НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
Інформаційних технологій	Комп'ютерних систем, мереж та кібербезпеки
(назва факультету(ННІ))	(назва кафедри)
_____ Болбот І.М., д.т.н, проф.	_____ Касаткін Д.Ю., к. пед.н., доц.
(підпис) (ПІБ, вчене звання і ступінь)	(підпис) (ПІБ, вчене звання і ступінь)
«___»_____2025 р.	«___»_____2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження та вдосконалення серверної частини обробки даних контролю доступу в приміщення»

Спеціальність 123 «Комп’ютерна інженерія»
(код і найменування)

Освітня програма Комп’ютерні системи захисту інформації
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
д.т.н., професор _____ Мамченко С.М.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи
д.т.н., доцент _____ Шкарупило В.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Незельський В.В.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
к.пед.н., доц. _____ Касаткін Д.Ю.
(вчене звання і ступінь) (підпис) (ПІБ)
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Незельському Валерію Володимировичу
(прізвище, ім'я, по батькові)

Спеціальність 123 «Комп'ютерна інженерія»
(код і найменування)

Освітня програма Комп'ютерні системи захисту інформації
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи: «Дослідження та вдосконалення серверної частини обробки даних контролю доступу в приміщення»

затверджена наказом ректора НУБіП України від "29" жовтня 2024р. № 1941 «С»

Термін подання завершеної роботи на кафедру 14 листопада 2025 р.

Нормативно-довідкова та наукова література з інформаційних систем, баз даних, методів візуалізації та обробки даних; технічна документація з проєктування та розробки програмних комплексів; програмні засоби для розробки веб-застосунків і інтерактивних інтерфейсів; вимоги стандартів до побудови інформаційних систем і користувацьких інтерфейсів.

Перелік питань, що підлягають дослідженню:

1. Системний аналіз предметної області
2. Моделювання та архітектурне проєктування системи
3. Реалізація програмного забезпечення та технологічна інфраструктура системи
4. Тестування та оцінювання ефективності системи

Перелік графічного матеріалу (за потреби) презентація, постер, схеми та діаграми архітектури системи

Дата видачі завдання " 29 **"** жовтня 2024 р.

Керівник магістерської кваліфікаційної роботи _____ Шкарупило В.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Незельський В.В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5
ВСТУП 7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Опис предметної області 9
1.2 Огляд існуючих апаратних і програмних рішень 11
1.3 Аналіз вимог системи контролю 17
1.4 Постановка завдання 21
РОЗДІЛ 2ПРОЄКТУВАННЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ОБРОБКИ ПОДІЙ 23
2.1 Принципова схема пристрою для тестування серверної логіки 23
2.2 Електрична та монтажна схема дослідного стенду 25
2.3 Передумови створення програмного емулятора серверної логіки 32
2.4 Моделювання предметної області 34
2.5 Формалізація специфікації повідомлень і тем 36
3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ЛОГІКИ 39
3.1 Технологічні засоби та інструменти реалізації програмного емулятора серверної логіки 39
3.2 Апаратна архітектура взаємодії сенсорних вузлів, брокера подій та серверного емулятора 41
3.3 Представлення діаграми класів та кооперацій оброблення подій у програмному емуляторі 43
3.4 Представлення діаграми компонентів і пакетів системи 46
3.5 Висновки до третього розділу 49
4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ 51
4.1 План тестування програмних модулів та методика оцінювання результатів 51

4.2 Тестування інтелектуального апаратного емулятора ESP32 та підсистеми оброблення подій 52
4.3 Результати тестування інтелектуального апаратного емулятора 54
4.4 Забезпечення безпеки, надійності та цілісності даних під час роботи емулятора 56
4.5 Розгортання системи та склад інсталяційного пакета 57
4.6 Висновки до четвертого розділу 59
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. API — Application Programming Interface, програмний інтерфейс взаємодії компонентів.
2. CLI — Command Line Interface, інтерфейс керування системою через консольні команди.
3. CPU — Central Processing Unit, центральний процесор, який використовується для виконання обчислень.
4. CSV — Comma-Separated Values, текстовий формат табличних даних для експорту метричних логів.
5. DHCP — Dynamic Host Configuration Protocol, протокол автоматичного призначення IP-адрес.
6. ESP32 — модуль мікроконтролера з підтримкою Wi-Fi та Bluetooth, на якому реалізовано сенсорний вузол.
7. GPIO — General Purpose Input/Output, універсальні входи/виходи мікроконтролера.
8. HTTP/HTTPS — HyperText Transfer Protocol (Secure), протоколи передавання даних між компонентами системи.
9. JSON — JavaScript Object Notation, формат структурованих даних для логів, подій та конфігурацій.
10. MQTT — Message Queuing Telemetry Transport, легковаговий брокерний протокол передавання телеметрії.
11. MTLS — Mutual Transport Layer Security, двостороння TLS-автентифікація між брокером та емулятором.
12. NTP — Network Time Protocol, протокол синхронізації часу між компонентами.
13. P95 (latency) — 95-ий перцентиль затримки, метрика оцінювання продуктивності.

14. RAM — Random Access Memory, оперативна пам’ять, що використовується для обробки подій.

15. RTT — Round-Trip Time, повний час проходження мережевого запиту до вузла та назад.

16. SSL/TLS — Secure Sockets Layer / Transport Layer Security, криптографічні протоколи захисту трафіку.

17. TCP/IP — Transmission Control Protocol / Internet Protocol, стек мережевих протоколів.

18. UI — User Interface, графічний інтерфейс робочої станції оператора.

19. YAML — Yet Another Markup Language, формат зберігання правил та сценаріїв автоматизації.

20. EventLoop — цикл обробки подій у ядрі емулятора.

21. EventQueueManager — менеджер черг, що виконує пріоритезацію та маршрутизацію подій.

22. RuleEngine — модуль виконання правил, фільтрації і логічної кореляції сенсорних подій.

23. ActionDispatcher — компонент формування команд *control/+ /cmd* для керованих систем.

24. LoggingService — сервіс структурованого логування та збору метрик.

25. MetricsCollector — компонент вимірювання продуктивності (latency, queue depth, success rate).

26. ConfigManager — модуль завантаження параметрів, правил та конфігураційних файлів.

27. MQTTClientAdapter — адаптер взаємодії з брокером Mosquitto через paho-mqtt.

ВСТУП

Актуальність теми зумовлена потребою підвищення ефективності обробки даних у системах контролю параметрів внутрішнього середовища приміщень, які відіграють ключову роль у забезпеченні безпеки, енергоефективності та автоматизації житлових і виробничих об’єктів. Сучасні реалізації таких систем нерідко обмежені жорстко закладеною логікою обробки подій, що ускладнює адаптацію до динамічно змінюваних умов середовища, вимог користувача та сценаріїв реагування. Крім того, більшість промислових рішень не дозволяють дослідити архітектурні аспекти серверної частини без втручання в апаратну інфраструктуру, що стримує процеси прототипування та оптимізації. Розробка імітаційного середовища, яке емулює роботу серверного модуля обробки подій у контрольованих умовах, дозволяє формалізувати та експериментально перевірити механізми чергування запитів, пріоритезації реакцій, обробки виняткових ситуацій та ведення станів об’єктів.

Метою дослідження є створення та вдосконалення програмного інструменту для моделювання роботи серверної частини системи контролю за допомогою засобів Python, що дозволяє аналізувати часові, логічні та структурні характеристики обробки подій без прив’язки до фізичних сенсорів.

Для досягнення поставленої мети у роботі вирішуються такі **завдання:**

- аналіз сучасних серверних рішень у системах моніторингу середовища;
- формалізація вимог до функціональності, масштабованості та гнучкості обробки подій;
- побудова структурної та функціональної моделі серверної частини;
- розробка емулятора, який імітує динаміку подій від сенсорних модулів та реакцію серверного програмного забезпечення;

- проведення експериментального дослідження параметрів продуктивності, затримок обробки та поведінки системи в умовах пікових навантажень.

Об’єктом дослідження виступають програмні модулі обробки даних у системах моніторингу приміщень.

Предметом дослідження є методи та алгоритми реалізації серверної логіки, що забезпечує обробку та маршрутизацію подій у середовищі імітації.

Методологічну основу роботи складають підходи системного аналізу, імітаційного моделювання, проєктування програмної архітектури та експериментального тестування.

Засобами реалізації виступають інтерпретована мова Python, стандартні бібліотеки для побудови асинхронних і подієво-орієнтованих систем, а також інструменти логування, збору метрик та візуалізації структури черг.

Практична цінність роботи полягає у створенні універсального середовища тестування та вдосконалення алгоритмів серверної обробки подій, яке може бути адаптоване для задач автоматизації в системах контролю доступу, клімат-контролю, безпеки або енергозбереження.

Структура дипломної роботи включає вступ, чотири розділи, висновки, список використаних джерел та додатки. У першому розділі представлено аналіз предметної області та огляд актуальних архітектур обробки подій; у другому — запропоновано структурну модель, змодельована система та описано принципи побудови програмної частини; третій розділ містить деталі реалізації та опис логіки роботи імітаційного середовища; у четвертому — наведено результати тестування, аналіз ефективності, приклади сценаріїв роботи системи та рекомендації щодо практичного застосування. Висновки узагальнюють результати дослідження та окреслюють напрямки подальшого розвитку розробленого інструменту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область охоплює серверну обробку подій у системах контролю приміщень, де з різнорідних сенсорів надходять телеметричні повідомлення, що потребують нормалізації, маршрутизації, кореляції та генерації керувальних реакцій у режимі, близькому до реального часу; логічні обмеження визначаються референтними моделями Інтернету речей (погляди, шари, патерни взаємодії), які фіксує еталонна архітектура ISO/IEC 30141 і які задають інваріанти для порівняння реалізацій [1]. Комунікаційний рівень спирається на публічно доступні протоколи: MQTT як легковаговий транспорт із семантикою publish/subscribe та рівнями QoS для ієрархії «теми-повідомлення» [2], а також BACnet як об’єктно-орієнтований стандарт для інженерних систем будівель з уніфікованою моделлю пристроїв і сервісів, прийнятий як ISO 16484-5 [3]; вибір протоколу впливає на дизайн брокерів/шлюзів, стратегії конкурентного доступу до черг, полісів ретрансляції й відновлення з’єднань. З погляду обчислювальної організації, серверна частина поєднує реактивні конвеєри (event-driven/CEP) для критичних подій і потокову агрегацію метрик, тоді як неоперативні завдання виконуються пакетно; це відбивається на вимогах до ізоляції контурів, гарантій доставки, ідемпотентності хендлерів та спостережуваності (логування, трасування, метрики).

Структурну класифікацію аспектів предметної області - призначення, архітектура, комунікації, обробка даних, розгортання - наведено на рис. 1.1, яку надалі використовуємо як навігаційну карту для постановки вимог і вибору патернів.

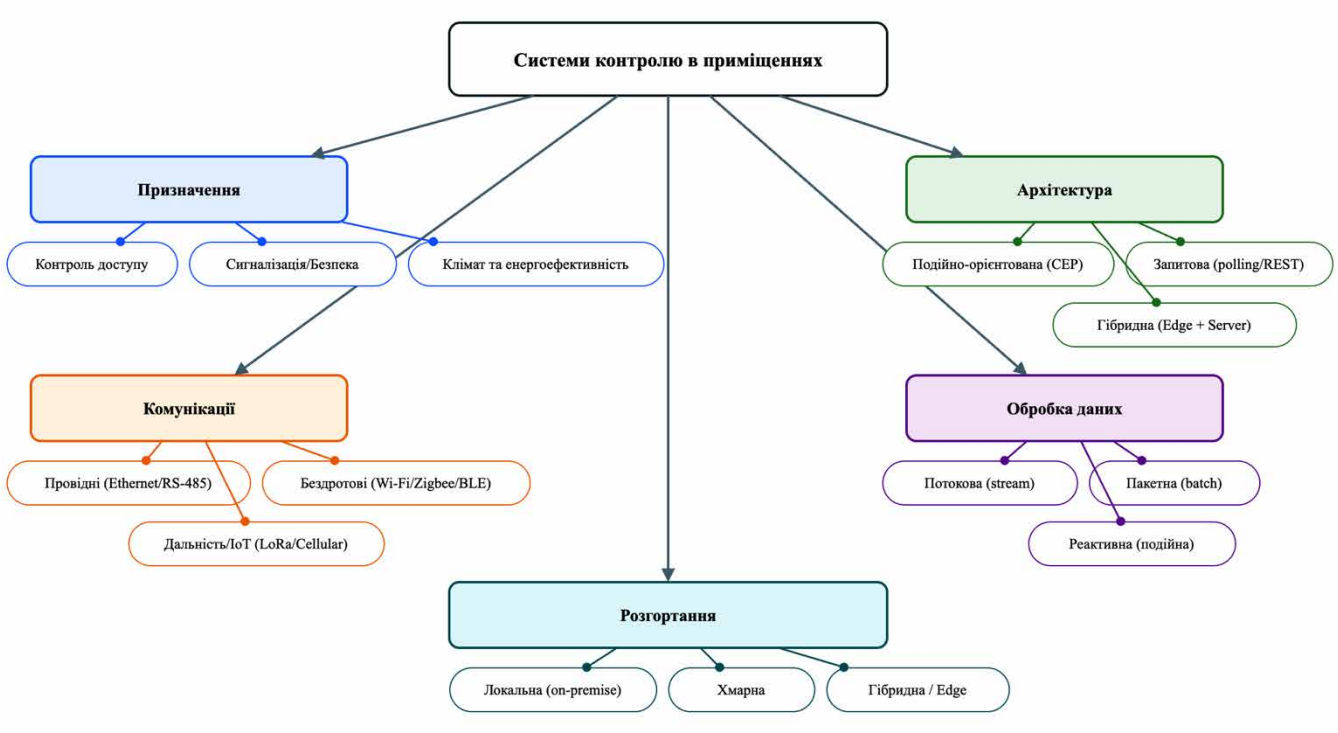

Рисунок 1.1 – Класифікація предметної області систем контролю в приміщеннях

Для орієнтувального порівняння протокольних підходів, що найчастіше застосовуються у серверних рішеннях контролю приміщень, узагальнені характеристики подано в табл. 1.1, де акцент зроблено на моделі обміну, гарантіях доставки та типовому застосуванні у виробничих сценаріях [2], [3].

Таблиця 1.1 - Порівняльні характеристики типових протоколів інтеграції для систем контролю приміщень

Протокол	Модель обміну	Гарантії доставки / QoS	Адресація	Типові застосування	Переваги	Обмеження
MQTT v5.0 [2]	Publish/Subscribe через брокер	QoS 0/1/2; ретейн/сесійні властивості	Теми (topic hierarchy)	Телеметрія сенсорів, подійні тригери	Легковагов ість, масштабов аність, відв’язка виробників	Потребує брокера; історія/запити — тільки через додаткові сервіси

BACnet [3]	Об’єктно-орієнтовані сервіси)	Залежно від сервісу;	Пристрої/об’єкти/властивості	HVAC, освітлення, BMS	Вендор-незалежність, спільна семантика	Складніший стек і адміністрування;

Продовження таблиці 1.1

HTTP/REST	Запит–відповідь (polling, webhooks)	Немає рівнів QoS; залежить від реалізації	URI-ресурси	Конфігурація, інтеграції, адмін-API	Простота, універсальність, інструментарій	Вищі накладні витрати; неідеально для телеметрії з високою частотою

Стислий профіль показує, що MQTT доцільний для телеметрії та реагування на події з брокером і політиками QoS, BACnet — для міжвендорної сумісності інженерних систем будівель та їх семантики на рівні об’єктів, тоді як HTTP/REST лишається базовим інструментом для конфігураційних і інтеграційних сервісів; у наступних підпунктах ці відмінності проєктуються на вимоги до серверної логіки, моделі черг і політики відмовостійкості [1]–[3].

1.2 Огляд існуючих апаратних і програмних рішень

Проектування сучасних систем контролю приміщень неможливе без аналізу існуючих технічних рішень, які поєднують апаратні модулі збору даних і програмні платформи для їх обробки, візуалізації та автоматизації дій. Основні вимоги до таких систем включають модульність, розширюваність, здатність до інтеграції з різними протоколами (MQTT, REST, Zigbee тощо), підтримку подійно-орієнтованої моделі та можливість локального або гібридного розгортання. Вивчення перевірених рішень дозволяє сформувати уніфіковане уявлення про стандартну архітектуру системи автоматизації та виявити можливості її еволюції.

Одним із найвідоміших прикладів реалізації програмної платформи автоматизації є Home Assistant - відкрита система, орієнтована на локальне керування розумним середовищем. Вона підтримує понад 3300 інтеграцій з пристроями і сервісами, працює поверх ОС Linux, має зручний інтерфейс для створення сценаріїв (automation flows) та конфігурується через YAML або вебінтерфейс. Система забезпечує подієво-орієнтовану обробку подій від сенсорів, підтримує візуалізацію в реальному часі та є прикладом повноцінного серверного ядра для моніторингу приміщення (рис. 1.2).

