Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 2302136.23.02.42. ПЗ

Виконав здобувач III курсу, група КІ2с-23-2 _____ Олександр ОЛІЙНИК
Підпис Ініціали, прізвище

Керівник _____ Ольга АТАМАНЮК
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«12» червня 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Олійнику Олександру Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії
Керівник проекту (роботи) Атаманюк Ольга Вадимівна, асистент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 5
2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Завдання на кваліфікаційну роботу
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Програмно-апаратний комплекс навчального стенду з Іот для підготовки фахівців з комп'ютерної інженерії та постановка щодо розробки
Проєктування системи обробки інформації у програмно-апаратному комплексі навчального стенду
Програмно-апаратна реалізація компонентів навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Апаратна архітектура навчального стенду IoT
Програмне забезпечення та серверна інфраструктура
Алгоритмічне забезпечення та верифікація системи

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – аналіз предметної області та технологій побудовинавчальних стендів з інтернету речей	01.03.2026	виконано
4	Робота над розділом 2 – основи проєктування та системного аналізу апаратно-програмних комплексів інтернету речей	01.04.2026	виконано
5	Робота над розділом 3 – програмно-апаратна реалізація кіберфізичної системи навчального стенду з інтернету речей	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ ___Олександр ОЛІЙНИК___
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ ___Ольга АТАМАНЮК___
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 2302136.23.02.42 ПЗ	Пояснювальна записка	64		
			Графічні матеріали			
2		КвРКІ 2302136.23.02.42 Е8	Апаратна архітектура	1		
			навчального стенду IoT			
3		КвРКІ 2302136.23.02.42 Е8	Програмне забезпечення т	1		
			а серверна			
			інфраструктура			
4		КвРКІ 2302136.23.02.42 Е8	Алгоритмічне	1		
			забезпечення			
			та верифікація системи			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 2302136.23.02.42 ВП		
Розробив		Олійник		02.06	Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Атаманюк		02.06		У	1	1
						ХНУ, КІ2с-23-2		
Н. контр.		Лисенко		02.06				
Затв.		Павлова		02.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії».

Автор роботи: Олександр ОЛІЙНИК.

Керівник роботи: Ольга АТАМАНЮК.

Пояснювальна записка: 64 с., 20 рис., 4 дод., 50 джерел.

Графічна частина: 3 креслення.

БАЗА ДАНИХ, ІНТЕРНЕТ РЕЧЕЙ, КІБЕРФІЗИЧНА СИСТЕМА, МІЖМАШИННА ВЗАЄМОДІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню програмно-апаратного комплексу навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії. Актуальність теми зумовлена стрімким впровадженням технологій IoT та гострою необхідністю модернізації освітнього процесу. Використання сучасних кіберфізичних навчальних макетів дозволяє студентам на практиці досліджувати повний життєвий цикл інформації: від збору первинної телеметрії периферійними датчиками до її інтелектуальної обробки, збереження та візуалізації на високому серверному рівні.

Метою роботи є проєктування, практична реалізація та експериментальне тестування апаратно-програмного комплексу для демонстрації принципів функціонування розподілених систем керування у реальному часі. Для цього було виконано системний аналіз архітектурних рівнів взаємодії, розроблено логіко-фізичну модель реляційної бази даних, спроєктовано асинхронне серверне програмне забезпечення мовою Python із використанням математичного гістерезису та створено інтерактивний користувацький веб-інтерфейс інженерного пульта. Проведені практичні випробування повністю підтвердили високу надійність та дидактичну ефективність розробленого рішення.

30.05.2026

Підпис здобувача

Дата

ЗМІСТ

					КвРКІ. 2302136.23.02.42 ПЗ		
Зм.	Арк.	№докум.	Підпис	Дата			
Виконав		Олександр олійник		02.06	Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії. Пояснювальна записка	Літера: у	Аркуш: 1
Перевір.		Ольга АТАМАНЮК		02.06			Аркушів: 64
Н.контр.		Сергій ЛИСЕНКО		02.06		ХНУ КІ2с-23-2	
Затвер.		Ольга ПАВЛОВА		02.06			

ВСТУП

Актуальність дослідження. Стрімкий розвиток та глобальне впровадження технологій Інтернету речей у всі сфери сучасного суспільства, починаючи від систем розумного будинку і закінчуючи складною промисловою автоматизацією, формує надзвичайно гостру потребу у висококваліфікованих інженерах. Класичні підходи до викладання дисциплін комп'ютерної інженерії, що базуються переважно на абстрактному теоретичному матеріалі. Такі комплекси мають наочно демонструвати повний життєвий цикл інформації: від збору первинної телеметрії датчиками до передачі даних мережею та їх високорівневої інтелектуальної обробки за допомогою сучасних мов програмування, зокрема Python.

Метою кваліфікаційної роботи є теоретичне проєктування та практична реалізація апаратно-програмного комплексу навчального стенду з Інтернету речей, який органічно об’єднує периферійні датчики, мікроконтролери та централізоване серверне програмне забезпечення на мові Python для ефективної демонстрації базових принципів побудови IoT-рішень під час професійної підготовки фахівців з комп’ютерної інженерії.

Об’єктом дослідження виступають процеси безперервного збору, мережевої передачі та програмної оркестрації інформації у розподілених кіберфізичних системах та сучасних середовищах Інтернету речей.

Предметом дослідження є інженерні архітектурні рішення, апаратні мікропроцесорні компоненти та алгоритми серверного програмного забезпечення на базі фреймворку Flask, що безпосередньо забезпечують автономну міжмашинну взаємодію, логування подій та графічну візуалізацію телеметрії у межах розробленого навчально-демонстраційного комплексу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ ПОБУДОВИНАВЧАЛЬНИХ СТЕНДІВ З ІНТЕРНЕТУ РЕЧЕЙ

1.1 Еволюція концепції Інтернету речей та необхідність практичної підготовки фахівців

Стрімкий розвиток інформаційно-комунікаційних технологій у XXI столітті призвів до формування нової глобальної парадигми -Інтернету речей. Ця концепція передбачає створення глобальної обчислювальної мережі фізичних предметів «речей», оснащених вбудованими технологіями для взаємодії один з одним або із зовнішнім середовищем. Впровадження IoT-систем докорінно змінює принципи автоматизації в таких сферах, як розумні будинки, промисловість 4.0, розумні міста, сільське господарство та медицина. Відповідно, на ринку праці виникла гостра потреба у кваліфікованих фахівцях з комп'ютерної інженерії, здатних проєктувати, налаштовувати та підтримувати такі комплексні системи.

