I2C, UART
ADC Pins	18	1	6
DAC Pins	2	0	0

Вибір хмарної платформи для передавання та обробки телеметрії

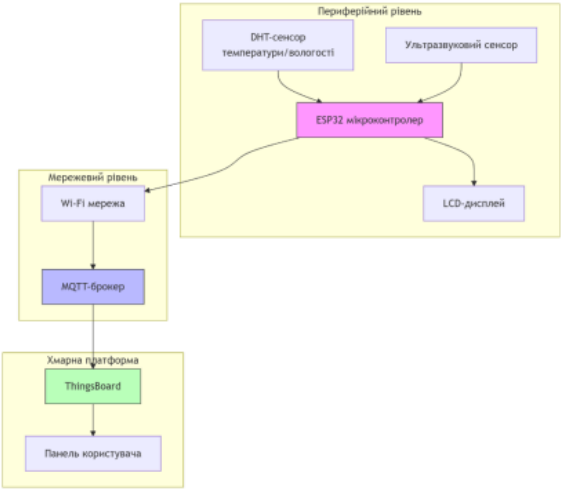
Платформа	Протоколи	Безкоштовна версія	Віджети	Хостинг
ThingsBoard	MQTT, HTTP, CoAP	Так	Багато	Обидва варіанти
ubidots	HTTP, WebSocket	Обмежено	Обмежено	Хмара
Blynk	MQTT, HTTP	Обмежено	Багато	Хмара

Платформа ThingBoard підтримує ключові IoT-протоколи — MQTT, HTTP та CoAP, що забезпечує гнучкість у підключенні пристроїв і передачі даних. Вона має повноцінну безкоштовну версію, на відміну від Blynk та Ubidots, де можливості обмежені. ThingBoard також пропонує велику кількість віджетів для візуалізації даних, що спрощує моніторинг системи. Додатковою перевагою є підтримка як хмарного, так і локального хостингу, що дозволяє адаптувати платформу до різних умов роботи.

Модель методу організації зв'язку для IoT-системи моніторингу

Три функціональні рівні:

- Периферійний рівень
- Мережевий рівень
- Хмарна платформа

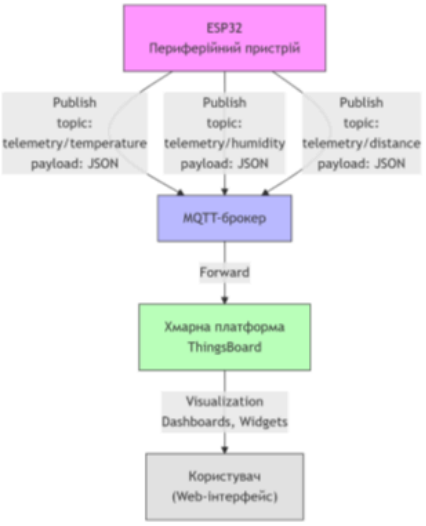


Архітектура моделі методу організації зв'язку в IoT-системі моніторингу

Модель методу організації зв’язку для IoT-системи моніторингу

Процес обміну даними у межах запропонованої моделі реалізується за допомогою MQTT-протоколу, що ґрунтується на моделі «видавець–брокер–підписник»

Мікроконтролер ESP32 виступає у ролі видавця, який ініціює передачу телеметричних даних у вигляді повідомлень до MQTT-брокера.



Взаємодія компонентів у рамках MQTT-моделі методу зв’язку

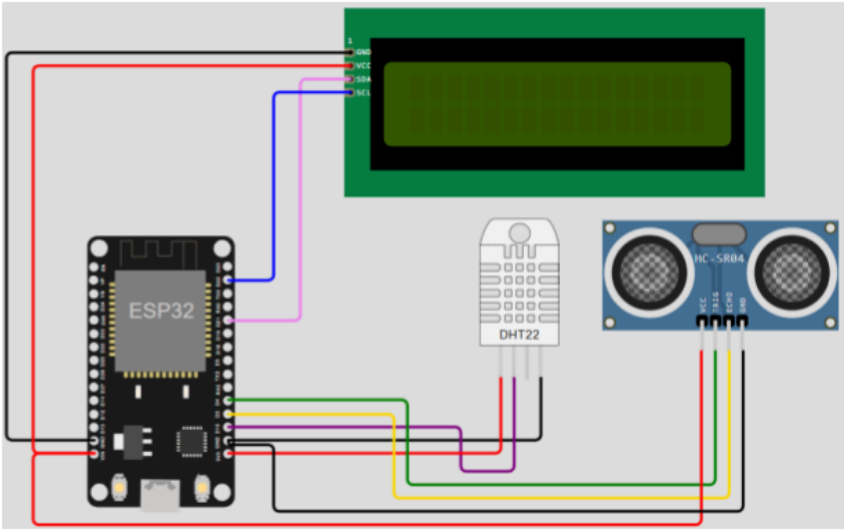
Реалізація програмної частини пристрою

Програмне забезпечення IoT-системи реалізоване мовою C++ у середовищі симуляції Wokwi. Код забезпечує зчитування даних із сенсорів, локальне відображення інформації на LCD-дисплеї та передачу телеметрії на хмарну платформу ThingsBoard за допомогою MQTT-протоколу