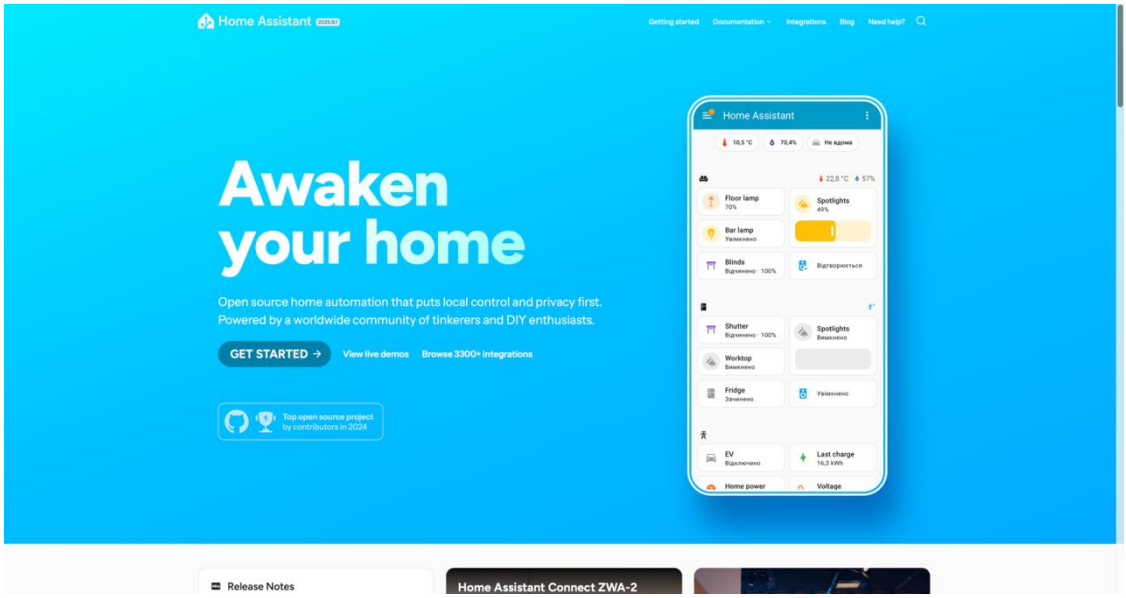

Рисунок 1.2 - Інтерфейс системи Home Assistant із локальним управлінням пристроями, виводом телеметрії та підтримкою подійного реагування (скріншот користувача з сайту [3]).

Альтернативним варіантом є openHAB — вендорно-незалежне рішення, побудоване на базі архітектури OSGi та Java Virtual Machine. Система підтримує правила у вигляді спеціалізованої DSL-мови, інтеграцію понад 400 протоколів через так звані bindings, та орієнтована на сценарії з високим рівнем кастомізації. Завдяки використанню архітектурного підходу з модулями (бандлами), openHAB дозволяє створювати складні логічні ланцюги взаємодії між подіями,

станами й автоматичними діями. Розгорнутий вигляд інтерфейсу платформи представлений на рис. 1.3.

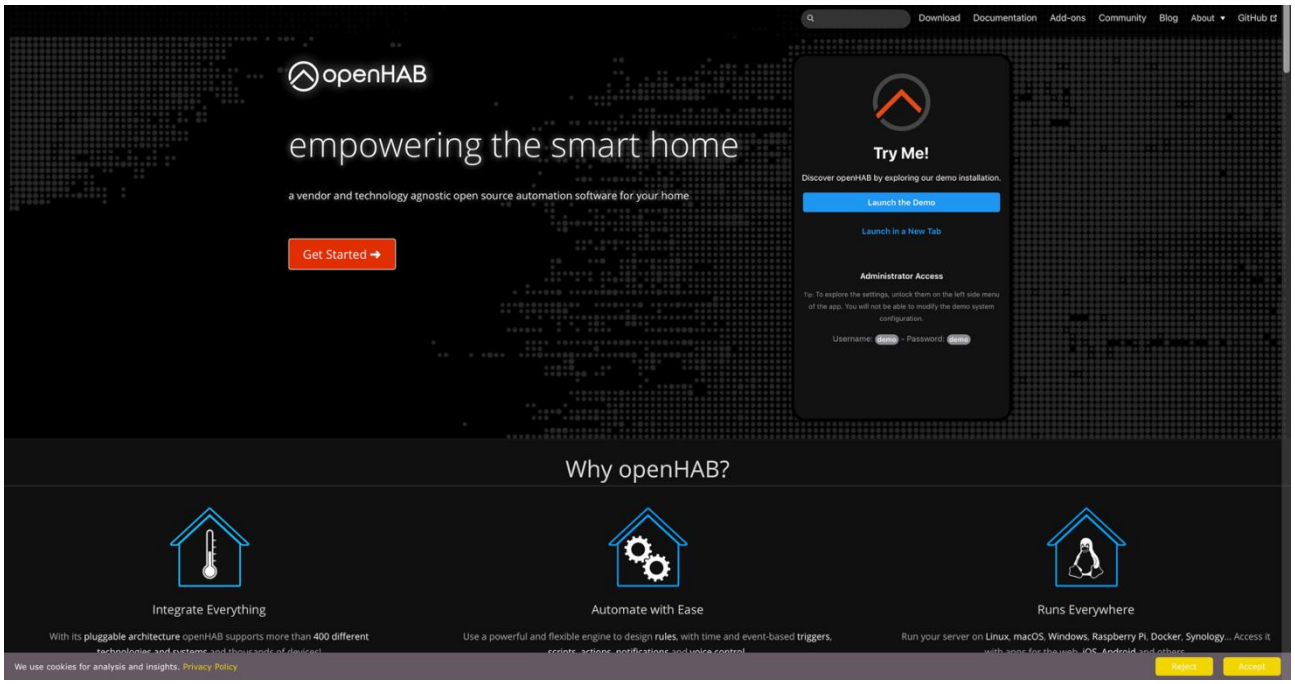

Рисунок 1.3 - Демонстраційна конфігурація платформи openHAB, що ілюструє інтерфейс взаємодії та системні модулі управління розумним середовищем (джерело: офіційна демонстраційна панель [4]).

У структурі більшості автоматизованих систем проміжну ланку між сенсорними вузлами та програмною платформою виконує MQTT-брокер, що реалізує шаблон обміну «publish/subscribe». Одним із провідних рішень у цій категорії є Eclipse Mosquitto — легковаговий брокер з відкритим кодом, який підтримує MQTT v3.1.1 та v5.0, QoS рівні 0–2, а також шифрування через TLS. Його архітектура забезпечує низьке енергоспоживання, стабільність роботи та сумісність із вбудованими системами (рис. 1.4). Використання такого брокера дозволяє розділити функції збору даних та їх інтерпретації, підвищуючи відмовостійкість та масштабованість системи.

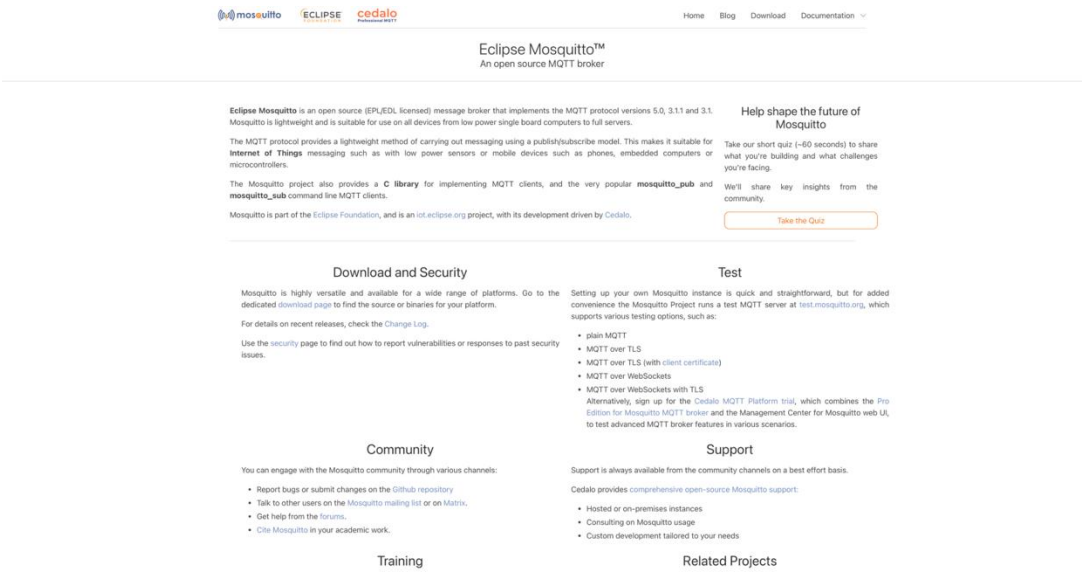

Рисунок 1.4 - Головна сторінка проєкту Eclipse Mosquitto з описом реалізованих можливостей, моделей використання та доступними варіантами конфігурації (джерело: офіційний сайт [5]).

Для реалізації edge-функціональності часто використовується Raspberry Pi 4 Model B, який дозволяє локально хостити як MQTT-брокер, так і саму платформу (наприклад, Home Assistant або кастомну серверну логіку). Завдяки чотирьохядерному ARM-процесору, до 8 ГБ RAM, Ethernet-інтерфейсу, USB-портам і GPIO Raspberry Pi може виступати як шлюз, так і обчислювальний вузол для обробки потоків подій (рис. 1.5).

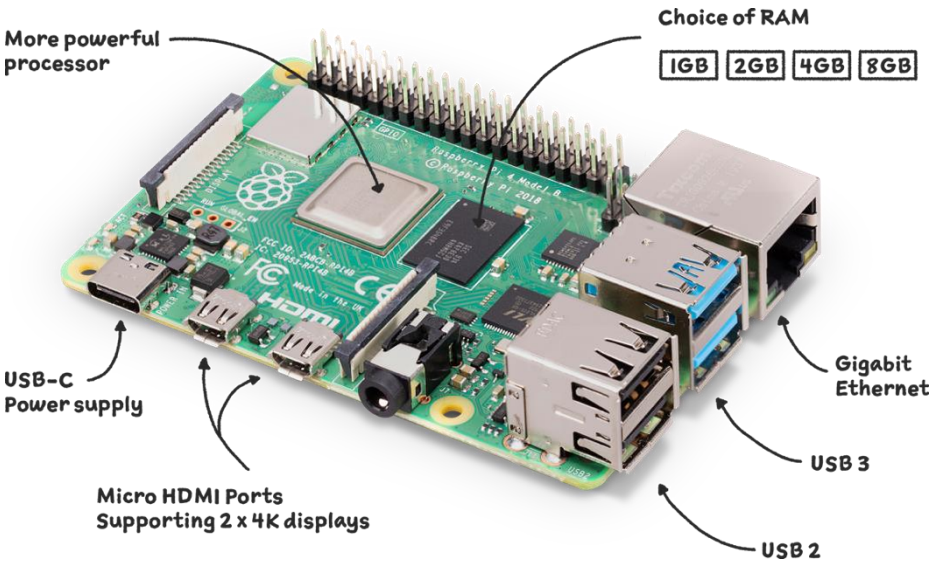

Рисунок 1.5 - Одноплатний комп’ютер Raspberry Pi 4 Model B із підписаними портами, які застосовуються в системах моніторингу та автоматизації (джерело: офіційна документація Raspberry Pi Foundation).

На рівні сенсорних вузлів ключовим компонентом є ESP32-WROOM-32, що поєднує в собі бездротові інтерфейси (Wi-Fi, Bluetooth), енергоефективний двоядерний процесор та великий набір периферійних модулів (GPIO, ADC, UART, I2C, SPI). Такі плати використовуються для підключення датчиків температури, руху, диму, освітленості тощо та надсилають події безпосередньо до MQTT-брокера (рис. 1.6).

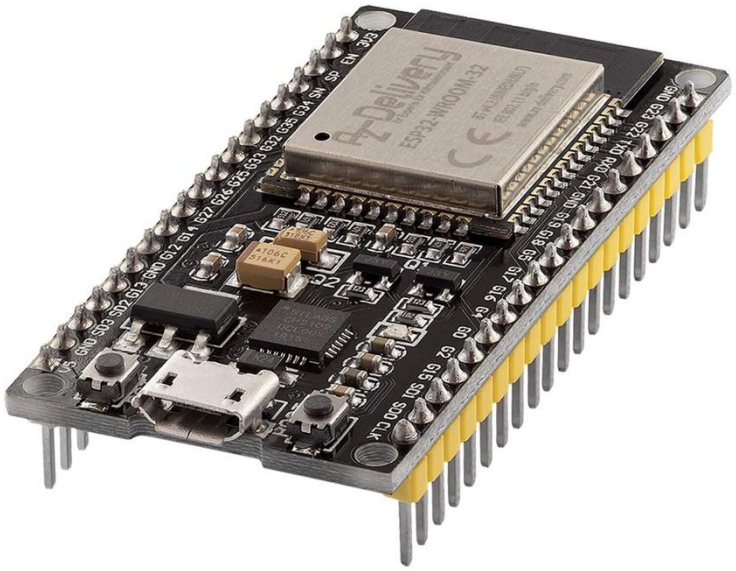

Рисунок 1.6 - Плата розробки ESP32-WROOM-32, яка використовується для побудови сенсорних пристроїв з можливістю бездротового зв’язку (джерело: технічна документація Espressif Systems).

Інженерна документація ESP32 містить схеми підключення і розведення пінів, що дозволяє розробляти модулі з підтримкою до 34 вхідних/вихідних ліній. Фрагмент схемотехнічної карти наведено на рис. 1.7.

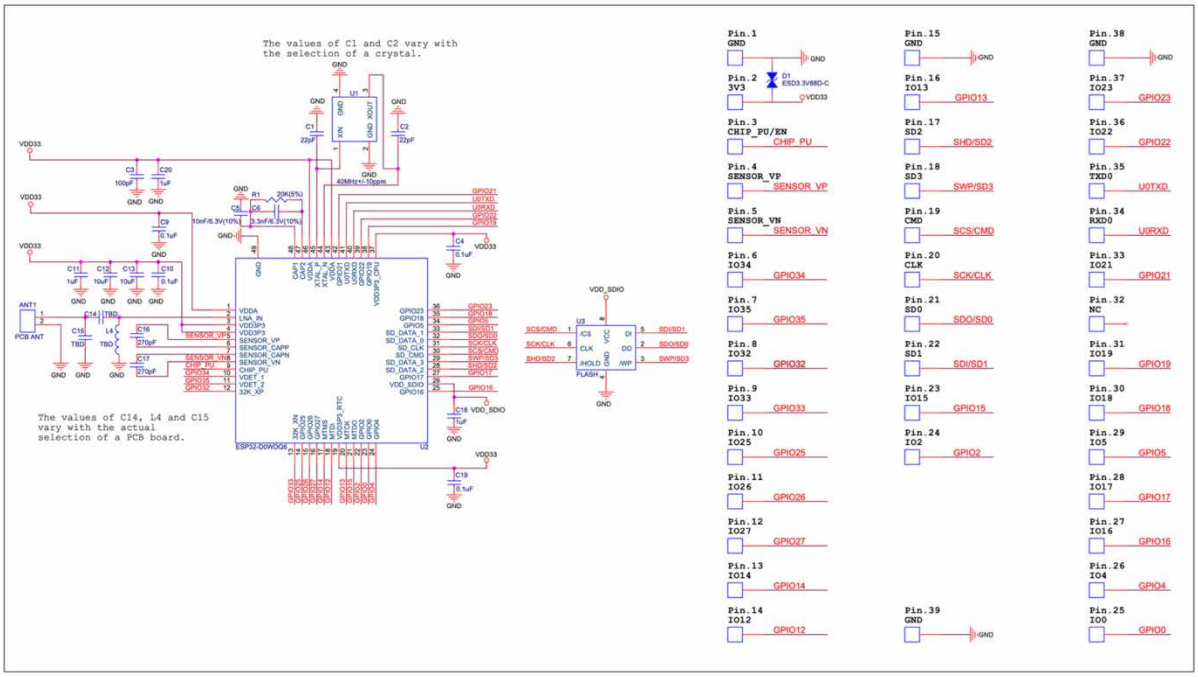

Рисунок 1.7 - Електрична схема плати ESP32 із пін-аутами, призначеними для інтеграції цифрових і аналогових сенсорів (джерело: даташит Espressif).

Для порівняльного аналізу ключові характеристики вищенаведених рішень разом із нашою реалізацією зведено у табл. 1.2, що дозволяє визначити місце нашої розробки у загальному технічному ландшафті.