Історичний розвиток концепції IoT бере свій початок з 1999 року, коли Кевін Ештон вперше використав цей термін, запропонувавши систему управління ланцюгами постачання на базі радіочастотної ідентифікації RFID. Наступним значним кроком став перехід до міжмашинної взаємодії M2M, що дозволило пристроям обмінюватися телеметрією без участі людини. Сучасний етап еволюції IoT характеризується синергією вбудованих систем, хмарних обчислень, алгоритмів штучного інтелекту та високорівневих мов програмування, таких як Python, що використовуються для розробки серверної частини та обробки великих даних.

Незважаючи на високу затребуваність IoT-інженерів, існує значний розрив між теоретичною підготовкою студентів у закладах вищої освіти та реальними вимогами індустрії. Традиційні підходи до вивчення комп'ютерних мереж та мікроконтролерів часто обмежуються використанням чисто програмних симуляторів, наприклад Cisco Packet Tracer, Proteus, Tinkercad. Хоча такі

інструменти дозволяють вивчити базові принципи маршрутизації та логіки роботи мікропроцесорів, вони не здатні повною мірою відтворити фізичні аспекти функціонування кіберфізичних систем. У віртуальному середовищі відсутні реальні проблеми, з якими стикається інженер: перешкоди в каналах зв'язку, брязкання контактів, втрата пакетів у Wi-Fi мережах, проблеми з живленням або необхідність калібрування аналогових датчиків.

Розглянемо еволюцію методів підготовки фахівців з комп'ютерної інженерії у контексті вивчення мережевих та вбудованих систем на рисунку 1.1.

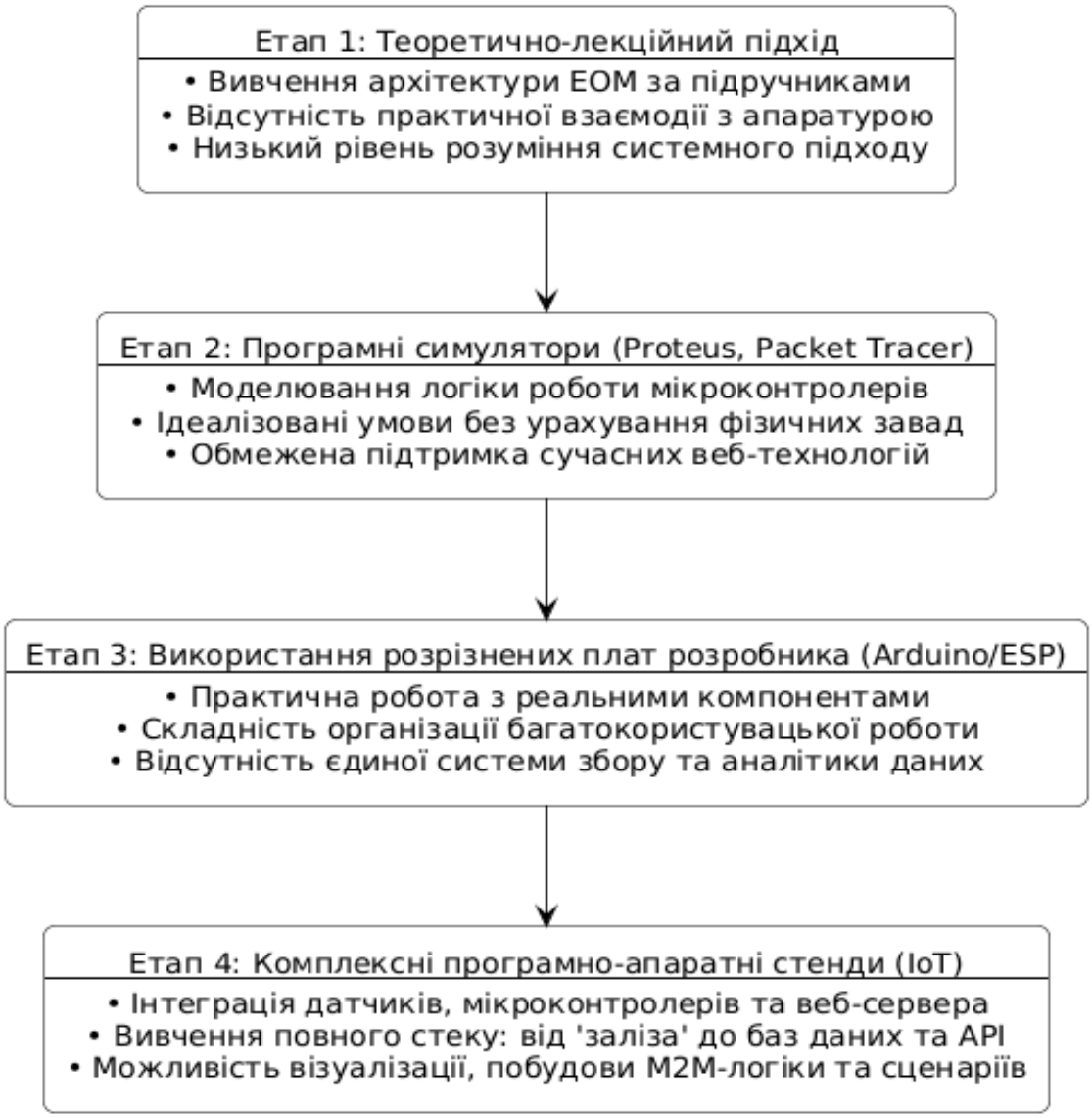


Рисунок 1.1 - Еволюція підходів до вивчення вбудованих та IoT систем

Саме тому критично важливою є розробка та впровадження в освітній процес фізичних програмно-апаратних навчальних стендів. Навчальний стенд - це комплексне рішення, яке поєднує реальне апаратне забезпечення (мікроконтролери, сенсори, актуатори) та сучасне програмне забезпечення серверні застосунки, бази даних, графічні інтерфейси користувача. Використання стендів дозволяє застосувати проєктно-орієнтований підхід до навчання.

Згідно рисунка 1.1, перехід до четвертого етапу використання комплексних стендів є логічною вимогою сучасної освіти. Навчальний стенд з Інтернету речей повинен демонструвати студентам не лише те, як отримати значення температури чи стану дверей, але й те, як ці дані перетворюються у JSON-формат, передаються через мережу, зберігаються у реляційній базі даних наприклад, SQLite і використовуються для прийняття автоматизованих рішень за заздалегідь визначеними правилами M2M логіка.

Крім того, сучасні навчальні стенди мають будуватися на базі актуальних технологічних стеків. Використання мови програмування Python для розробки серверної частини, фреймворки Flask або FastAPI є обґрунтованим вибором, оскільки Python є стандартом у галузях IoT, Data Science та кібербезпеки. Це дозволяє студентам вивчати принципи RESTful API, розробляти алгоритми обробки телеметричних даних та досліджувати вразливості систем.