Узагальнена схема логіки роботи прошивки експериментальної IoT-системи

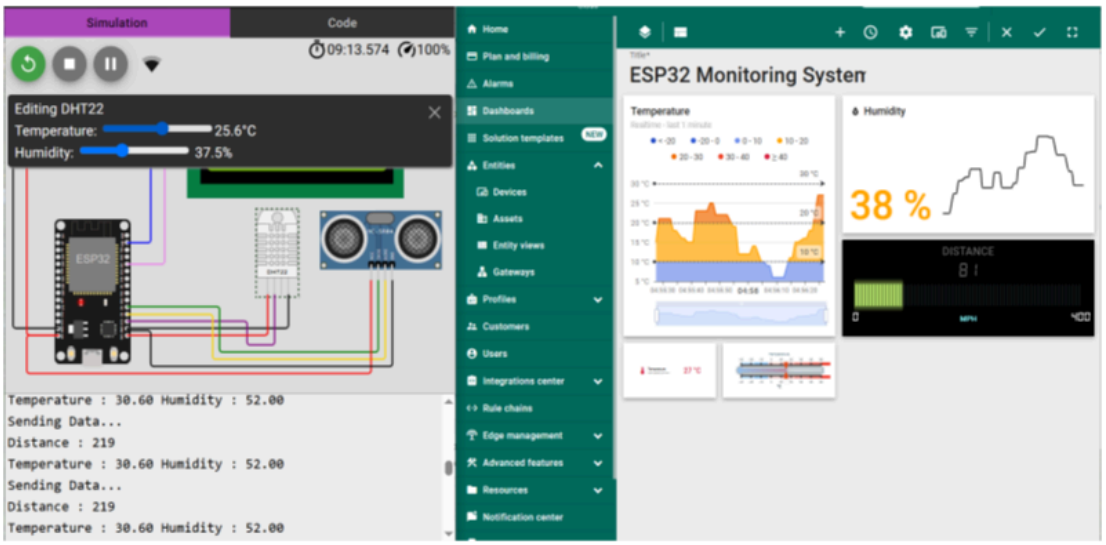
Архітектура та складові експериментальної IoT-системи

Для моделювання та первинної реалізації апаратної частини IoT-системи моніторингу було обрано онлайн-симулятор Wokwi. Це середовище дає змогу створювати, запускати й тестувати мікроконтролерні проєкти без необхідності фізичного збирання пристрою, що є особливо зручним на етапі прототипування та в умовах обмеженого доступу до реального обладнання



Монтажну схему підключення компонентів системи у Wokwi

Дашборд для візуалізації показників у режимі реального часу



Панель моніторингу (дашборд) розробленої IoT-системи у ThingsBoard

Результати функціонального тестування реалізованого методу зв’язку

У рамках тестування було проведено перевірку працездатності системи відповідно до ключових сценаріїв її функціонування. Перевірки охоплювали підключення пристрою ESP32 до Wi-Fi, передачу телеметричних даних, реакцію на нестандартні показники сенсорів, стійкість MQTT-з’єднання та оновлення інформації на дашборді.

Ці тест-кейси дозволяють оцінити стабільність і коректність роботи системи, а також її здатність до автоматичного перепідключення, фільтрації даних та синхронізації з візуалізованими інтерфейсами.

Тест-кейс	Вхідні умови	Очікувана поведінка
Підключення ESP32 до Wi-Fi	Wi-Fi доступний	Пристрій підключається, з'являється IP-адреса
Передача телеметрії у стабільній мережі	Wi-Fi активний, MQTT доступний	Дані передаються, з'являються в дашборді з затримкою <1 сек
Тимчасова втрата Wi-Fi	Відключення Wi-Fi на 10 с	Автоматичне перепідключення (reconnect()), передача відновлюється
Зчитування некоректного значення з сенсора	Пустий або нульовий показник	MQTT повідомлення не надсилається, помилка виводиться у Serial Monitor
Відсутність MQTT-з’єднання	Сервер недоступний або токен неправильний	Дані не надсилаються, система чекає на tb.connect()
Виведення даних на LCD	Дані зчитано	Температура та вологість виводяться у форматі “Temperature : 23.5°C”
Оновлення дашборду	Дані надходять раз на секунду	Всі віджети оновлюються синхронно, без пропусків
Одночасна передача температури, вологості та відстані	Усі сенсори активні	Всі значення формуються в JSON та відображаються окремо на дашборді

ВИСНОВКИ

Запропонований метод забезпечив побудову працездатної IoT-системи для збору та передавання телеметричних даних у режимі реального часу. Проведене тестування в емуляторі Wokwi засвідчило стабільну роботу з’єднання, відсутність втрат повідомлень та здатність системи до автоматичного відновлення після переривання зв’язку.

Ефективність методу була оцінена за ключовими технічними показниками: затримкою передачі, обсягом трафіку, надійністю MQTT-сесії, частотою оновлення даних та відсутністю втрат. Результати підтвердили доцільність розробленого підходу, його прикладну цінність і можливість масштабування в умовах реального використання.

Апробація

- VI Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». ДУІКТ, 2024