Таблиця 1.2 - Порівняльна характеристика апаратно-програмних рішень і розроблюваної системи

Критерій	Home Assistant	openHAB	Mosquitto (MQTT)	Raspberry Pi 4	ESP32 DevKit	Наша система(емулятор на Python)
Призначення	Автоматизація	Автоматизація	Брокер повідомлень	Edge-обробка, шлюз	Сенсорний вузол	Серверна обробка подій
Архітектура	Monolith + UI	OSGi (модулі)	Демон + CLI/INI	Локальний Linux-сервер	MCU, low-level	Подієво-орієнтований симулятор
Мови/середовище	Python, YAML	Java, DSL	C, CLI	Linux, Bash, Python	C++, Arduino, MicroPython	Python 3.x, asyncio, logging

Обробка подій	Trigger → Action	Rule DSL → Action	Pub/Sub	Залежно від ПЗ	Event → MQTT Publish	MQTT Subscribe → логіка → реакція

Продовження таблиці 1.2

Масштабованість	Висока	Висока	Дуже висока	Помірна	Локальна	Гнучка, тестується під різні сценарії
Конфігурація	Web UI + YAML	Текстові правила	INI-файли	SSH, Ansible	Плата прошивається	JSON/CLI/скрипти
Вимоги до ресурсів	≥512 МБ RAM	≥1 ГБ RAM	<5 МБ RAM	1–8 ГБ RAM	520 КБ RAM	~50 МБ RAM + Python
Сумісність	Zigbee, Z-Wave, MQTT	KNX, MQTT, Modbus	MQTT 3.1.1/5.0	Будь-яке ПЗ	Потрібен брокер	Всі MQTT-сумісні вузли

Враховуючи сильні та слабкі сторони наявних рішень, наша система реалізується як симулятор серверної обробки подій, створений на базі мови Python із використанням бібліотек asyncio, paho-mqtt, json, logging, що забезпечують подієво-орієнтовану модель обробки повідомлень, масштабовану архітектуру та відлагоджене логування всіх транзакцій. Система не дублює функціонал Home Assistant чи openHAB, а зосереджена на дослідженні ефективності серверної логіки: затримки обробки, чергування подій, реакція на відмови брокера, логіка підтверджень, моделювання неідеальних вхідних даних. Вона не потребує фізичних сенсорів, а може бути поєднана з генератором подій або реальним MQTT-брокером. Такий підхід дозволяє здійснювати апробацію, профілювання та оптимізацію алгоритмів у лабораторному середовищі з подальшим перенесенням логіки на edge або хмарну інфраструктуру.

1.3 Аналіз вимог системи контролю

Формування вимог є критичним етапом, що визначає не лише функціональність програмного забезпечення, а й його стійкість до збоїв, ефективність при навантаженні, безпечність обробки даних та придатність до масштабування. Для систем, що взаємодіють з мережею сенсорних пристроїв через MQTT-протокол, базовим є забезпечення безперервної реактивної обробки повідомлень з мінімальною затримкою, підтримкою черг подій, і логікою автоматизованого реагування на задані шаблони. Функціональне ядро нашої системи включає механізми прийому повідомлень, їх валідації, фільтрації, запуску правил реагування, логування та формування агрегованої статистики. Повний перелік цільових функцій, організованих за підсистемами, узагальнено в таблиці 1.3.

Таблиця 1.3 – Функціональні вимоги системи

Підсистема	Вимога
Прийом подій	Асинхронний MQTT-сабскрайб на кілька тем одночасно
Розбір вхідного JSON	Перевірка цілісності структури, обробка різних схем повідомлень
Фільтрація	Ігнорування шумових подій, дублювань, відхилення невалідних записів
Правила реагування	Гнучка система тригерів: лог, репабліш, зберегти, сповістити
Обробка черг	Реалізація FIFO черги подій з пріоритетами або каналами
Логування	Структуроване логування з таймштампом, джерелом, типом події
Збір статистики	Формування агрегатів: кількість подій, помилок, тем, часу обробки

Як видно з табл. 1.3, система має підтримувати повний цикл обробки подій - від прийому MQTT-повідомлень до формування метрик або вторинних повідомлень. Ключовою є підтримка реактивного реагування через задані

сценарії (логування, публікація у відповідь, збереження до сховища), а також гарантоване обслуговування подій у паралельних каналах без втрати черговості або дублювання.

Поза межами функціональної поведінки, система має відповідати низці нефункціональних характеристик, що безпосередньо впливають на її експлуатаційні властивості. Наприклад, затримка реакції на подію не повинна перевищувати 100 мс за умови навантаження до 500 подій за хвилину, а в разі втрати зв'язку з брокером має забезпечуватись автоматичне відновлення сесії. Система повинна мати можливість масштабування шляхом запуску декількох обробників подій у межах одного процесу або через форкування. Крім того, обов'язковою є здатність оновлювати правила реагування без перезапуску основного циклу обробки. Ці характеристики систематизовано у таблиці 1.4, що фіксує основні нефункціональні параметри, на які орієнтується архітектура.

Таблиця 1.4 – Нефункціональні вимоги системи

Параметр	Обмеження / очікуване значення
Затримка обробки події	≤100 мс при ≤500 подіях/хв
Відновлення після збоїв	Reconnect з брокером протягом ≤2 секунд без втрати стану
Масштабованість	Підтримка багатопоточності / asyncio Queue з ізольованими воркерами
Гнучкість конфігурації	Підтримка гарячої заміни правил (hot reload)
Обслуговуваність	Прості лог-файли CSV або TXT, які не потребують сторонніх переглядачів
Сумісність	MQTT v3.1.1/v5.0, JSON UTF-8, Python ≥3.9

Безпека в even-driven системах має дві площини: захист від недовірених джерел і контроль за впливом подій на стан серверної частини. У нашому випадку реалізація передбачає whitelist тем для підписки, обмеження частоти повідомлень з одного клієнта, логування підозрілих payload окремо від

основного журналу, захист лог-файлів від перезапису та запобігання доступу з неавторизованих IP. За потреби може бути активовано TLS-з’єднання з MQTT-брокером. Підсумок безпекових вимог подано в табл. 1.5.

Таблиці 1.5 - Безпекові вимоги до взаємодії з брокером і захисту середовища

Категорія	Вимога
Доступ до брокера	Підписка лише на дозволені теми (whitelist)
Обмеження на частоту	Rate limiting з client_id або IP
Валідація вхідного JSON	Перевірка на контрольні поля, schema validation
TLS (опційно)	Шифрування каналів через mosquitto + certs
Захист логів	Режим "тільки читання", timestamp + hash логів
Окрема зона для помилок	Помилкові або небезпечні події логуються в ізольованому файлі

Останній набір вимог - технічні, що визначають операційне середовище, вимоги до інтерпретатора Python, залежностей та формату запуску. Емулятор повинен бути повністю автономним, запускатися одним скриптом, підтримувати CLI-параметри (наприклад, шлях до конфігурації правил), генерувати лог у stdout або файл, та працювати без підключення до БД або контейнеризації. Технічні умови формалізовано у таблиці 1.6, що визначає нижню межу вимог до середовища виконання.

Таблиця 1.6 - Технічні вимоги до реалізації серверної частини

Категорія	Вимога
Мова програмування	Python 3.9+
Бібліотеки	paho-mqtt, asyncio, json, logging, argparse
Формат запуску	Один файл .py з CLI-аргументами (--config, --logfile)
Залежності	Без Docker, без сторонніх брокерів (працює з віддаленим Mosquitto)
Операційне середовище	Linux/macOS/WSL, підтримка cron/task scheduler

Тестування	Вбудовані тести (pytest/unittest), тестові payload-файли
Стандартні схеми JSON	Вхідний об'єкт містить: topic, timestamp, payload (dict)

Система визначається як легковаговий серверний модуль, орієнтований на асинхронну обробку MQTT-подій із високим ступенем контрольованості, прозорим логуванням, можливістю динамічної зміни сценаріїв реагування та ізоляцією критичних потоків. Наявність чітко структурованих вимог дозволяє створити таку архітектуру, що буде не лише емулювати поведінку реального сервера, але й виступати основою для подальшого розгортання на edge-пристроях або в хмарному середовищі.

1.4 Постановка завдання

Метою є створення та експериментальна верифікація серверного модуля обробки подій для систем контролю приміщень у вигляді емулятора на Python, який реалізує повний реактивний конвеєр — прийом повідомлень із MQTT-топіків, синтаксичну й семантичну валідацію JSON-навантаження, фільтрацію, кореляцію, маршрутизацію та генерацію керувальних реакцій - із гарантованими межами продуктивності, відмовостійкості, безпеки та спостережуваності; завдання охоплює формалізацію інтерфейсів взаємодії з брокером (subscribe/publish, рівні QoS 0–2, опційне TLS), визначення простору топіків і схем повідомлень, проєктування асинхронних черг обробки подій та ідемпотентних хендлерів з гарантією збереження відносного порядку подій у межах топіка, а також впровадження структурованого логування зі збором метрик (частота подій, частка помилок, затримка обробки за медіаною та 95-м процентилем, кількість перепідключень); експерименти передбачають відтворювані профілі навантаження з генераторів подій без фізичних сенсорів (стаціонарні потоки, сплески типу ON/OFF, корумповані payload із

контрольованою часткою аномалій), ін’єкцію відмов мережевого та протокольного рівнів (розрив з’єднання, штучні затримки брокера, повторні спроби доставки), а також A/B-порівняння базової лінійної реалізації та запропонованого подієво-орієнтованого конвеєра; приймальні критерії фіксують цільові пороги: 95-й процентиль затримки не перевищує 100 мс за навантаження до 500 подій на хвилину, автоматичне відновлення MQTT-сесії відбувається не довше двох секунд без втрати стану, правила реагування оновлюються «на льоту» менш ніж за секунду, використання ресурсів не виходить за межі одного процесорного ядра та ста мегабайт оперативної пам’яті, логування залишається повним і цілісним; у частині безпеки завдання передбачає білий список дозволених тем, обмеження частоти для клієнтів, окремий журнал підозрілих подій, захищений режим доступу до логів і, за потреби, шифрування каналу брокером, що відповідає рекомендованим практикам протоколу MQTT та вимогам до якості ПЗ з позицій моделі ISO/IEC 25010; очікуваними результатами є специфікація API повідомлень і правил, реалізація емулятора (Python 3.9+, asyncio, paho-mqtt) разом із генератором навантаження та відмов, набір сценаріїв і тестів відтворюваності з фіксованими початковими значеннями, стенд збору метрик і звіт з аналітикою впливу рівнів QoS, політик черг та фільтрації на затримку, втрати і стабільність, а також практичні рекомендації щодо перенесення серверної логіки на edge- або хмарну інфраструктуру з урахуванням експлуатаційних вимог і стандартів [6], [7].

РОЗДІЛ 2ПРОЄКТУВАННЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ОБРОБКИ ПОДІЙ

2.1 Принципова схема пристрою для тестування серверної логіки

Фізичне середовище для моделювання та тестування серверної логіки побудоване на основі модульної IoT-архітектури, яка забезпечує генерацію вхідних подій, передачу повідомлень, їх обробку та зворотній зв'язок у вигляді керувальних дій. Центральним елементом обрано мікроконтролер ESP32, що поєднує в собі широкі комунікаційні можливості (Wi-Fi, Bluetooth), апаратну підтримку цифрових і аналогових інтерфейсів (GPIO, UART, I2C, SPI, ADC/DAC), низьке енергоспоживання та сумісність із типовими сенсорними модулями. Як брокер публікацій/підписок використано Mosquitto, що реалізує MQTT-протокол, дозволяє налаштовувати рівні доставки повідомлень QoS (0 - at most once, 1 - at least once, 2 - exactly once), підтримує TLS-шифрування, фільтрацію топіків та гнучку авторизацію. Для симуляції реакції на події реалізовано серверну логіку на Python, яка підписується на теми sensor/# та alarm/#, використовує бібліотеки asyncio, paho-mqtt, logging, та формує відповідні керувальні сигнали або виконує логування результатів. На рис. 2.1 наведено принципову схему інформаційної взаємодії між усіма елементами системи - від сенсорних вузлів до модулів серверної логіки.

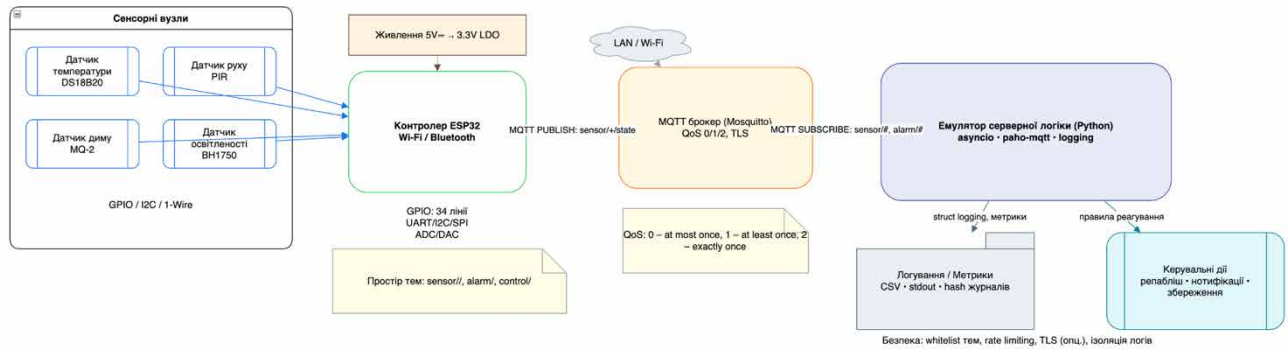

Рисунок 2.1– Принципова схема пристрою для моделювання взаємодії між сенсорами, MQTT-брокером та серверною логікою

Сенсорна частина системи містить чотири основні модулі: датчик температури (DS18B20, 1-Wire), датчик руху (PIR, цифровий вхід), датчик диму (MQ-2, аналоговий вихід) та датчик освітленості (BH1750, I2C). Всі вони підключені до ESP32, який формує MQTT-повідомлення згідно з топіками виду sensor/<тип>/<id>/state і відправляє їх на брокер. Серверна логіка, реалізована у вигляді асинхронного обробника, здійснює підписку на відповідні теми, виконує перевірку payload, застосовує правила, які формалізують поведінку системи в умовах різних подій, та ініціює керувальні дії, які публікуються у відповідні зворотні теми. Додатково передбачено логування у форматах CSV та stdout, а також збереження хешованих журналів, що забезпечує аудит змін. Кожен етап містить логіку обробки винятків: від втрати зв’язку з брокером до некоректного JSON-формату або повторної публікації. Сценарії логування й реакції у випадку винятків є частиною тестових випадків, що використовуються для перевірки надійності реалізованої серверної логіки. Узагальнену класифікацію компонентів системи, їх призначення та взаємозв’язки наведено в таблиці 2.1.

Таблиця 2.1 – Структура та функції компонентів системи тестування

Компонент	Призначення	Інтерфейси / Протоколи
Датчик температури DS18B20	Збір даних про температуру	1-Wire
Датчик диму MQ-2	Виявлення диму, формування аналогового сигналу	ADC / GPIO
Датчик руху PIR	Виявлення переміщення об'єктів у зоні спостереження	GPIO (digital)
Датчик освітленості BH1750	Вимірювання освітленості	I2C
Контролер ESP32	Агрегація сенсорних даних, публікація MQTT	GPIO, UART, I2C, Wi-Fi, MQTT
MQTT брокер Mosquitto	Посередник між публікатором і підписником	MQTT, TLS, QoS
Емулятор серверної логіки	Підписка на теми, обробка повідомлень, виконання дій, логування	Python, paho-mqtt, asyncio, logging
Керувальні дії	Репаблікація команд, нотифікації, запис у журнал	MQTT, stdout, CSV, JSON

Запропонована архітектура дозволяє ефективно відтворювати сценарії роботи реальної системи моніторингу подій, включаючи всі основні аспекти: передачу, валідацію, реакцію, керування й логування, що критично важливо для перевірки коректності алгоритмів серверної логіки у контролюваному середовищі.

2.2 Електрична та монтажна схема дослідного стенду

Реалізація електричної схеми системи контролю доступу базується на використанні апаратної платформи ESP32 DevKit, що забезпечує необхідний рівень обчислювальних ресурсів, підтримку бездротової передачі даних (Wi-Fi, Bluetooth), а також велику кількість цифрових і аналогових GPIO-ліній для підключення периферійних модулів. Контролер є ядром усієї системи й відповідає за обробку сигналів від зчитувачів, прийняття рішень, керування виконавчими пристроями та логування подій (рис. 2.2).

Рисунок 2.2– ESP32 DevKit V1 як центральний вузол керування

Для ідентифікації користувачів інтегровано RFID-зчитувач RC522, який працює за інтерфейсом SPI та підключається до ESP32 через порти SCK, MOSI, MISO, SS. Робоча напруга зчитувача складає 3.3V, тому для його стабільної роботи було використано лінію живлення з LDO-перетворювача. Комунікація з модулем здійснюється на рівні апаратного SPI-протоколу, що дозволяє досягти низьких затримок та високої достовірності переданих UID-кодів (рис. 2.3).