Отже, створення спеціалізованого програмно-апаратного комплексу навчального стенду з Інтернету речей є практичною задачею. Її вирішення дозволить підвищити якість підготовки фахівців з комп’ютерної інженерії, сформувавши у них комплексне розуміння архітектури кіберфізичних систем на всіх рівнях моделі OSI: від фізичного до прикладного. Навчальний стенд є багатокомпонентною інженерною системою, що інтегрує фізичні апаратні модулі, до складу яких входять мікроконтролерні вузли, датчики телеметрії та виконавчі пристрої, з комплексом програмних засобів, охоплюючи серверні

платформи, сховища даних та клієнтські графічні інтерфейси для моніторингу та керування процесами.

1.2 Аналіз існуючих апаратних платформ та мікроконтролерів для освітніх цілей

Проєктування та реалізація програмно-апаратного комплексу навчального стенду з Інтернету речей вимагає обґрунтованого вибору елементної бази. Апаратна платформа кінцевих вузлів повинна забезпечувати надійне зчитування телеметрії з датчиків, первинну обробку сигналів та передачу інформації до центрального сервера через мережеві інтерфейси. Для підготовки фахівців з комп'ютерної інженерії вибір архітектури мікроконтролера має не лише практичне, а й глибоке методичне значення: студенти повинні вивчати як низькорівневі аспекти обробки даних, робота з регістрами, перериваннями, таймерами, аналогово-цифровими перетворювачами, так і високорівневі протоколи взаємодії.

У сучасній інженерній та освітній практиці найбільшого поширення набули три основні класи апаратних рішень:

1. 8-бітні мікроконтролери (архітектура AVR, платформи типу Arduino Uno/Nano).
2. 32-бітні мікроконтролери з інтегрованими бездротовими інтерфейсами (архітектура Tensilica Xtensa, платформи сімейства ESP8266/ESP32).
3. Одноплатні комп'ютери (архітектура ARM, платформи типу Raspberry Pi / Orange Pi).

Розглянемо архітектурні особливості, переваги та обмеження кожної платформи у контексті розробки навчального стенду.

Платформи на базі 8-бітних мікроконтролерів ATmega328P Arduino Uno тривалий час були стандартом для навчання завдяки простоті архітектури та низькій вартості. Вони дозволяють детально вивчити базові засади

мікропроцесорної техніки, пряме керування GPIO-виводами, конфігурацію апаратних інтерфейсів UART, SPI та I2C на рівні регірів. Проте для сучасних завдань Інтернету речей ці мікроконтролери мають критичні обмеження. Обсяг оперативної пам'яті становить лише 2 Кбайт, а флеш-пам'яті -32 Кбайт, що унеможливлює розгортання сучасних стеків мережевих протоколів, наприклад TLS для шифрування даних. Головним недоліком є відсутність вбудованих мережевих інтерфейсів. Для підключення до мережі необхідне використання зовнішніх модулів Ethernet-shield W5500 або Wi-Fi модуль ESP01, що ускладнює апаратну схему стенду, знижує її надійність та підвищує вартість кінцевого виробу.

Одноплатні комп'ютери, наприклад Raspberry Pi представляють протилежний полюс обчислювальної потужності. Вони оснащені багатоядерними процесорами архітектури ARM, мають гігабайти оперативної пам'яті та працюють під керуванням повноцінних операційних систем сімейства Linux. Це дозволяє програмувати безпосередньо на платі мовою Python, підключати бази даних та розгортати веб-сервери. Однак використання одноплатного комп'ютера як кінцевого сенсорного вузла є економічно та інженерно необґрунтованим.

Вони мають високе енергоспоживання, тривалий час завантаження ОС, високу вартість і ризик пошкодження файлової системи на карті пам'яті при раптовому вимкненні живлення, що часто трапляється в навчальних лабораторіях. Крім того, робота з периферією через шари абстракції ОС Linux віддаляє студентів від розуміння реального часу та апаратної взаємодії на низькому рівні.

З огляду на зазначені недоліки крайніх архітектур, оптимальним вибором для реалізації кінцевих вузлів навчального стенду виступають 32-бітні мікроконтролери, зокрема сімейства ESP32. Для побудови повноцінної кіберфізичної системи їх необхідно інтегрувати з широким спектром

периферійних пристроїв. Типовий набір електронних компонентів та сенсорів, необхідних для практичної підготовки фахівців (рис 1.2).

Інфраструктура стенду має охоплювати як базову схемотехніку, резистори, кнопки, потенціометри, так і складні цифрові модулі, ультразвукові далекоміри, RFID-зчитувачі, OLED-дисплеї, сервоприводи. Сучасні 32-бітні платформи ідеально підходять для обслуговування такої розмаїтої елементної бази, оскільки мають достатню кількість GPIO-виводів, апаратну підтримку різноманітних інтерфейсів та інтегровані модулі зв'язку для обробки й передачі зібраної телеметрії.

Рисунок 1.2 - Типовий набір датчиків, актуаторів та електронних компонентів для навчального стенду з Інтернету речей [48]

32-бітні мікроконтролери сімейства ESP32 (архітектура Xtensa LX6/LX7) на сьогодні є найбільш збалансованим та технологічним рішенням для індустрії та освіти в галузі комп'ютерної інженерії. Модуль ESP32 поєднує в собі низьку вартість класичних мікроконтролерів та обчислювальні можливості, достатні для побудови складних IoT-пристроїв.

Основні технічні характеристики архітектури ESP32 включають:

1. Двоядерний 32-бітний процесор із частотою до 240 МГц.
2. Обсяг вбудованої оперативної пам'яті SRAM до 520 Кбайт.
3. Інтегровані апаратні модулі Wi-Fi та Bluetooth.
4. Багатий набір периферійних інтерфейсів: 12-бітні АЦП, 8-бітні ЦАП, апаратні модулі ШІМ, сенсорні контакти, а також інтерфейси SPI, I2C, UART та CAN-bus.

Великий обсяг пам'яті та висока тактова частота ESP32 дозволяють реалізовувати криптографічні алгоритми захисту інформації, підтримувати захищені з'єднання HTTPS та роботу з сучасними IoT-протоколами. З методичної точки зору, ESP32 є унікальним інструментом, оскільки підтримує програмування у середовищі Arduino IDE, у професійному середовищі ESP-IDF на мові C/C++ з використанням операційної системи реального часу FreeRTOS, а також підтримує виконання MicroPython.