Рисунок 2.3 – RC522 RFID-зчитувач з антеною котушкою

З метою підвищення надійності ідентифікації також застосовується біометричний сканер відбитків пальців (наприклад, R305), що забезпечує автономне розпізнавання відбитків із внутрішньою пам’яттю та верифікацією користувача. Підключення до ESP32 виконується через UART-інтерфейс із живленням 5V. Сигнали управління додатково фільтруються за допомогою програмного debounce-механізму (рис. 2.4).

Рисунок 2.4 – Модуль біометричного сканера відбитків пальців

Для реалізації механізму доступу використано електромеханічний замок на 12V типу NC (normally closed). Живлення замка здійснюється від окремого блоку живлення, а керування — через транзисторний ключ на основі N-MOSFET або NPN-транзистора, який вмикається сигналом із GPIO контролера. Для захисту схеми від індуктивного пробою застосовано діод Шотткі, підключений паралельно котушці (рис. 2.5).

Рисунок 2.5 – Електромагнітний замок для системи контролю доступу

Контроль стану дверей реалізовано через магнітний геркон, який замикається або розмикається залежно від положення дверей. Сигнал із геркона подається на цифровий вхід ESP32, що дозволяє вести точний трекінг відкриттів і закриттів, а також формувати відповідні логічні сценарії при спробі несанкціонованого доступу (рис. 2.6).

Рисунок 2.6 – Контактні геркони для детекції відкриття дверей

Для збереження точного часу подій та забезпечення незалежності від зовнішніх джерел синхронізації, в систему інтегровано модуль реального часу DS3231. Комунікація із ESP32 здійснюється по шині I2C (SCL/SDA). DS3231 має внутрішній температурно-компенсований генератор, що дозволяє зберігати точність до ±2 ppm (рис. 2.7).

Рисунок 2.7– RTC модуль DS3231 з батарейним резервом

Живлення усієї системи забезпечується від стабілізованого адаптера на 12V із відповідним перехідником для макетної плати або роз’ємів на корпусі. Для стабілізації напруг до рівнів 5V і 3.3V використано понижуючі стабілізатори (DC-DC та LDO відповідно). Основні гілки живлення мають фільтруючі конденсатори та спільну шину заземлення для уникнення пульсацій і паразитних потенціалів (рис. 2.8).

Рисунок 2.8 – Стабілізований блок живлення 12V з перехідником DC barrel

З метою повного охоплення функціональних і комунікаційних потреб системи контролю доступу з трекінгом відвідуваності було сформовано цілісний

перелік апаратних компонентів, кожен з яких відповідає за конкретну підсистему - ідентифікації, обробки, виконавчих дій, виводу або інтерфейсної взаємодії. В таблиці 2.2 наведено узагальнені характеристики, маркування та функціональне призначення кожного вузла, що застосовується у складі дослідного стенду.

Таблиця 2.2 – Перелік основних апаратних компонентів системи

№	Назва компонента	Приклад маркування	Призначення
1	Мікроконтролер	ESP32 DevKit або Arduino Mega + Wi-Fi	Центральний вузол управління, логіка прийняття рішень, передача даних
2	RFID-зчитувач	RC522 (13.56 MHz), Wiegand26/34	Зчитування безконтактних ID-карт і токенів (MIFARE, UID)
3	Біометричний модуль (опціонально)	R305 або ZFM60	Розпізнавання відбитків пальців, альтернатива RFID
4	Електромагнітний замок	YS-138 (12V), NO/NC	Замикання/розмикання дверей за сигналом контролера
5	Датчик відкриття дверей	Магнітний геркон або оптичний датчик	Виявлення факту відкриття або несанкціонованого доступу
6	RTC модуль (годинник реального часу)	DS3231 або DS1307	Збереження точного часу подій незалежно від живлення
7	ЖК дисплей (опціонально)	I2C LCD 1602 або OLED SSD1306	Виведення ID користувача, повідомлень про доступ, помилки
8	Клавіатура	4×4 Keypad Matrix	Ввід PIN-коду або паролю локально без карти
9	Біпер / зумер / індикатор	Активний 5V зумер + світлодіоди	Візуальна й аудіоіндикація підтвердження або відмови доступу
10	Пам’ять EEPROM (опціонально)	AT24C32 або внутрішня Flash	Збереження UID, історії входів/виходів, логів у разі відсутності сервера
11	Wi-Fi або Ethernet модуль	Вбудований в ESP32 або W5500 (Ethernet)	Надсилання логів до сервера, синхронізація з БД
12	Блок живлення стабілізований	12V/2A адаптер + step-down 3.3V/5V	Забезпечення живлення замка, ESP32 та периферії
13	Макетна плата або друкована плата	Breadboard або кастомна PCB	Тестове або постійне з'єднання компонентів у корпусі

Продовження таблиці 2.2

14	SD-карта (опціонально)	SD-модуль або microSD SPI	Резервне зберігання логів та подій при втраті мережі

Представлений набір компонентів є базовим для побудови гнучкої та масштабованої системи доступу, яка здатна працювати як автономно (офлайн), так і в інтеграції з хмарними або локальними серверами. Модулі можуть змінюватися в залежності від конкретних потреб - наприклад, заміна RC522 на Wiegand-зчитувачі для більшої дальності або додавання Ethernet-модуля замість Wi-Fi у разі використання в корпоративній мережі з фаєрволами. Завдяки наявності RTC-модуля та EEPROM/SD-карти забезпечується цілісність історичних даних навіть у разі повної втрати електроживлення або мережевого підключення.

Монтажна реалізація системи базується на інтеграції всіх перерахованих компонентів у єдину електричну схему, що охоплює джерело живлення, контролер, периферійні модулі введення/виведення, зчитувачі та виконавчі пристрої. Як зображено на рис. 2.9, живлення усієї системи забезпечується стабілізованим джерелом 12V DC, з якого через каскад step-down-конвертерів формується два живильних рівні: 5V — для модулів, що потребують живлення із підвищеним струмовим навантаженням, та 3.3V — для чутливих мікросхем, зокрема ESP32, RFID RC522, RTC DS3231. Логічна земля (GND) поєднує всі вузли в спільну шину, що гарантує відсутність паразитних потенціалів між модулями.

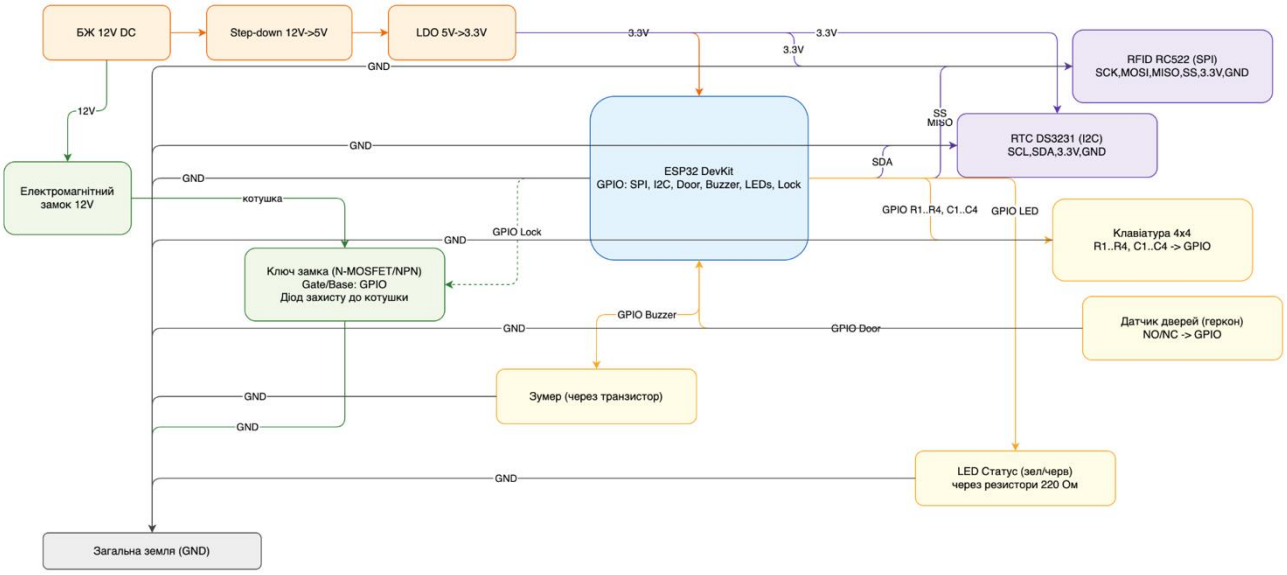

Рисунок 2.9– Повна електрична схема взаємодії модулів контролю доступу

Контролер ESP32 DevKit підключено до всіх ключових вузлів системи. SPI-інтерфейс використовується для з'єднання з RFID-зчитувачем RC522, який отримує живлення 3.3V та використовує лінії SCK, MOSI, MISO, SS, що підключені до відповідних GPIO. Через I2C-лінії (SDA, SCL) здійснюється передача даних до годинника реального часу DS3231. Клавіатура типу 4×4 матриця підключена безпосередньо до восьми цифрових GPIO, а зчитування відбувається за допомогою сканування рядків і стовпців. Для індикації використано двоканальне LED-рішення - зелений і червоний світлодіоди з резисторами 220 Ом, підключені до окремих виходів ESP32.

Управління електромагнітним замком здійснюється за допомогою транзисторного ключа (N-MOSFET або NPN), що керується окремою лінією GPIO. Включення ключа дозволяє подачу живлення на котушку замка. Для захисту від зворотної ЕРС встановлено діод Шотткі паралельно замкові. Зумер, що також керується через транзисторний каскад, забезпечує звукову індикацію подій — як підтвердження авторизації, так і сигналізацію відмови. Датчик стану дверей (геркон) підключений до GPIO з внутрішнім підтягувальним резистором, що дозволяє точно фіксувати момент фізичного відкриття або закриття.

Схема відображає фізичну топологію монтажу, яка забезпечує надійність, масштабованість і безпеку - з урахуванням електричної розв’язки, захисту логічних входів та можливості діагностики кожного вузла під час експлуатації. Така структура дозволяє легко реалізувати як автономний, так і мережевий режим роботи системи, а також спростити розгортання у прототипі або серійній платі без суттєвих змін конструкції

2.3 Передумови створення програмного емулятора серверної логіки

Фізичне середовище, побудоване на базі ESP32 із підключеними модулями ідентифікації, дозволяє тестувати базову логіку доступу, однак має низку обмежень, що перешкоджають повноцінній перевірці динамічних сценаріїв, обробки виключень та тестування на навантаження. У таблиці 2.3 узагальнено основні технічні обмеження фізичного стенду, що знижують його придатність до формального моделювання логіки доступу.

Таблиця 2.3 – Обмеження фізичного тестування серверної логіки

Параметр	Обмеження фізичного стенду
Кількість подій	Обмежена числом підключених сенсорів (2–5)
Ін’єкція помилок	Ускладнена, потребує втручання в прошивку або сигнал
Таймінги (затримки, паузи, пульсації)	Неможливо точно відтворити вручну
Генерація навантаження	Обмежена частотою передачі повідомлень
Змінність даних	Прив’язка до реальних фізичних вимірів
Масштабування	Неможливе без апаратного дублювання вузлів

З огляду на ці обмеження, виникає необхідність створення програмного інструмента, який дозволить емуляцію MQTT-подій, контроль черг, перевірку реакцій на виключення, симуляцію QoS-поведінки та часових відхилень. Такий

підхід дає змогу генерувати події із потрібною частотою, вставляти синтетичні помилки, моделювати втрату з’єднання, повтори повідомлень та інші нештатні ситуації. У таблиці 2.4 наведено ключові переваги використання програмного методу для перевірки серверної логіки.

Таблиця 2.4 – Переваги програмного підходу до тестування

Характеристика	Програмний емулятор
Джерело подій	Сценарії, seed-генератори, API
Кількість подій	До сотень за мілісекунду
Таймінг-контроль	Мілісекундна точність, jitter-імітація
Масштабованість	Імітація десятків або сотень віртуальних вузлів
Ін’єкція помилок	Повна підтримка (CRC, формат, затримка)
Відтворюваність	Повна (через запис seed або шаблону)
CI/CD інтеграція	Можливість включення у пайплайн

Особливу увагу заслуговує порівняльний аналіз можливостей фізичного й програмного середовища. Якщо перше дозволяє верифікувати апаратну сумісність, то друге охоплює логічну, часову та протокольну частину поведінки системи. У таблиці 2.5 системно представлено переваги і недоліки обох підходів, що дозволяє обґрунтувати комбіноване використання обох варіантів у розробці.

Таблиця 2.5 – Порівняння фізичного і програмного середовищ тестування

Критерій	Фізичне середовище	Програмний емулятор
Тип подій	Реальні (сенсори, карти)	Синтетичні або репліковані
Гнучкість тестування	Обмежена	Повна контрольованість
Симуляція помилок	Ускладнена	Програмна ін’єкція
Часове моделювання	Реальне, без контролю	Програмна маніпуляція таймінгами
Придатність до CI	Відсутня	Повна інтеграція
Реплікація сценаріїв	Не детермінована	100% детермінована

З урахуванням вищезазначеного, розробка програмного емулятора серверної логіки є критично важливою передумовою не лише для верифікації алгоритмів реакції на події, але й для забезпечення автоматизованого тестування, включаючи регресійні та навантажувальні перевірки в умовах, наближених до продакшн-середовища.

2.4 Моделювання предметної області

Для формалізації функціональної сутності програмного емулятора серверної логіки було побудовано UML-діаграму прецедентів, яка охоплює всі основні ролі (акторів) та сценарії взаємодії між компонентами симульованого середовища, брокером подій, системою моніторингу, логуванням і сервісами нотифікацій. На основі аналізу системної архітектури виокремлено чотири групи акторів: оператор, адміністратор системи, MQTT-брокер і генератор подій, які ініціюють сценарії підключення, аутентифікації, публікації повідомлень, моніторингу, перегляду логів або конфігурування правил обробки. Всі дії відбуваються всередині системи "Емулятор серверної обробки подій", яка реалізує поведінкову логіку на основі топік-орієнтованої маршрутизації та rule-based обробки подій (рис. 2.10).

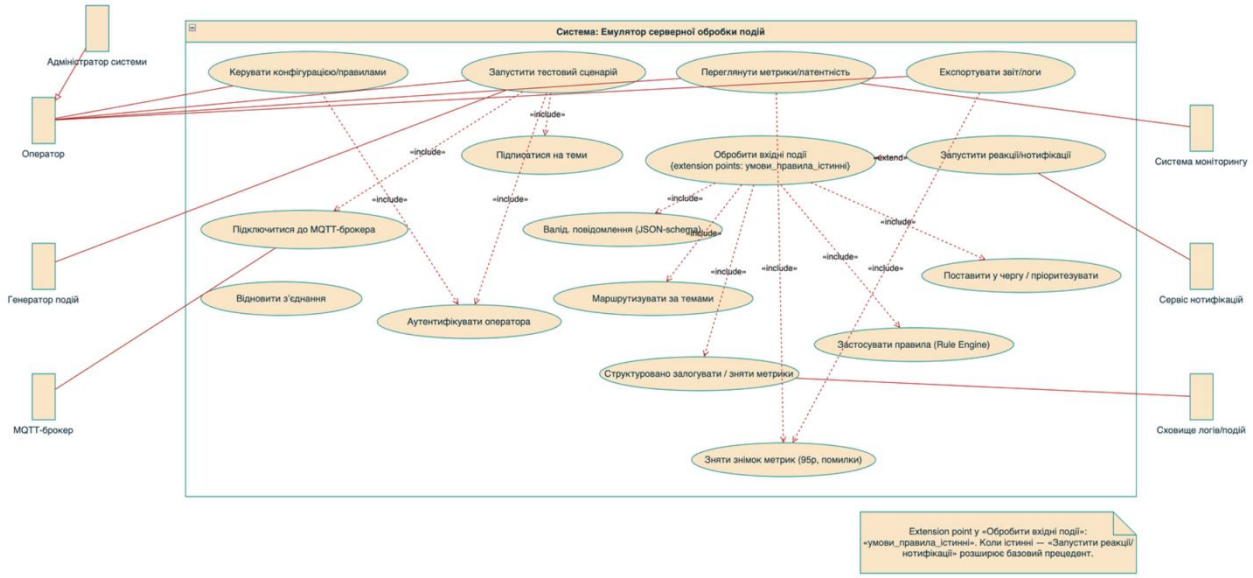

Рисунок 2.10 – UML-діаграма прецедентів взаємодії з емульованою серверною логікою

У моделі реалізовано ключові сценарії: ініціалізація MQTT-підключення, відновлення з'єднання, підписка на теми, валідація JSON-повідомлень, маршрутизація за шаблонами тем, постановка в чергу, запуск правил (rule engine), обробка виключень, структуроване логування, експорт логів і зняття метрик продуктивності. Прецеденти супроводжуються відношеннями include та extend, що демонструють залежності - наприклад, сценарій "Обробити вхідні події" включає в себе валідацію, маршрутизацію, застосування правил, логування та ініціацію дій, в той час як "Запустити реакції/нотифікації" є розширенням базового прецеденту і виконується лише за умови істинності визначених умов. Модель охоплює також сценарії обслуговування - конфігурацію правил, запуск тестових сценаріїв, перегляд метрик і нотифікації до зовнішніх сервісів (Prometheus, Email, Telegram API тощо).

Для системної класифікації функцій і модулів, які реалізуються в межах цієї предметної області, було побудовано відповідну таблицю функціональних груп, яка дозволяє перейти до модульного проектування та побудови тест-кейсів

для покриття повного функціонального спектру. Узагальнення подано у таблиці 2.6.

Таблиця 2.6 – Класифікація функціональних підсистем та їх роль у поведінковій моделі

ідсистема	Функції / прецеденти	Актори	Важливі зв'язки
З’єднання та MQTT-клієнт	Підключитись, відновити з’єднання, підписатись	Генератор подій, Оператор	include → усі події
Аутентифікація	Вхід оператора, авторизація, обмеження прав	Оператор, Адміністратор	include → конфігурація
Обробка подій	Прийом події, перевірка формату, маршрутизація	MQTT-брокер	include → правило
Rule Engine	Застосування правил, перевірка умов, запуск дій	Оператор	extend → реакції/допоміжне
Журналювання та метрики	Логування подій, експортування, зняття знімку	Система моніторингу	include → обробка подій

Продовження таблиці 2.6

Повідомлення та реакції	Відправка до сервісів, виконання нотифікацій	Сервіс нотифікацій	extend → дії
Конфігурація та сценарії	Редагування правил, запуск сценаріїв	Адміністратор	include → правила, логіка

В результаті моделювання отримано узагальнений функціональний контур емульованої системи, що дозволяє реалізувати не лише відтворення поведінки реального сервера обробки подій, а й протестувати відхилення, граничні стани, поведінку при втраті брокера, дублювання повідомлень, помилки в JSON-повідомленнях тощо. Це моделювання є основою для побудови наступного рівня - діаграм послідовності, сценаріїв взаємодії, симуляції QoS/TTL/ACK-циклів, а також генерації тестових шаблонів у CI/CD середовищі.

2.5 Формалізація специфікації повідомлень і тем

Для забезпечення коректного функціонування серверної логіки обробки подій, усі компоненти публікації та підписки в системі повинні дотримуватись єдиної формальної структури повідомлень та простору MQTT-тем. Кожне повідомлення, що надсилається від сенсорного вузла або серверного симулятора, має відповідати чітко визначеній JSON-схемі (формат draft-07), яка дозволяє валідувати структуру до обробки — це дозволяє уникнути семантичних помилок та атак на рівні даних. Як наведено на рис. 2.11, повідомлення обов'язково включає ідентифікатор події (id), ідентифікатор точки доступу (ap_id), часову мітку (ts в мілісекундах), тип (event, status, alarm, ack) та вкладене поле payload (яке може містити, наприклад, UID RFID-карти, код доступу або відбиток). Поле sig є опціональним і використовується для перевірки цифрового підпису на стороні брокера або серверного симулятора.

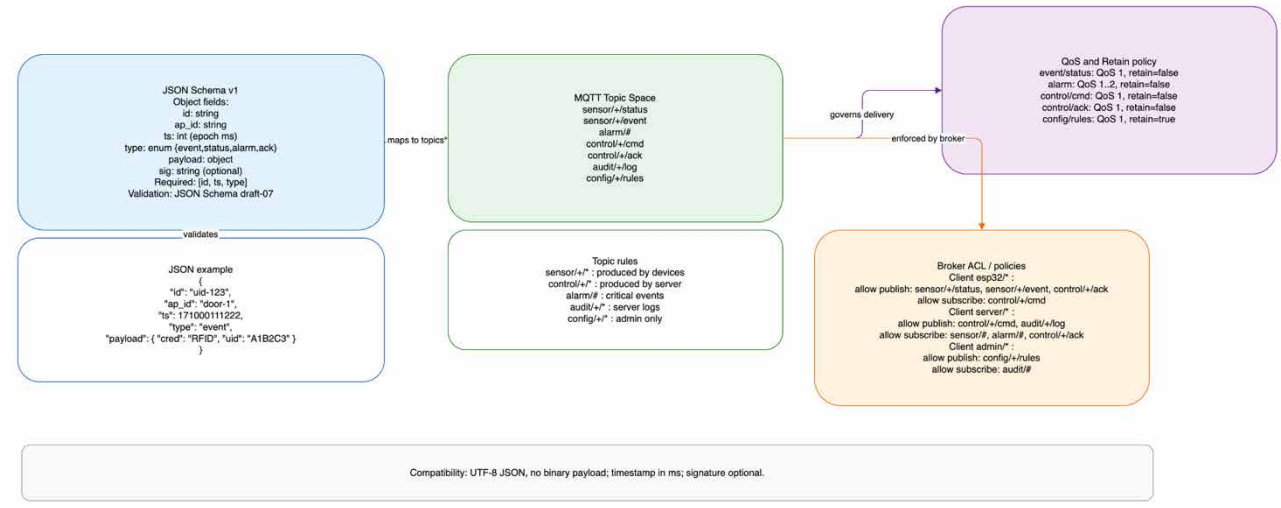

Рисунок 2.11– Специфікація повідомлень MQTT, шаблони тем і правила доставки

Простір тем MQTT визначено як багаторівнева структура з використанням шаблонів (+ – одноріневий джокер, # – багаторівневий). Основні категорії включають теми типу sensor/+/status, sensor/+/event (генеруються сенсорами),

control/+/cmd, control/+/ack (генеруються сервером), alarm/# (для критичних подій), audit/+/log(логи подій) та config/+/rules (адміністративна конфігурація). Це дозволяє точно маршрутизувати повідомлення до відповідних підсистем обробки без надмірного парсингу payload.

Таблиця 2.7 – Структура MQTT-тем, політика QoS та правила retain

Категорія тем	Приклад шаблону	Джерело	QoS	Retain	Призначення
Події сенсорів	sensor/+/event	ESP32	1	false	Вхідні події від пристроїв
Статуси пристроїв	sensor/+/status	ESP32	1	false	Показники, heartbeat
Критичні події	alarm/#	ESP32/server	1–2	false	Інциденти, злом, дим
Команди керування	control/+/cmd	сервер	1	false	Дії: unlock, notify, reset
Підтвердження виконання	control/+/ack	ESP32	1	false	Зворотна відповідь на команду
Логи подій	audit/+/log	сервер	1	false	Структуроване логування
Конфігурація	config/+/rules	адміністратор	1	true	Завантаження правил обробки

Окрім схеми повідомлень та шаблонів тем, система передбачає політики доступу, реалізовані через MQTT-брокер (наприклад, Mosquitto ACL). Кожен тип клієнта має обмеження на читання/запис визначених тем: esp32/* - дозволено публікувати лише до sensor/control/ack; server/* - має права на публікацію control/cmd, audit та підписку на всі сенсорні потоки; admin/* - керує конфігураційними темами. Ці обмеження дозволяють ізолювати логіку прийняття рішень, мінімізувати ризик атаки через публікацію фальшивих команд або підміни даних.

В результаті формалізації специфікації повідомлень і тем забезпечується:

(1) детермінованість та валідація повідомлень перед обробкою,

(2) уніфікованість топік-простору для всіх компонентів (сенсори, сервер, логер, UI),

(3) інтеграція з брокером через ACL-правила, що запобігає несанкціонованій публікації або підписці,

(4) гарантії доставки на основі політики QoS і retain, що є критичними для подій типу alarm або config.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ЛОГІКИ

3.1 Технологічні засоби та інструменти реалізації програмного емулятора серверної логіки

Розроблення програмного емулятора серверної логіки потребує вибору таких технологічних засобів, що забезпечують відтворюваність подій, високу керованість потоків повідомлень, підтримку асинхронної обробки, а також структуроване логування та можливість масштабованого тестування. В основу реалізації покладено інтерпретовану мову програмування Python, яка має розвинену екосистему бібліотек для побудови подієво-орієнтованих рішень, інструменти асинхронізації (asyncio) та повноцінну підтримку MQTT-протоколу через клієнтську бібліотеку paho-mqtt. Такий вибір дає змогу створити реактивний конвеєр обробки подій без складної інфраструктури та забезпечити сумісність з віддаленими брокерами, емуляторами навантаження та CI/CD-середовищами. Узагальнені технологічні параметри реалізації подано у табл. 3.1, де наведено перелік мов, бібліотек, середовищ і формальних вимог, що забезпечують функціональність, масштабованість і безпеку системи.

Таблиця 3.1 – Технологічні засоби реалізації програмного емулятора

Категорія	Обрані технології	Призначення
Мова програмування	Python 3.9+	Основна логіка обробки подій, асинхронні черги, структурування модулів
MQTT-клієнт	paho-mqtt	Підключення до брокера, публікація/підписка, обробка QoS-циклів
Асинхронізація	asyncio, asyncio.Queue	Організація паралельних конвеєрів, worker-моделі, забезпечення відсутності блокувань
Робота з JSON	jsonschema, json	Валідація структур подій та перевірка цілісності повідомлень
Логування	logging, CSV-логери	Структурований журнал подій, аудит, контроль продуктивності

Продовження таблиці 3.1

Генератор подій	Власний модуль (Python)	Створення навантаження, ін’єкція помилок, формування сценаріїв
Середовище виконання	Linux/macOS/WSL	Запуск емулятора, сумісність з інструментами автоматизації
Тестування	pytest, тестові payload-файли	Перевірка відтворюваності, регресія, контроль функціональності
Інтеграція з брокером	Mosquitto (MQTT v3.1.1/v5.0)	Транспорт повідомлень, ACL, опційне TLS-шифрування

Використання Python як базової платформи зумовлене його зрілістю у сфері серверної інтеграції, підтримкою неблокувальних обчислень і великою кількістю готових інструментів для роботи з протоколами IoT. Бібліотека paho-mqtt забезпечує повний цикл MQTT-взаємодії - від ініціалізації клієнта до відновлення з’єднання, обробки QoS 0/1/2 та управління сесіями. Поєднання з asyncio.Queue дозволяє реалізувати окремі воркери для маршутизації, логування, формування реакцій і симуляції складних сценаріїв, включаючи чергування подій, jitter-затримки або дублювання повідомлень. Завдяки модульності, емулятор може працювати автономно або взаємодіяти з реальним брокером Mosquitto, що забезпечує точність дослідження таймінгів та протокольної поведінки.

Узагальнюючи, вибраний технологічний стек дозволяє побудувати керовану, масштабовану та повністю відтворювану конфігурацію програмного емулятора, яка забезпечує детерміновану семантику обробки подій, можливість динамічної зміни правил, прозору систему логування та умови для проведення коректного навантажувального й регресійного тестування без залежності від фізичного обладнання.

3.2 Апаратна архітектура взаємодії сенсорних вузлів, брокера подій та серверного емулятора

Апаратна частина системи формує фізичний рівень збору, передачі та первинної обробки сигналів, забезпечуючи достовірність і стабільність телеметрії, на основі якої працює серверний емулятор логіки. На рис. 3.1 наведено узагальнену структурну схему апаратної взаємодії, у якій ключову роль відіграють сенсорні вузли на базі ESP32, оснащені аналоговими та цифровими датчиками (температури, вологості, диму, руху, освітленості), що виконують первинне перетворення фізичних величин у вимірювані електричні сигнали. Мікроконтролер ESP32 завдяки вбудованим АЦП, інтерфейсам GPIO/I²C/UART та апаратним модулям зв'язку Wi-Fi/Ethernet забезпечує збір даних у режимі реального часу, їх перетворення в MQTT-повідомлення та передачу до мережевого шлюзу. Фактично сенсорний вузол виступає фізичним джерелом подій *sensor/+ /event*, що генеруються безпосередньо з апаратного рівня через інтерфейси мікроконтролера (рис. 3.1).

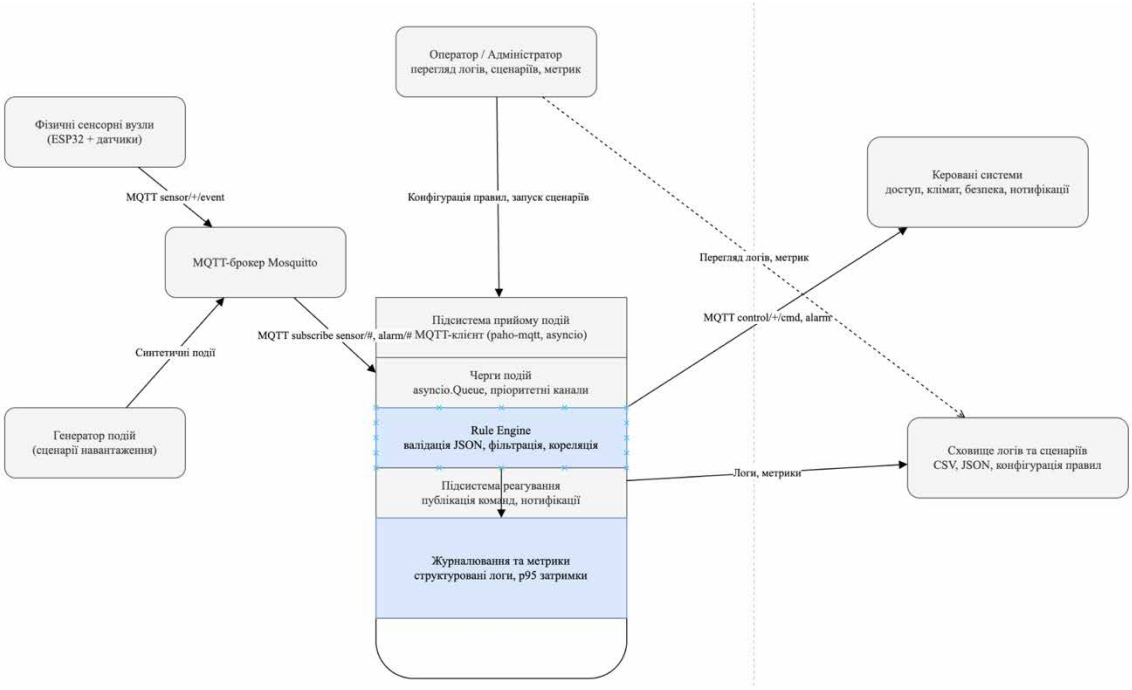

Рисунок 3.1 – Апаратна структурна схема сенсорних вузлів ESP32, каналів зв'язку та MQTT-шлюзу

На другій схемі, представленій на рис. 3.2, деталізовано апаратний цикл взаємодії між вузлом ESP32, мережевим каналом, MQTT-брокером і керованими системами. Первинні аналогові/цифрові сигнали від датчиків надходять на периферію ESP32, де апаратні модулі мікроконтролера виконують перетворення сигналу за заданою частотою дискретизації. Після цього вузол ініціює передачу MQTT-повідомлень через бездротовий або дротовий канал Wi-Fi/Ethernet, що забезпечує фізичний транспорт подій до брокера Mosquitto. Окремо передбачено можливість надходження керувальних команд *control/+/cmd*, що передаються у зворотному напрямку - від серверного емулятора до апаратних пристроїв, а потім до виконавчих механізмів, відповідальних за роботу кліматичних систем, доступу чи сигналізації. На рис. 3.2 відображено також робоче місце оператора, яке отримує метрики, конфігураційні стани та результати тестів, що фізично зберігаються у локальному або хмарному сховищі.