Для порівняльного аналізу наведено рисунок 1.3, систематизації критеріїв вибору апаратної платформи розроблено структурну схему класифікації апаратних засобів IoT-вузлів, що відображає архітектурний розподіл обчислювальних ресурсів та інтерфейсів зв'язку.

Зважаючи на проведені дослідження, для створення навчального стенду з Інтернету речей найбільш доцільно обрати архітектуру на основі 32-бітних мікроконтролерів з вбудованими бездротовими інтерфейсами. Це забезпечить необхідну гнучкість при розгортанні лабораторних робіт: від базового зчитування показників з датчиків до побудови складних розподілених бездротових мереж із захистом даних.

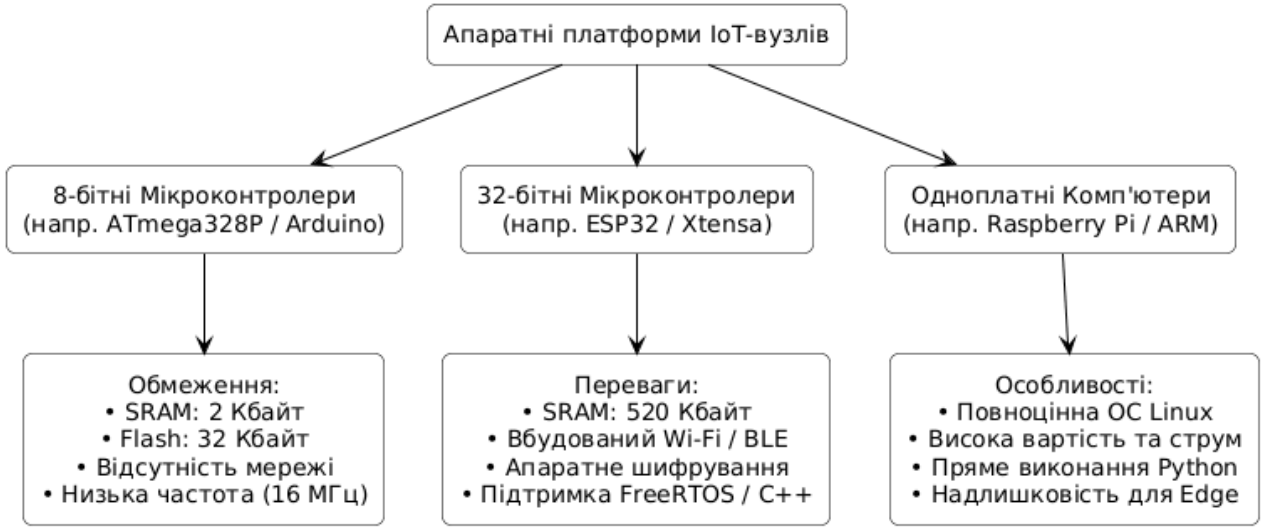


Рисунок 1.3 - Структурно-функціональний аналіз апаратних платформ для Інтернету речей

При цьому центральний сервер стенду, який збирає інформацію, реалізується на базі ПК або локального сервера мовою Python, що дозволяє продемонструвати студентам класичну клієнт-серверну архітектуру, яка використовується у промислових індустріальних IoT-рішеннях.

1.3 Огляд програмних архітектур, IoT-протоколів та веб-технологій

Проєктування програмної складової навчального стенду з Інтернету речей вимагає комплексного підходу до вибору архітектури, мережевих протоколів та інструментів розробки. У класичній моделі кіберфізичних систем програмне забезпечення поділяється на три логічні рівні: рівень збору даних, рівень обробки та збереження, а також рівень представлення інформації користувачеві. В освітньому процесі важливо, щоб кожен із цих рівнів був реалізований за допомогою прозорих, актуальних та індустріально визнаних технологій, що дозволить студентам безперешкодно засвоювати принципи побудови розподілених інформаційних систем.

Фундаментом взаємодії між апаратною та програмною частинами є мережеві протоколи прикладного рівня. Історично в IoT-системах

використовується кілька основних підходів до передачі телеметрії. Протокол HTTP, побудований на архітектурі «клієнт-сервер», є найпоширенішим інструментом для організації RESTful API. Хоча класичний HTTP передбачає встановлення нового з'єднання для кожного запиту, що збільшує накладні витрати трафіку, його використання в навчальних цілях є надзвичайно корисним для розуміння базових принципів роботи веб-серверів.

Альтернативним підходом є використання протоколу MQTT, який працює за моделлю «видавець-підписник» і оптимізований для пристроїв з низьким енергоспоживанням та нестабільним зв'язком. Ще однією важливою технологією є WebSockets, що забезпечує повнодуплексний зв'язок у реальному часі. Для базового навчального стенду реалізація взаємодії через HTTP POST та GET запити є оптимальною, оскільки вона легко інспектується стандартними інструментами розробника у браузері і дозволяє студентам наочно бачити структуру пакетів у форматі JSON.

Для розробки серверної частини навчального стенду найдоцільніше використовувати мову програмування Python. Цей вибір обґрунтований її лаконічним синтаксисом, величезною екосистемою бібліотек та тотальним домінуванням у сферах аналізу даних та машинного навчання, які тісно пов'язані з Інтернетом речей. Серед розмаїття веб-фреймворків Python особливої уваги заслуговує мікрофреймворк Flask. На відміну від важковагових рішень, таких як Django, Flask не нав'язує жорсткої структури проєкту та складних абстракцій баз даних, що робить його ідеальним інструментом для швидкого прототипування IoT-додатків. Використання Flask дозволяє буквально кількома рядками коду створити повноцінний API-інтерфейс для прийому даних від мікроконтролерів та управління виконавчими пристроями.

Збереження історичних даних телеметрії є невід'ємною частиною будь-якої системи моніторингу. Для навчального програмно-апаратного комплексу доцільно відмовитися від розгортання складних систем керування базами даних, таких як PostgreSQL або MySQL, на користь вбудованої реляційної бази даних

SQLite. Ця технологія зберігає всю базу даних у вигляді єдиного локального файлу, не вимагає налаштування окремого серверного процесу та підтримує стандартний синтаксис мови SQL. Це значно спрощує процес розгортання стенду на комп'ютерах студентів та дозволяє легко експортувати лог-файли для подальшого математичного аналізу або побудови графіків.