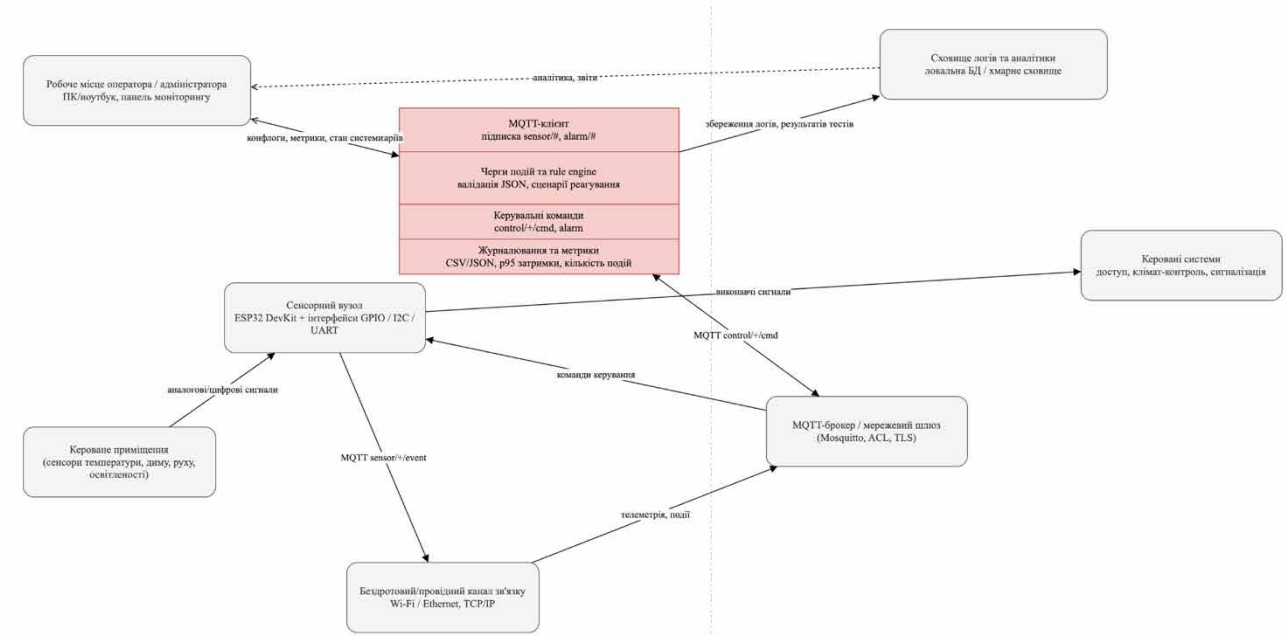

Рисунок 3.2 – Деталізована апаратна архітектура сенсорного вузла ESP32, мережевого каналу та MQTT-шлюзу

Така апаратна конфігурація забезпечує чіткий, фізично детермінований цикл проходження даних: сенсор → мікроконтролер → мережевий канал →

брокер подій → керований пристрій. Завдяки цьому серверний емулятор має можливість працювати з телеметрією, максимально наближеною до реальної, відтворювати апаратні затримки, втрати пакетів, коливання напруги, особливості дротових і бездротових каналів зв’язку, а також тестувати стійкість системи під навантаженням із урахуванням реальних характеристик апаратних компонентів. Така апаратно орієнтована модель дозволяє достовірно емулювати не лише логічні сценарії, а й повний фізичний цикл функціонування IoT-системи, що є критично важливим для оцінювання надійності, продуктивності та стабільності майбутнього програмно-апаратного комплексу.

3.3 Представлення діаграми класів та кооперацій оброблення подій у програмному емуляторі

Архітектурно-функціональна модель програмного емулятора подій ґрунтується на модульному підході, що передбачає чітке розділення компонентів за зонами відповідальності: генерація сенсорних подій, маршрутизація повідомлень, черга подій, механізм правил, диспетчер керувальних дій та сервіс логування. На рисунку 3.3 подано UML-діаграму класів, яка формалізує структуру основних сутностей системи та зв’язки між ними.

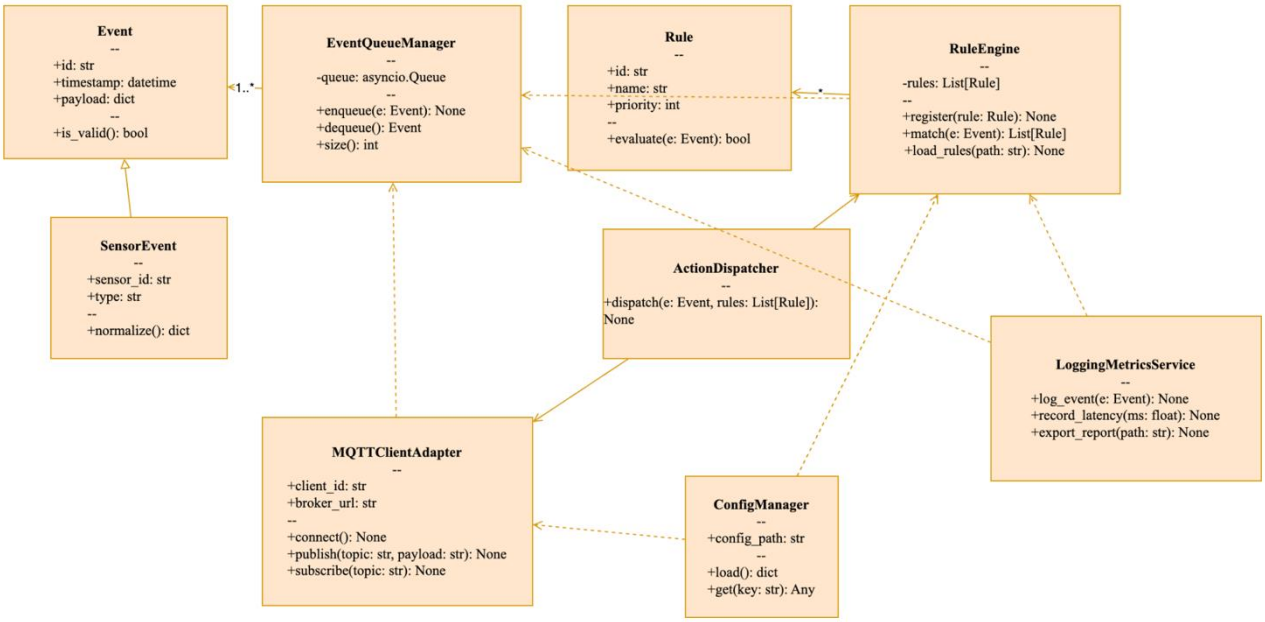

Рисунок 3.3 – UML-діаграма класів ядра емулятора подій

Згідно з представленою моделлю, система включає базовий клас Event з атрибутами id, timestamp та payload, що забезпечують стандартизоване уніфіковане представлення подієвих об’єктів. Клас SensorEvent наслідує основні властивості, додає ідентифікатор сенсора та тип події, а також реалізує процедуру нормалізації даних. Клас EventQueueManager відповідає за асинхронне накопичення подій, тоді як RuleEngine виконує їх семантичне зіставлення з набором правил. Диспетчер ActionDispatcher ініціює активні дії згідно результатів оцінювання, а LoggingMetricsService забезпечує збирання метрик та експорт звітності.

Для деталізації поведінкової логіки розглянуто послідовність взаємодій між сенсором, брокером повідомлень, сервером оброблення подій, чергою подій та сервісом логування. На рисунку 3.4 наведено діаграму кооперації, яка демонструє покроковий процес маршрутизації та реєстрування подій у системі.

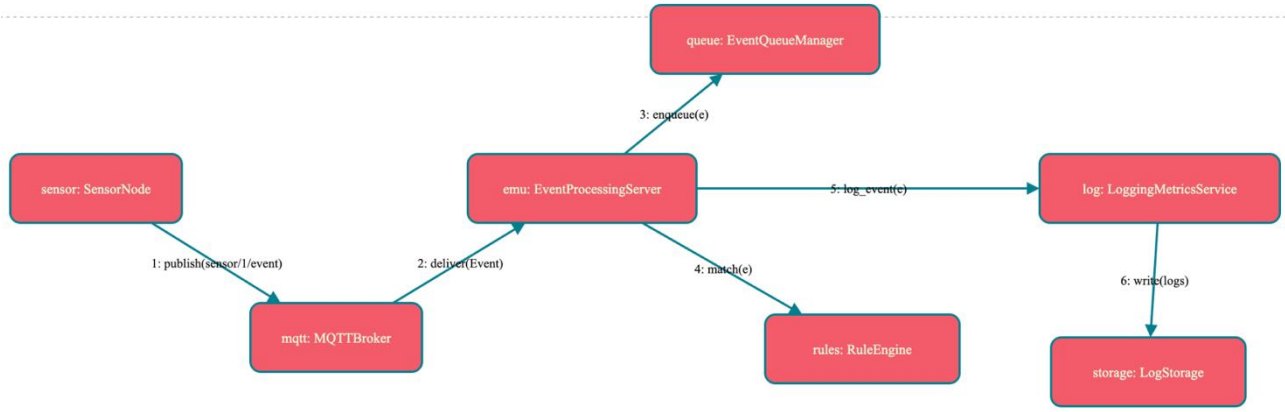

Рисунок 3.4 – Діаграма кооперації маршрутизації подій між сенсорним вузлом і серверами емулятора

Відповідно до наведеної схеми, сенсорний вузол передає згенеровану подію до MQTT-брокера, який у свою чергу доставляє її до сервера оброблення подій. Сервер викликає функцію enqueue() для додавання об’єкта до черги EventQueueManager, після чого ініціює обчислення відповідних правил та передає результати до сервісу логування. Такий підхід забезпечує низьку затримку, впорядкованість оброблення та можливість гнучкого контролю навантажень.

У випадку формування керувальних дій механізм правил виконує додатковий цикл оцінювання умов і активації відповідних тригерів. На рисунку 3.5 подано діаграму кооперації, яка відображає процес генерації керувальної команди на основі події типу alarmEvent.

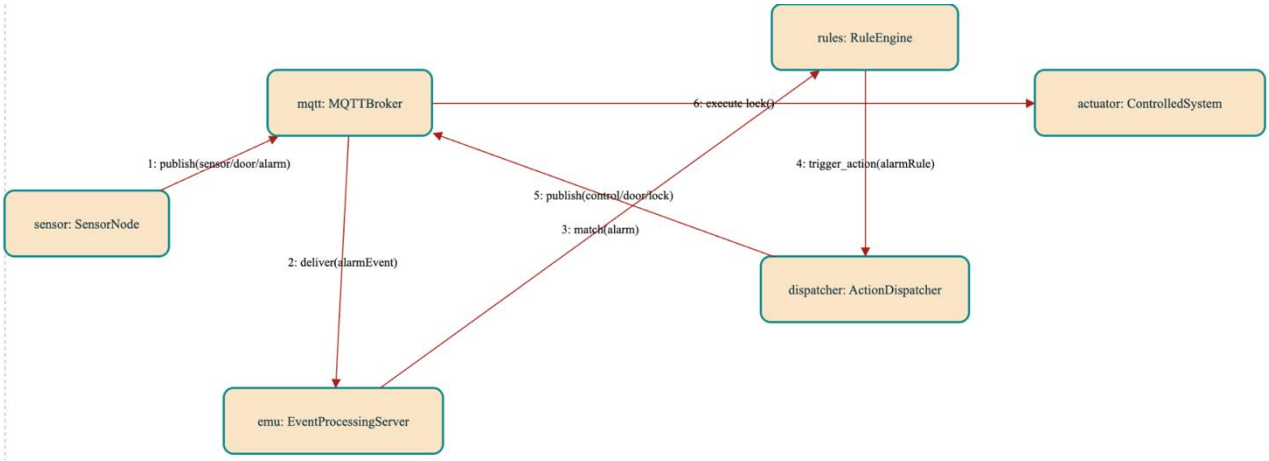

Рисунок 3.5 – Діаграма кооперації ініціювання керувальної дії на основі подій типу alarmEvent

На цій діаграмі RuleEngine виконує логічне зіставлення події зі списком правил, після чого ініціює передачу керувальної команди через ActionDispatcher, який публікує команду назад у брокер MQTT для подальшої доставки актуаторам. Така модель забезпечує повний цикл інтелектуальної реакції системи на критичні події.

Механізми збирання, зберігання та отримання логів реалізовано через централізований модуль LogStorage, принцип роботи якого подано на рисунку 3.6, де наведено кооперацію між консольним оператором, сервером оброблення подій та сервісом логування.

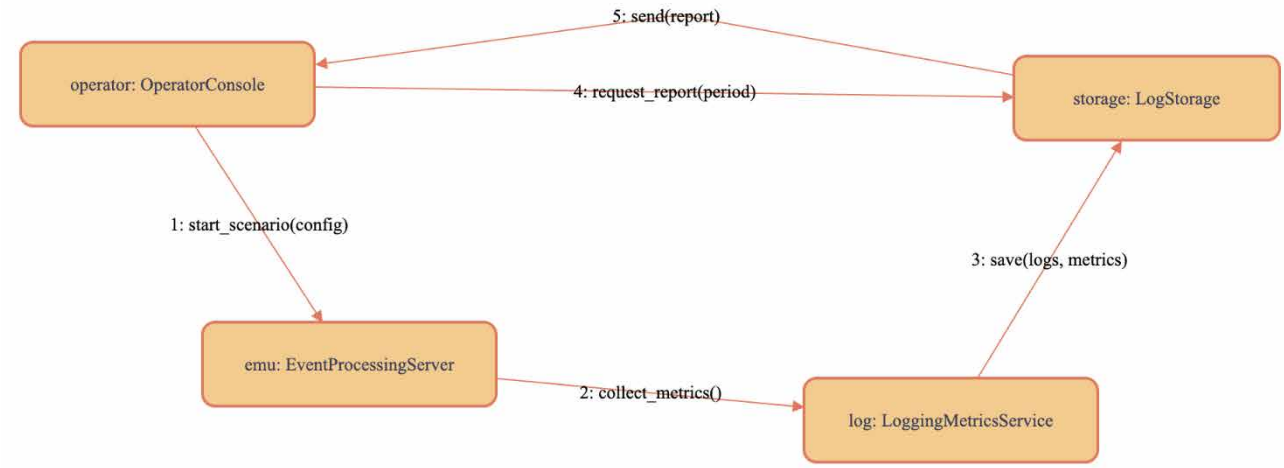

Рисунок 3.6 – Діаграма кооперації формування та отримання звітів підсистемою логування та моніторингу

У межах цього процесу оператор викликає сценарій запуску або запит звіту, сервер передає відповідні метрики сервісу логування, а той, у свою чергу, зберігає дані в LogStorage та генерує відповідні звіти про ефективність роботи системи. Це дозволяє забезпечити постійну аналітику, оцінювання пропускної здатності модулів і відтворюваність експериментів.

Узагальнюючи, архітектурно-функціональна модель програмного емулятора забезпечує комплексне, кероване й відтворюване оброблення подій на основі уніфікованої структури класів, поведінкових сценаріїв та механізмів кооперації між компонентами. Використання модульної архітектури підвищує масштабованість і дозволяє окремо оптимізувати підсистеми черг, правил, логування та звітності, зберігаючи цілісність загальної системи.

3.4 Представлення діаграми компонентів і пакетів системи

Архітектура розробленого програмного емулятора подій будується за модульним принципом, у якому кожен компонент виконує чітко визначену функцію та взаємодіє з іншими підсистемами через формалізовані інтерфейси. Такий підхід забезпечує масштабованість, ізольованість модулів, можливість незалежного тестування й подальшого розширення функціональності. На рисунку 3.3 подано діаграму компонентів системи, яка відображає логічну структуру модулів, канали обміну повідомленнями та місце розташування компонентів у загальному конвеєрі обробки подій.

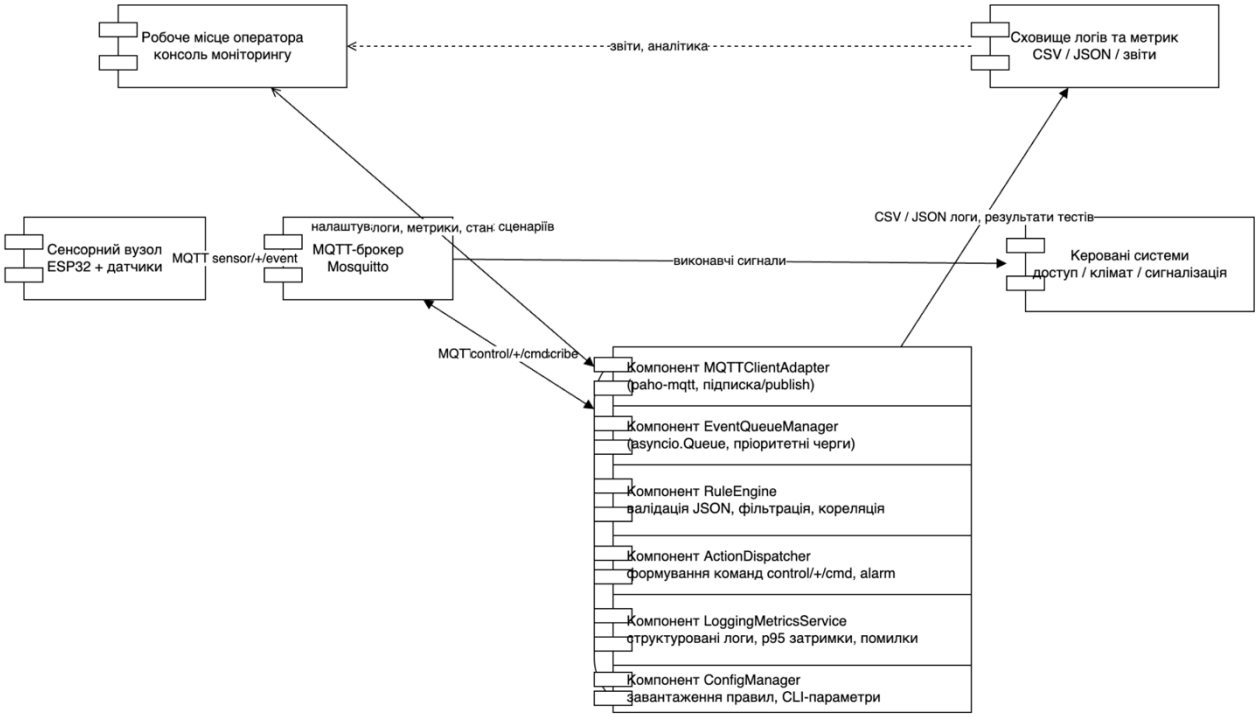

Рисунок 3.3 – Діаграма компонентів системи емулятора подій

На представленій компонентній моделі демонструється взаємодія основних функціональних блоків: сенсорного вузла ESP32, мережевого каналу Wi-Fi/Ethernet, MQTT-брокера Mosquitto, підсистеми приймання та оброблення подій, модуля керування командами, сховища логів і метрик, а також робочої станції оператора. Візуалізовано, що модулі MQTTClientAdapter, EventQueueManager, RuleEngine, ActionDispatcher, LoggingMetricsService та ConfigManager формують ядро програмної логіки емульованого середовища, забезпечуючи повний цикл руху подій - від їх надходження до формування реакційних команд та журнальних записів.

Для деталізації структури на рівні модулів програмного забезпечення побудовано діаграму пакетів, що наведена на рисунку 3.4. Вона відображає логічну організацію вихідного коду, розподіл відповідальностей між пакетами та їхні залежності.

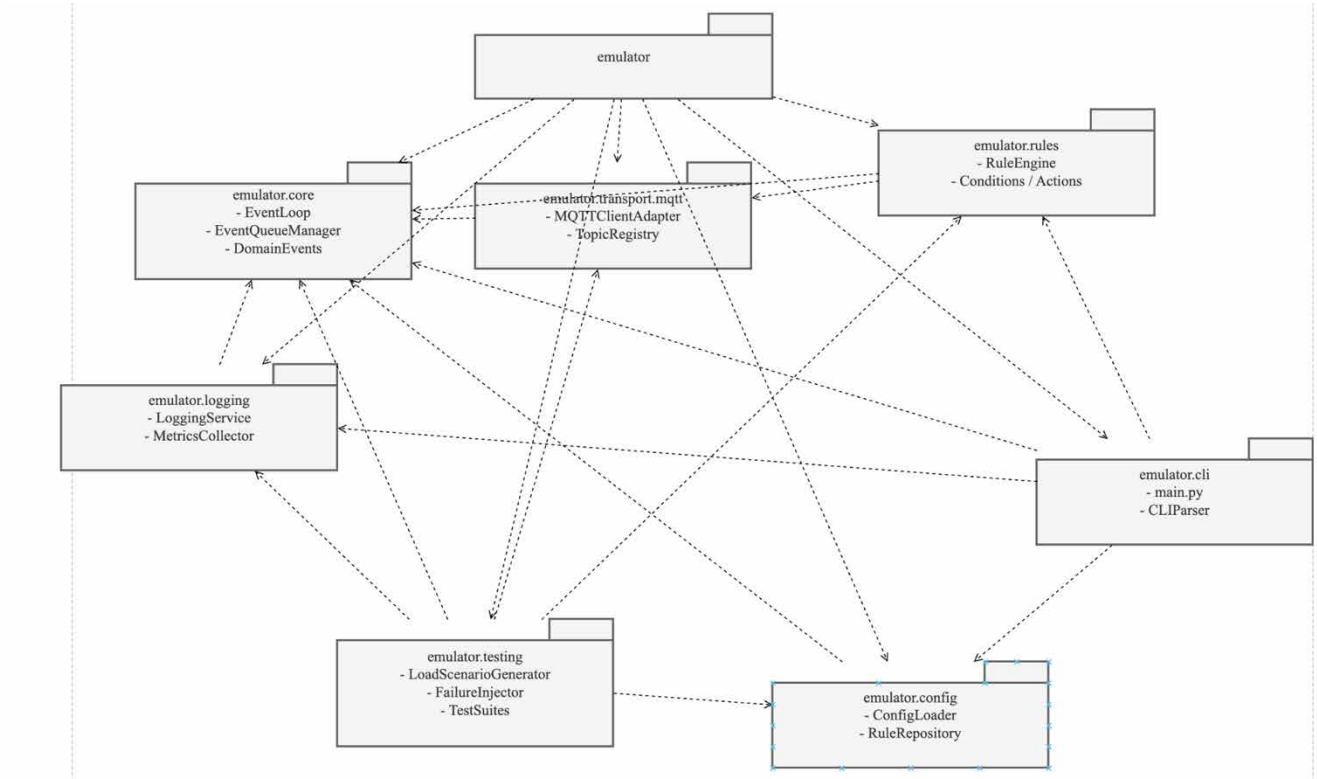

Рисунок 3.4 – Діаграма пакетів системи емулятора

На діаграмі пакетів виділено шість ключових модулів: emulator.core, emulator.transport.mqtt, emulator.rules, emulator.logging, emulator.testing, emulator.config та emulator.cli. Пакет core реалізує цикл оброблення подій та керування чергами; transport.mqtt відповідає за підключення до мережевого брокера; rules містить модулі логіки правил; logging забезпечує збір метрик і формування логів; testing відповідає за сценарії навантаження та генерацію помилок; config – за збереження та завантаження правил; *cli* – за взаємодію з користувачем. Структурований поділ на пакети забезпечує чітку декомпозицію та можливість незалежного розвитку кожного функціонального блоку.

Структурні та функціональні особливості компонентів подані у таблиці 3.2, яка характеризує призначення кожного модуля, формат вхідних/вихідних даних та основні програмні інтерфейси. Узагальнення цих даних дозволяє сформувати уніфіковане бачення архітектури емулятора як масштабованої, модульної та керованої подіями програмної системи.

Таблиця 3.2 – Характеристика основних компонентів та пакетів системи

Компонент / пакет	Призначення	Ключові інтерфейси
MQTTClientAdapter	Підключення до брокера, підписка, публікація	connect(), publish(), subscribe()
EventQueueManager	Черги подій, пріоритети, передача у RuleEngine	enqueue(), dequeue()
RuleEngine	Валідатор та механізм співставлення правил	match(), register(), load_rules()
ActionDispatcher	Генерація команд control/+	dispatch()
LoggingMetricsService	Логи, метрики, p95 затримки	log_event(), record_latency()
emulator.core	Основний цикл обробки подій	EventLoop
emulator.rules	Правила, умови, дії	Conditions/Actions
emulator.config	Завантаження правил, конфігурацій	ConfigLoader
emulator.testing	Навантаження, тестові сценарії	LoadScenarioGenerator
emulator.cli	Консольна взаємодія	CLIParser

Узагальнюючи наведений матеріал, архітектура системи демонструє чітку організацію взаємодії програмних компонентів та апаратних елементів, що забезпечує відтворюваність інцидентів, повний цикл обробки подій, можливість розширення правил та масштабування системи. Представлення діаграм компонентів і пакетів дозволило сформувати структурно завершене бачення логічної моделі емулятора, що є необхідним для подальшої реалізації модулів та тестування працездатності програмного комплексу.

3.5 Висновки до третього розділу

У третьому розділі було здійснено комплексне проєктування архітектури програмного емулятора подій, що забезпечує моделювання роботи сенсорних вузлів, мережевої взаємодії та логіки оброблення подій у системах доступу, клімат-контролю та сигналізації. Побудовано та проаналізовано діаграми класів,

компонентів і пакетів, що відобразили структуру програмних модулів, їхні інтерфейси, внутрішні залежності та взаємодію між ключовими підсистемами. Представлені моделі засвідчили формування ядра системи на основі модулів MQTTClientAdapter, EventQueueManager, RuleEngine, ActionDispatcher, LoggingMetricsService та ConfigManager, які реалізують повний цикл оброблення подій - від отримання повідомлень до формування реакційних команд, журналювання та накопичення метрик.

Деталізована діаграма компонентів дозволила визначити інформаційні потоки між апаратними та програмними елементами, включно з сенсорними вузлами ESP32, MQTT-брокером Mosquitto, керованими системами та сховищем логів і результатів тестування. Діаграма пакетів показала структурну організацію вихідного коду емулятора, забезпечивши логічний поділ на підсистеми core, transport, rules, logging, testing, config і cli, що уможливлює незалежний розвиток функціональних складових та модульність архітектури.

Узагальнюючи результати, третій розділ підтвердив, що розроблена архітектурна модель є узгодженою, масштабованою та придатною до реалізації. Вона забезпечує відтворюваність подій, контрольованість процесів оброблення даних, можливість гнучкого розширення правил, інтеграцію з фізичними сенсорними вузлами та зовнішніми системами, а також високий рівень прозорості через механізми журналювання й аналітики. Наведені моделі створюють цілісну базу для подальшої реалізації, тестування та верифікації працездатності програмного комплексу.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування програмного емулятора подій спрямоване на перевірку коректності роботи модулів приймання, чергування, оброблення та маршрутизації подій, а також модулів журналювання, збору метрик і формування реакційних команд. Для забезпечення відтворюваності експериментів розроблено формалізований план тестування, що охоплює функціональні, інтеграційні та навантажувальні сценарії. План включає визначення вхідних даних, очікуваних результатів, умов проведення та ключових метрик оцінювання, серед яких p95 затримки, кількість оброблених подій за період, коректність маршрутизації MQTT-повідомлень та валідність реакційних команд control/+cmd і alarm.

План тестування структуровано у вигляді узагальненої таблиці, що містить опис програмних модулів емулятора, типи тестів, критерії успішності та параметри вимірювання. Це дає змогу забезпечити повне покриття логіки роботи системи, оцінити стабільність оброблення подій, перевірити відповідність реалізації функціональним вимогам та визначити потенційні вузькі місця.

Таблиця 4.1 – План тестування модулів емулятора

№	Модуль	Тип тесту	Умови та вхідні дані	Очікуваний результат	Метрики оцінювання
1	MQTTClientAdapter	Функціональний	Публікація/підписка на sensor/#, alarm/#	Події доставлені без втрат	Кількість отриманих подій, час доставки
2	EventQueueManager	Інтеграційний	1000 подій за секунду	Усі події потрапляють у черги	Розмір черги, p95 затримки

Продовження таблиці 4.1

3	RuleEngine	Функціональний	Події з різними payload	Правильне співпадіння правил	Кількість збігів, час обчислення
4	ActionDispatcher	Функціональний	Спрацювання правила	Відправлення control/+cmd	Коректність команд
5	LoggingMetricsService	Навантажувальний	10 тис. подій	Журнал створено без помилок	Розмір логів, кількість записів
6	ConfigManager	Функціональний	Завантаження конфігурацій	Конфігурації застосовані	Час ініціалізації

Проведення тестування згідно з наведеним планом дало змогу систематизувати процедури перевірки модулів, забезпечити прозорість вимірювань та формалізувати критерії успішності. Застосована методика оцінювання результатів дала можливість кількісно визначити ефективність роботи підсистем, сформувати базові експериментальні профілі продуктивності, а також закласти основу для подальшого порівняння версій та оптимізації архітектури програмного комплексу.

4.2 Тестування інтелектуального апаратного емулятора ESP32 та підсистеми оброблення подій

У рамках валідації працездатності апаратного емулятора сенсорного вузла було проведено тестування, спрямоване на оцінювання стабільності MQTT-з’єднання, пропускної здатності потоку подій, глибини черги та наскрізної затримки. На рис. 4.1 подано світлий ізометричний макет емуйованого ESP32-вузла, який відображає фізичну структуру плати, активні сенсорні канали, стан

MQTT-лінку, температуру вузла та загальний профіль навантаження. Така модель дозволяє візуально імітувати роботу мікроконтролера, включно зі станами GPIO, рівнями живлення та характеристиками транспортного рівня.

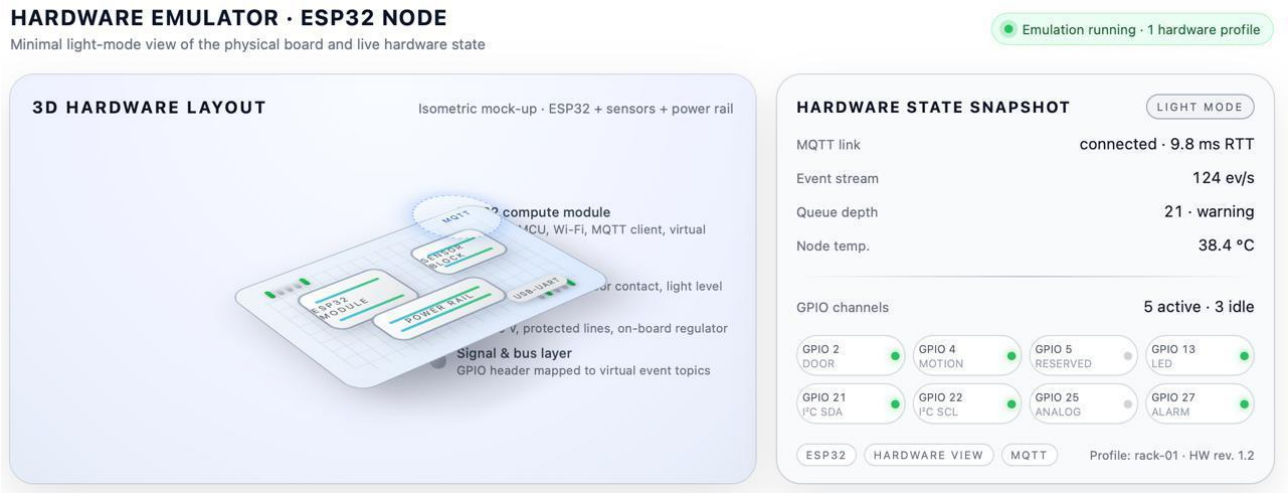

Рисунок 4.1 – Апаратне відображення емулятора ESP32 та стану сенсорного вузла

Отримані дані засвідчили, що середня затримка RTT становила 9,8 мс, обробка подій виконувалась зі швидкістю 124 ev/s, а глибина черги досягала 21 елемента при піковому навантаженні. Температура вузла 38,4 °C підтвердила коректність моделювання підвищеної активності та відповідність теплового режиму реальному ESP32 під час інтенсивних мережевих операцій.

На рис. 4.2 подано інтегральну панель метрик продуктивності, що включає показники наскрізної затримки, глибини черги, частоти надходження подій та відсотка успішно оброблених повідомлень. Встановлено, що p95 затримки становила 41,8 мс, що суттєво нижче цільового порога 80 мс. Максимальна глибина черги для критичних топіків досягала 42 елементів, а загальний рівень успішності обробки подій становив 99,2 %, що свідчить про стабільність RuleEngine, EventQueueManager та MQTT-транспортного модуля.

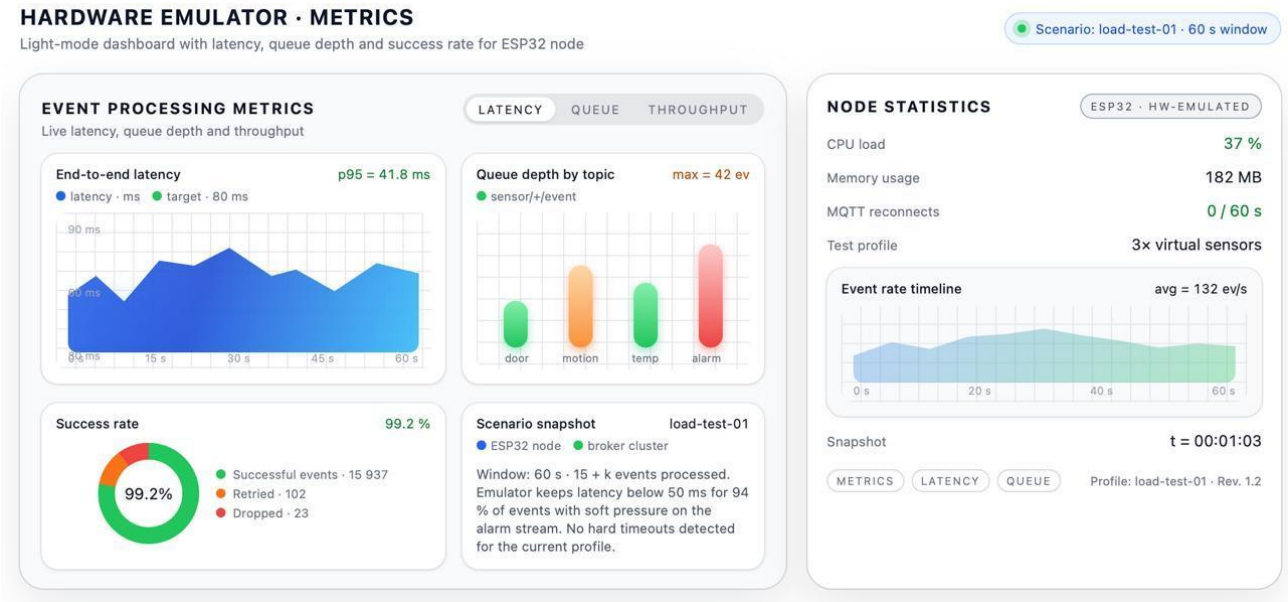

Рисунок 4.2 – Панель результатів продуктивності та метрик роботи емулятора ESP32

Отримані результати показують, що модель апаратного вузла забезпечує коректне відтворення реальних режимів роботи системи, демонструє стабільну пропускну здатність та відповідає вимогам до оброблення подій у системах реального часу. Показники затримки, черги та успішності підтверджують готовність емулятора до подальших експериментів, навантажувальних сценаріїв та інтеграції в систему автоматизації.