Рівень представлення даних відіграє ключову роль у сприйнятті кіберфізичної системи користувачем, формуючи так званий людино-машинний інтерфейс. Сучасні панелі управління будуються з використанням зв'язки технологій HTML5, CSS3 та JavaScript. Динамічне оновлення даних на веб-сторінці без її перезавантаження реалізується за допомогою технології асинхронних запитів AJAX. Для візуалізації аналітичних трендів та відображення перетину критичних меж застосовуються спеціалізовані JavaScript-бібліотеки, такі як Chart.js, що дозволяють генерувати інтерактивні графіки безпосередньо у браузері клієнта. Окремою вимогою до людино-машинного інтерфейсу інженерних систем є впровадження систем контролю доступу, механізмів підтвердження дій та ведення системного журналу подій для фіксації моментів спрацьовування правил міжмашинної взаємодії.

Застосування протоколу HTTP з методами POST та GET для організації взаємодії є раціональним вибором для навчальної інфраструктури. Така архітектура забезпечує високу прозорість обміну даними, оскільки дозволяє студентам проводити моніторинг інформаційних пакетів за допомогою вбудованих засобів налагодження веббраузера, що сприяє глибокому засвоєнню структури даних у форматі JSON.

Синтез наведених програмних технологій формує цілісну екосистему, яка повністю відповідає сучасним стандартам розробки промислових інформаційних систем. Такий технологічний стек забезпечує мінімальний поріг входження для студентів, зберігаючи при цьому безмежні можливості для масштабування проєкту: від додавання складних алгоритмів предиктивної аналітики на Python до інтеграції з хмарними сервісами обробки великих даних.

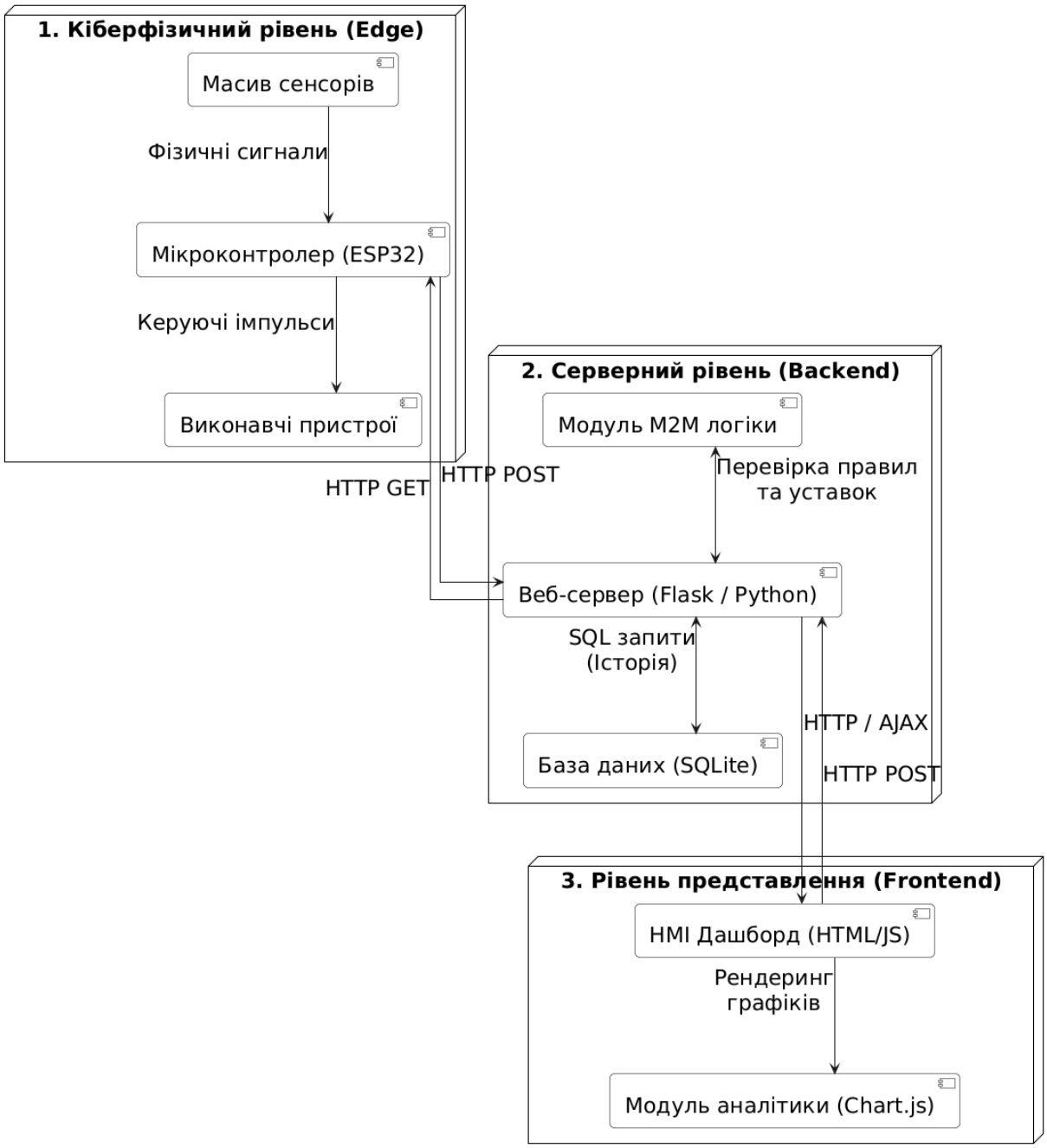


Рисунок 1.4 - Архітектура програмного забезпечення навчального IoT-стенду

Для наочного представлення взаємозв'язків між описаними програмними компонентами розроблено діаграму архітектури програмного забезпечення комплексу (рис 1.4). Це рішення полегшує діагностику системи, адже студенти отримують можливість інспектувати структуру переданого JSON-контенту безпосередньо в консолі браузера, що підвищує ефективність навчання

1.4 Постановка задачі на розробку навчального програмно-апаратного комплексу

Аналіз сучасного стану розвитку кіберфізичних систем та підходів до підготовки інженерних кадрів дозволяє чітко сформулювати існуючу науково-практичну проблематику. Основна проблема полягає у фрагментованості освітнього процесу, де вивчення апаратної складової, мережевих протоколів та високорівневого програмування часто відбувається ізольовано один від одного. У реальних промислових умовах фахівець з комп'ютерної інженерії стикається з необхідністю комплексного проєктування систем, що охоплюють увесь стек технологій від збору фізичних сигналів до їх візуалізації та аналітики на центральному сервері. Відсутність єдиної апаратно-програмної платформи, яка б наочно демонструвала безперервний потік даних, ускладнює формування у студентів цілісного системного мислення та розуміння архітектури Інтернету речей.