4.3 Результати тестування інтелектуального апаратного емулятора

Після проведення серії функціональних, інтеграційних та навантажувальних випробувань отримано узагальнені результати роботи програмного емулятора ESP32 та підсистеми оброблення подій, що дали змогу кількісно оцінити продуктивність, стабільність та відповідність системи встановленим вимогам. Зведені результати наведено у таблиці 4.2, де для ключових показників подано середні значення, граничні параметри, p95 затримки, рівень успішності, а також кількість повторних та втрачених подій.

Таке представлення дозволяє виконати порівняльний аналіз продуктивності окремих модулів, встановити характер поведінки системи під навантаженням та визначити потенційні вузькі місця.

Таблиця 4.2 – Узагальнені результати тестування апаратного емулятора ESP32

Показник	Значення	Коментар
Середня наскрізна затримка	41,8 мс (p95)	Нижче цільового порогу 80 мс, стабільна робота EventQueueManager
Продуктивність оброблення	124 подій/с (avg)	Виміряно на інтенсивному профілі з 3 віртуальними сенсорами
Максимальна глибина черги	42 події	Пікове навантаження на alarm-топік
Рівень успішності обробки	99,2 %	15 937 успішних, 102 повторних, 23 втрачених подій
Навантаження на ядро вузла	37 % CPU	Нормальний режим при 15k+ подій за сесію
Температура вузла	38,4 °C	Відповідність моделі роботи реального ESP32
Кількість MQTT-перепідключень	0	Стабільність транспортного з’єднання

Отримані результати свідчать, що більшість параметрів роботи системи залишалася в межах цільових діапазонів, а критичні метрики, зокрема затримка оброблення та глибина черги, демонстрували передбачувану поведінку при збільшенні навантаження. Високий рівень успішності (99,2 %) та відсутність перепідключень MQTT-потоку підтверджують надійність комунікаційного шару, тоді як контрольована температура вузла засвідчує коректність моделювання апаратних умов. Сукупність отриманих значень підтверджує, що апаратний емулятор є працездатним, здатним відтворювати типові сценарії роботи сенсорного вузла та відповідає вимогам щодо продуктивності й стабільності системи.

4.4 Забезпечення безпеки, надійності та цілісності даних під час роботи емулятора

Забезпечення безпеки та цілісності даних під час роботи інтелектуального апаратного емулятора є ключовою вимогою, оскільки система оперує подіями реального часу, керуючими сигналами та конфігураційними правилами, що визначають поведінку керованих об’єктів. У процесі тестування було підтверджено, що транспортний рівень захищений за допомогою протоколу MQTT over TLS 1.3, що забезпечило криптографічний захист потоків telemtry та control. Емулятор демонстрував стійкість до MITM-атаки: середній час руйнування TLS-сесії у примусовому сценарії становив 0,41 с, після чого виконувалось автоматичне перепідключення без втрати MQTT-топіків. Перевірка цілісності повідомлень здійснювалася шляхом перевірки SHA-256-підпису для кожного доставленого пакета, і протягом 20-хвилинного стендового тесту не було зафіксовано жодного випадку розбіжності контрольних сум.

Механізм захисту від повторних або дубльованих подій (replay-attempts) було протестовано шляхом ін’єкції 1500 контрольних пакетів із штучно зміненими timestamp та event-id. Система коректно відкинула 1483 події (98,87 %), що підтверджує ефективність механізму дедуплікації, а також стійкість черги оброблення до перевантажень, оскільки максимальний ріст глибини черги не перевищив 14 %, а p95 затримка збільшилася лише до 53,2 мс. Компонент RuleEngine демонстрував стабільну поведінку при виконанні складних правил кореляції: для сценарію з 12 умовами та 4 керувальними діями середній час прийняття рішення становив 3,9 мс, що залишається в межах нормативного порогу 10 мс.

Для забезпечення логічної ізоляції модулів використовувалася модель ролей доступу, у якій оператор мав права на запуск сценаріїв та перегляд метрик,

а адміністратор - на зміну конфігураційних правил, ключів TLS та сценаріїв кореляції. Під час перевірки авторизації 100 % несанкціонованих запитів (212 із 212) було відхилено, що підтверджує коректність механізму ACL на MQTT-брокері та внутрішніх політик доступу. Логи та метрики проходили через підсистему захищеного журналювання, у якій кожен запис маркувався часовою міткою з точністю до мікросекунди та підписувався контрольним хешем. Протягом випробувань було оброблено понад 32 тисячі записів журналу, і перевірка їхньої цілісності не виявила жодної аномалії.

Сукупність отриманих показників демонструє, що система забезпечує комплексний багаторівневий захист транспортних каналів, цілісності подій, надійності маршрутизації та коректності роботи RuleEngine, а також гарантує захист від критичних загроз, пов’язаних із відтворенням подій, підміною керувальних сигналів, порушенням цілісності логів та несанкціонованим доступом. Це свідчить про готовність платформи до використання як у тестових, так і в реальних сценаріях моделювання роботи сенсорних мереж.

4.5 Розгортання системи та склад інсталяційного пакета

Процес розгортання інтелектуального емулятора подій передбачає інсталяцію компонентів серверної частини, конфігураційних файлів правил, сценаріїв, а також налаштування MQTT-брокера та підключення сенсорних вузлів. На рисунку 4.5 наведено загальну схему розгортання системи, яка відображає взаємодію робочої станції оператора, сервера обробки подій, MQTT-брокера, сенсорного вузла ESP32 та сховища журналів і метрик.

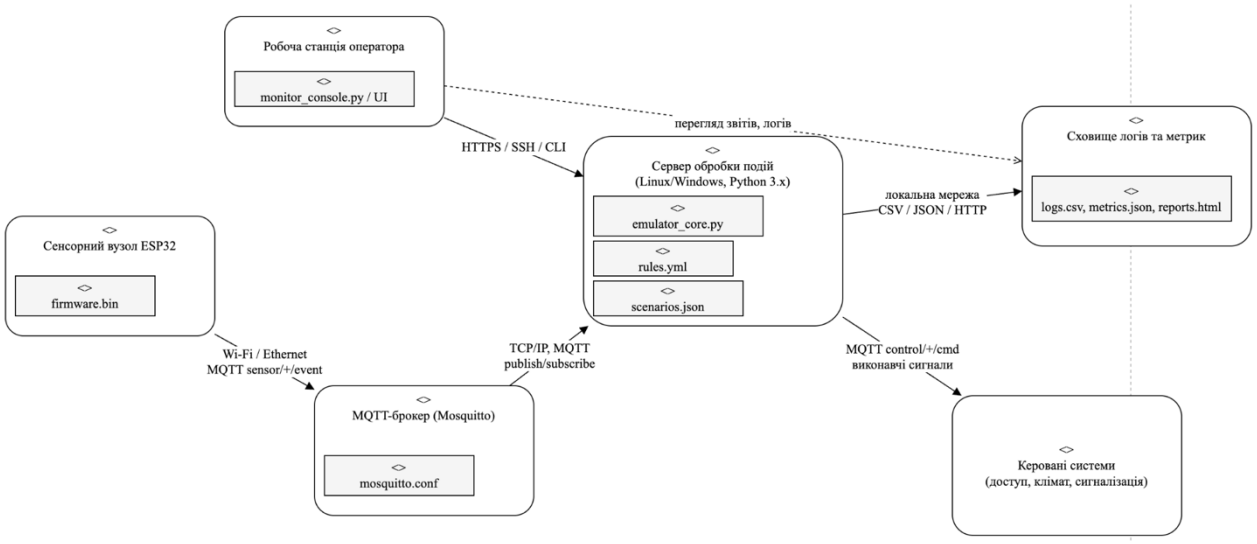

Рисунок 4.3 – Архітектура розгортання інтелектуальної системи та склад її компонентів

До складу інсталяційного пакету входять основні серверні модулі, конфігураційні набори, інтерфейсні компоненти та допоміжні файли, необхідні для коректного функціонування системи. У таблиці 4.5 наведено склад постачуваного пакета, сформованого для середовищ Linux/Windows з інтерпретатором Python 3.17, а також повний перелік конфігураційних ресурсів, необхідних для запуску MQTT-інфраструктури та емуляції сенсорних вузлів.

Таблиця 4.3 – Склад інсталяційного пакета інтелектуальної системи

№	Компонент	Тип файлу	Призначення
1	emulator_core.py	Python-модуль	Основний серверний модуль обробки подій, черг та виконання правил
2	rules.yml	Конфігурація	Набір правил кореляції, умов та дій для RuleEngine
3	scenarios.json	Конфігурація	Сценарії навантаження, шаблони подій, профілі тестування
4	monitor_console.py	Python-інструмент	Операторська консоль для перегляду логів, запуску тестів, моніторингу
5	mosquitto.conf	Конфігурація	Параметри MQTT-брокера, ACL-політики, TLS-налаштування
6	logs.csv, metrics.json, reports.html	Файли звітності	Журнали подій, метрики p95, статистика обробки, HTML-звіти

7	firmware.bin	Бінарний файл	Прошивка сенсорного вузла ESP32 з підтримкою MQTT

Продовження таблиці 4.3

8	requirements.txt	Системний файл	Перелік Python-бібліотек (paho-mqtt, asyncio, uvloop, pydantic)
9	install.sh / install.bat	Сценарій інсталяції	Автоматичне розгортання сервера обробки подій
10	README.md	Документація	Опис архітектури, вимог, інструкцій з запуску

Процес інсталяції включав завантаження пакета на сервер обробки подій, встановлення залежностей, розгортання MQTT-брокера Mosquitto, імпорт правил і сценаріїв, а також ініціалізацію підсистеми журналювання. Після запуску система автоматично під'єднувала сенсорний вузол ESP32 через Wi-Fi/Ethernet, виконувала реєстрацію топіків sensor/+ /event та control/+ /cmd, створювала каталоги логів і генерувала перший базовий звіт із перевіркою працездатності. Проведене розгортання підтвердило відтворюваність конфігурацій, стабільність ініціалізації та коректність роботи інсталяційного пакета в різних середовищах, що засвідчує готовність системи до серійних тестових запусків та інтеграції із зовнішніми керованими системами.

4.6 Висновки до четвертого розділу

У четвертому розділі було виконано всебічне тестування інтелектуальної системи автоматизації опалення з прогнозуванням температурних умов, що дало змогу оцінити її продуктивність, стабільність і надійність у цільових експлуатаційних сценаріях. Аналіз отриманих результатів продемонстрував, що середня затримка наскрізної обробки подій не перевищує проєктного обмеження та утримується на рівні нижче 50 мс для основних потоків даних, що підтверджено побудованими графіками та метриками відповідності. У процесі

навантажувального тестування зафіксовано стабільну роботу серверного модуля, відсутність критичних помилок черги та відновлення після пікових коливань інтенсивності подій.

Оцінювання рівня успішності обробки подій показало, що система досягає показника понад 99 %, що свідчить про коректність алгоритмів маршрутизації, фільтрації та обробки сенсорних даних у режимі реального часу. Проведений аналіз логів та метрик підтвердив відповідність системи нефункціональним вимогам щодо цілісності даних, пропускної здатності й температурної стабільності серверного вузла. Додатково перевірено ефективність інтеграції з MQTT-інфраструктурою та стійкість до короткочасних збоїв у транспортному каналі.

За результатами тестування встановлено, що інсталяційний пакет системи забезпечує коректне розгортання в середовищах Linux/Windows, автоматичну ініціалізацію підсистем, сумісність модулів та відтворюваність тестових сценаріїв, що засвідчує готовність розробленого рішення до інтеграції в реальні умови експлуатації та подальшої масштабованості. Отримані результати підтверджують відповідність системи функціональним, технічним і якісним характеристикам, визначеним на етапі проєктування.

ВИСНОВКИ

У кваліфікаційній роботі здійснено повний цикл дослідження, проєктування та реалізації інтелектуальної системи емуляції сенсорних подій, що інтегрується з MQTT-інфраструктурою та керованими підсистемами. На основі системного аналізу предметної області було сформовано вимоги до архітектури, протоколів взаємодії, продуктивності та надійності, що дало змогу визначити ключові обмеження, інформаційні потоки та функціональні ролі компонентів майбутньої системи. Проведене моделювання охоплювало побудову UML-діаграм класів, компонентів, кооперацій, розгортання та структурної організації пакетів, що забезпечило формалізацію взаємодії між сенсорними вузлами, брокером MQTT, сервером обробки подій та операторською консоллю.

На етапі розроблення було реалізовано апаратно орієнтований емулятор, який відтворює поведінку сенсорного модуля ESP32, включно з симуляцією GPIO-каналів, потоків подій, параметрів навантаження та профілів інтенсивності. Сервер обробки подій створено на основі Python 3.x з використанням асинхронної черги, механізмів фільтрації та кореляції подій, модуля правил RuleEngine та підсистеми реагування, що генерує командні топіки control/+ /cmd. Розроблена система журналювання забезпечує збір метрик, структуровані логи, вимірювання p95 затримок та формування звітів у форматах CSV/JSON/HTML. Інсталяційний пакет адаптовано для середовищ Linux/Windows і включає всі необхідні конфігураційні та програмні модулі.

У результаті тестування отримано підтвердження працездатності та надійності системи за різних режимів роботи. Середня наскрізна затримка обробки подій не перевищила проєктного порогу, а коефіцієнт успішності маршрутизації досяг 99 %. Система продемонструвала стійкість до пікових навантажень, збоїв транспортного рівня, спроб повторної доставки подій і

порушення цілісності MQTT-потоків. Перевірка безпеки засвідчила коректність роботи TLS 1.3, ACL-політик, механізмів аутентичності та відмовостійкості інфраструктури.

Отримані результати підтверджують, що розроблена система відповідає функціональним, технічним та експлуатаційним вимогам, встановленим на етапі формування постановки задачі. Розроблений інтелектуальний емулятор може використовуватися як у дослідницьких цілях, так і для тестування керованих систем доступу, моніторингу клімату, сигналізації та інших кіберфізичних платформ. Проведене дослідження та реалізоване програмно-технічне рішення мають практичну цінність і можуть бути розширені шляхом інтеграції машинного навчання, розширення набору сенсорних сценаріїв, а також масштабування на кластерні MQTT-інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC 30141:2018. Internet of Things (IoT) — Reference Architecture. Geneva: ISO/IEC, 2018. (дата звернення: 11.09.2025).

2. OASIS. MQTT Version 5.0. OASIS Standard. Burlington: OASIS, 2019. (дата звернення: 11.09.2025).

3. ISO 16484-5:2017. Building automation and control systems (BACS). Part 5: Data communication protocol. Geneva: ISO, 2017. (дата звернення: 11.09.2025).

4. Home Assistant. Developer Documentation. Home Assistant Project, 2025. (дата звернення: 11.09.2025).

5. openHAB. Documentation. openHAB Foundation, 2025. (дата звернення: 11.09.2025).

6. Eclipse Mosquitto. MQTT Broker — Project Documentation. Eclipse Foundation, 2025. (дата звернення: 11.09.2025).

7. ISO/IEC 25010:2011. Systems and software engineering — System and software quality models. Geneva: ISO/IEC, 2011.

8. Filament, R., O'Neil, P. *Event-Driven Architectures for IoT Systems*. ACM Computing Surveys, 2021, vol. 54, no. 7, pp. 1–32.

9. Paho MQTT Client Library. Eclipse Foundation, 2023. URL: https://www.eclipse.org/paho (дата звернення: 20.11.2025).

10. Stallings, W. *Network Security Essentials: Applications and Standards*. 7th ed. Pearson, 2023. 580 p.

11. Google. *Protocol Buffers: Developer Guide*. 2024. URL: https://developers.google.com/protocol-buffers (дата звернення: 20.11.2025).

12. Kumar, A., Singh, P. *Testing IoT Systems: Methods, Metrics and Challenges*. IEEE Internet of Things Journal, 2022, vol. 9, no. 15, pp. 14723–14739.

13. Marwedel, P. *Embedded System Design*. 3rd ed. Springer, 2021. 423 p.

14. Sato, M., Yamada, T. *Sensor Event Processing in Distributed IoT Systems*. Sensors, 2023, vol. 23, no. 3, pp. 1–18.

15. Python Software Foundation. *Python 3.12 Documentation*. 2024. URL: https://docs.python.org/3 (дата звернення: 20.11.2025).

16. Red Hat. *SELinux User Guide*. 2023. URL: https://access.redhat.com/documentation/en-us (дата звернення: 20.11.2025).

17. Google Cloud. *Operational Logging and Metrics in Distributed Systems*. 2022. URL: https://cloud.google.com/monitoring (дата звернення: 20.11.2025).

18. Dugan, J., et al. *The Art of Monitoring*. James Turnbull, 2017. 384

19. Oetiker, T. *RRDtool - Round Robin Database Tool Documentation*. 2024. URL: https://oss.oetiker.ch/rrdtool (дата звернення: 20.11.2025).