Виходячи з наявної проблематики, головною метою даної кваліфікаційної роботи є проєктування, розробка та впровадження комплексного навчального програмно-апаратного стенду. Для забезпечення високого рівня залученості та наочності освітнього процесу комплекс має імітувати реальні індустріальні процеси, зокрема функціонування сегмента розумної будівлі або автоматизованої серверної інфраструктури. Такий контекстний підхід дозволить студентам не лише вивчати абстрактні алгоритми збору телеметрії, але й вирішувати конкретні інженерні завдання, такі як контроль мікроклімату, моніторинг безпеки приміщення та автоматичне керування системами вентиляції, освітлення чи оповіщення.

Для структурування процесу розробки було проведено декомпозицію поставленої мети на три основні функціональні рівні, які відображають ієрархію побудови кіберфізичної системи (рис. 1.5).

Рисунок 1.5 Структурна декомпозиція завдань розробки навчального стенду

Аналіз наведеної схеми декомпозиції дозволяє деталізувати зміст інженерних завдань, що підлягають вирішенню у межах цієї роботи.

На апаратному рівні завдання полягає у проєктуванні архітектури підключення різнорідних сенсорних модулів, які генерують широкий спектр сигналів: від аналогових показників температури до дискретних сигналів датчиків руху та відкриття дверей. Важливим аспектом є інтеграція актуаторів, що забезпечують зворотний зв’язок. Навчальний стенд повинен не просто збирати дані, а й дозволяти студентам досліджувати процеси взаємодії з апаратними інтерфейсами у реальному часі, що є критично важливим для розуміння принципів роботи вбудованих систем.

Практична реалізація поставлених завдань вимагає створення фізичного прототипу. Приклад комплексного втілення такого навчального стенду наведено на рисунку 1.6.

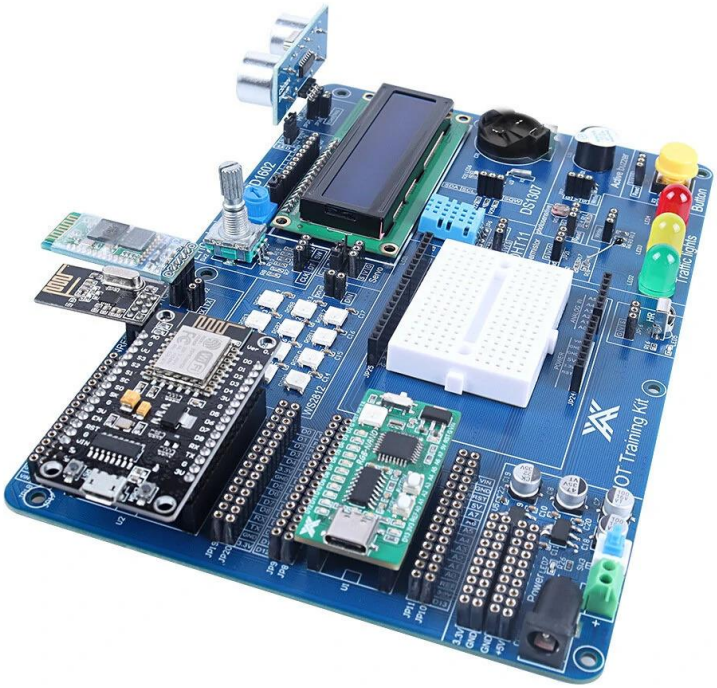


Рисунок 1.6 - Загальний вигляд апаратного комплексу для вивчення[48]

Основою представленої системи слугує потужний 32-бітний мікроконтролерний модуль, наприклад, ESP32, який інтегрує обчислювальні ресурси та бездротові інтерфейси зв’язку, необхідні для роботи кінцевого вузла. Апаратна база об'єднує на одній платформі різнорідні компоненти, що дозволяє студентам імітувати повноцінний моніторинг середовища. Серед них: датчики температури та вологості (DHT11), ультразвукові далекоміри, сенсори освітленості, а також модулі для виявлення протікання води чи контролю вологості ґрунту, що є базою для створення сценаріїв контролю мікроклімату в розумній будівлі. Зворотний зв’язок та виконавчий вплив забезпечуються за допомогою актуаторів - рідкокристалічного дисплея для локального виводу телеметрії, світлодіодної індикації та п’єзовипромінювача. Вбудовані реле та

роз'єми для підключення сервоприводів чи електромагнітних замків дозволяють реалізувати повноцінну автоматизацію, наприклад, керування доступом або вентиляцією. Фізична конструкція стенду спроєктована за модульним принципом з ергономічним групуванням та чітким маркуванням компонентів на спільній монтажній базі. Це усуває проблему нестабільних контактів та безладу в дротах, дозволяючи студентам одразу переходити до вивчення програмних аспектів: конфігурування апаратних інтерфейсів та реалізації мережевих протоколів. Такий комплексний підхід перетворює абстрактні концепції IoT на відчутний інженерний об'єкт та забезпечує наочність повного циклу збору, обробки і передачі даних у кіберфізичній системі.

На програмному рівні Backend завдання спрямоване на розробку серверної платформи, здатної функціонувати як інтелектуальний диспетчер. Програмне забезпечення повинно забезпечувати стабільний прийом запитів, їх валідацію та потокову обробку. Особлива увага приділяється механізму міжмашинної взаємодії. Ця логіка є серцем системи: сервер має не лише пасивно зберігати дані в БД, але й активно порівнювати їх із пороговими уставками, ініціюючи автоматичні сценарії реакції. Це перетворює стенд із простого реєстратора даних на повноцінний контролер, що здатний забезпечувати автономне функціонування об'єкта, наприклад, при виникненні аварійних ситуацій.

Рівень інтерфейсу Frontend має бути спроєктований з урахуванням потреб як студента, що вивчає систему, так і інженера, що її налаштовує. Веб-дашборд повинен забезпечувати наочну візуалізацію даних, перетворюючи сирий потік чисел на зрозумілі графіки та статуси. Водночас він мусить слугувати інструментом для конфігурації: студенти повинні мати можливість задавати умови автоматизації, не втручаючись у код сервера, що навчає їх принципам побудови систем з віддаленим управлінням.

Таким чином, комплексна постановка задачі охоплює повний цикл проєктування IoT-системи: від створення фізичних вузлів до реалізації інтелектуального програмного забезпечення та зручного інтерфейсу

користувача. Такий підхід забезпечує глибоке розуміння того, як окремі апаратні та програмні компоненти об'єднуються у єдину, функціонально завершену кіберфізичну систему, здатну до адаптивної роботи та моніторингу параметрів середовища. Вирішення цих завдань у сукупності дозволить створити інструментарій, який відповідає високим вимогам сучасної інженерної освіти та сприяє отриманню студентами практичних навичок, необхідних для розробки перспективних ІТ-продуктів.

1.5 Висновки до першого розділу

У першому розділі кваліфікаційної роботи було проведено ґрунтовний аналіз стану, методів та засобів побудови навчальних стендів з Інтернету речей. Дослідження історичної еволюції кіберфізичних систем показало перехід від локального апаратного контролю до хмарних платформ та інтелектуальних систем адаптивного керування. З'ясовано, що для якісної практичної підготовки інженерів життєво необхідним є впровадження апаратно-програмних комплексів, які дозволяють досліджувати повний цикл обігу інформації.

Аналіз апаратних платформ довів, що найоптимальнішим рішенням для побудови кінцевих вузлів є використання 32-бітних мікроконтролерів із вбудованими мережевими інтерфейсами, які забезпечують ідеальний баланс між продуктивністю та економічною доцільністю. Огляд програмних архітектур підтвердив раціональність використання мови програмування Python, мікрофреймворку Flask та вбудованої бази даних SQLite для реалізації серверної частини стенду.

В результаті проведених досліджень було сформульовано чітку постановку задачі на розробку апаратно-програмного комплексу навчального стенду, визначено його архітектурні межі та ключові функціональні вимоги до програмного забезпечення та людино-машинного інтерфейсу.

2 ОСНОВИ ПРОЄКТУВАННЯ ТА СИСТЕМНОГО АНАЛІЗУ АПАРАТНО-ПРОГРАМНИХ КОМПЛЕКСІВ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Архітектурні рівні організації IoT-систем та теоретичні засади їх міжрівневої взаємодії

Проєктування сучасних навчальних стендів у галузі комп'ютерної інженерії вимагає глибокого теоретичного обґрунтування вибору архітектурних рішень. Інтернет речей як комплексна дисципліна об'єднує в собі декілька концептуальних просторів, починаючи від фізичної взаємодії з навколишнім середовищем за допомогою сенсорів і закінчуючи високорівневою аналітикою даних на базі сучасних мов програмування. Для побудови ефективного навчального посібника у вигляді фізичного стенду найдоцільнішим є використання класичної трирівневої архітектурної моделі, яка згодом розширюється до п'ятирівневої для деталізації мережевих процесів та обробки інформації. Орієнтація на таку структуру дозволяє студентам послідовно вивчати шлях проходження інформаційного сигналу та розуміти системні зв'язки між окремими компонентами комплексу.

Перший рівень, який традиційно називають фізичним або сенсорним, відповідає за безпосередній збір первинної телеметрії та взаємодію з виконавчими механізмами. У контексті навчального стенду цей рівень є критично важливим, оскільки саме тут закладаються фундаментальні знання про аналогово-цифрове перетворення, фільтрацію шумів та інтерфейси передачі даних на короткі відстані. Теоретичний аналіз цього рівня передбачає вивчення математичних моделей функціонування датчиків температури, вологості, газу чи освітленості. Кожен сенсор генерує електричний сигнал, який трансформується у цифровий код за допомогою вбудованих або зовнішніх аналого-цифрових перетворювачів мікроконтролера. На цьому ж рівні діють виконавчі пристрої, що демонструють зворотний зв'язку системи на зміну параметрів середовища, трансформуючи керуючі електричні сигнали у фізичну дію. Математичне

моделювання процесів на фізичному рівні спирається на теорему Котельникова-Шеннона, яка визначає мінімальну частоту дискретизації для точного відновлення аналогового сигналу без втрати інформаційного вмісту.

Другий рівень архітектури охоплює мережеву взаємодію та транспортування даних від кінцевих вузлів до центрального обчислювача або хмарної платформи. Тут розглядаються математичні та логічні моделі передачі інформації за допомогою різноманітних бездротових та дротових технологій. Для навчальних стендів особливу цінність мають енергоефективні бездротові протоколи локальної дії, які дозволяють продемонструвати студентам обмеження пропускної здатності, вплив завад на якість зв'язку та методи забезпечення цілісності пакетів даних. Важливим аспектом теоретичного аналізу мережевого рівня є оцінка накладної вартості протоколів, де порівнюються обсяги службових даних у пакетах різних прикладних рішень. Особлива увага приділяється дослідженню затримок при поширенні сигналів та мінімізації втрат пакетів в умовах щільного завантаження частотного діапазону.

Третій рівень координує обробку, збереження та візуалізацію інформації, що реалізується засобами високорівневого програмного забезпечення на базі мови Python. На цьому етапі дані, що надійшли з нижчих рівнів, піддаються структуризації, валідації та аналізу. Вибір Python як базового інструменту для цього рівня зумовен його інтерпретованою природою, лаконічністю та наявністю потужних бібліотек для роботи з мережевими сокетами, асинхронними потоками та базами даних. Студенти отримують можливість досліджувати алгоритми фільтрації часових рядів, розробляти логіку міжмашинної взаємодії за принципом реактивного керування та будувати графічні інтерфейси для оперативного моніторингу стану системи. Теоретичний аналіз обробки даних на цьому рівні також включає вивчення методів серіалізації інформації в уніфіковані текстові чи бінарні формати для оптимізації міжпроцесної взаємодії.

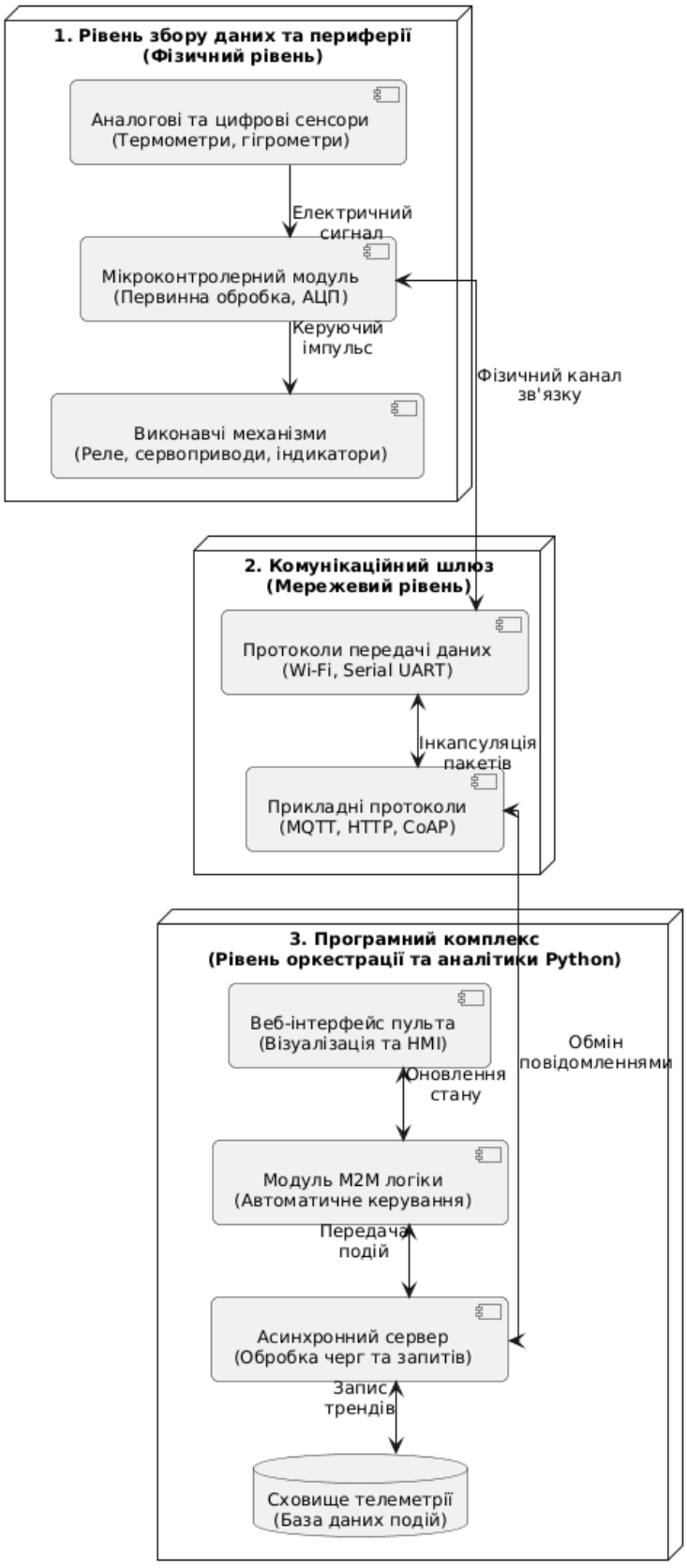


Рисунок 2.1 - Порівняльна структурна схема традиційної електромережі та мережі типу Smart Grid

Детальний аналіз наведеної структурно-функціональної схеми (рис 2.1), дозволяє простежити повний життєвий цикл інформації всередині кіберфізичного простору навчального стенду. Процес починається у крайній лівій точці фізичного рівня, де сенсори безперервно фіксують коливання параметрів навколишнього середовища. Оскільки більшість природних процесів є аналоговими за своєю природою, первинний електричний сигнал передається на вбудований аналого-цифровий перетворювач мікроконтролера. Математична точність цього перетворення безпосередньо залежить від розрядності матриці перетворювача, яка визначає крок квантування напруги. На цьому етапі виникає ефект шуму квантування, мінімізація якого розглядається в рамках теоретичного курсу комп'ютерної інженерії за допомогою програмних фільтрів ковзного середнього або цифрових низькочастотних фільтрів Баттерворта, реалізованих безпосередньо на мікроконтролері.

Сформований цифровий масив даних потребує пакування в логічні структури для подальшого транспортування. Мікроконтролерний модуль виконує функцію первинного обчислювального вузла, здійснюючи маркування даних часовими мітками або унікальними ідентифікаторами датчиків. Тут же реалізується зворотний зв'язок, коли цифрова команда від центрального сервера трансформується за допомогою широтно-імпульсної модуляції або дискретних логічних рівнів у механічний рух сервоприводів чи зміну стану комутаційних реле. Ця замкнена петля регулювання є класичним прикладом автоматизованого керування, що демонструє студентам практичне втілення теорії автоматичного регулювання в сучасних вбудованих системах.

Наступний етап взаємодії пов'язаний з подоланням межі між апаратними інтерфейсами мікроконтролера та глобальними мережевими середовищами. Комунікаційний шлюз виконує роль інтелектуального транслятора протоколів. Фізичний канал зв'язку, який може бути представлений послідовною шиною або радіочастотним каналом, забезпечує передачу потоку байтів до мережевого рівня. Теоретична складність цього процесу полягає в необхідності інкапсуляції

низькорівневих пакетів у високорівневі структури стеків TCP/IP або UDP. При вивченні цього блоку студенти аналізують вплив архітектури прикладних протоколів на загальну ефективність системи. Порівняльний аналіз накладних витрат показує, що текстові протоколи типу HTTP REST мають значно більшу збитковість трафіку через об'ємні текстові заголовки, тоді як бінарні або спеціалізовані легкомовні протоколи, наприклад MQTT, мінімізують обсяг службової інформації до кількох байтів, що робить їх ідеальними для обмежених за ресурсами пристроїв.

Завершальний етап обробки інформації відбувається на рівні оркестрації та аналітики, який повністю розгорнутий за допомогою інструментів мови Python. Асинхронна архітектура сервера дозволяє ефективно масштабувати систему та обробляти сотні паралельних запитів від комунікаційних шлюзів без блокування основного потоку виконання. Модуль міжмашинної взаємодії функціонує як експертна система, яка на основі детермінованих або стохастичних правил аналізує вхідний потік телеметрії. Якщо значення з датчика перевищує встановлений математичний поріг, логічний двигун миттєво ініціює подію зворотного керування. Одночасно з цим сервіс довготривалого збереження здійснює запис очищених і структурованих даних у реляційну або документоорієнтовану базу даних. Це дозволяє накопичувати історичні тренди для подальшого статистичного аналізу або навчання предиктивних моделей. Людино-машинний інтерфейс, реалізований у вигляді веб-пульта, динамічно оновлює свій стан за допомогою протоколу двонаправленого обміну повідомленнями у реальному часі, надаючи оператору повний контроль над фізичними процесами стенду та наочно демонструючи концепцію єдиного інформаційного простору кіберфізичної системи. Завдяки такій інтеграції, студент отримує можливість досліджувати закономірності поведінки об’єкта, виконувати віддалену діагностику вузлів та проводити детальну верифікацію реакцій автоматики на змінні зовнішні чинники, що суттєво поглиблює практичні навички проектування складних мережевих рішень.

2.2 Теоретичний аналіз та моделювання алгоритмів міжмашинної взаємодії M2M у локальних мережах

Основою інтелектуальності будь-якої сучасної кіберфізичної системи є її здатність до автономного функціонування без безпосереднього втручання людини-оператора. У контексті розроблюваного навчального стенду ця здатність реалізується через парадигму міжмашинної взаємодії, яка забезпечує прямий обмін інформацією та керуючими командами між різнорідними сенсорними вузлами та виконавчими механізмами на базі наперед заданих логічних правил. Теоретичне моделювання таких алгоритмів вимагає докорінного переходу від простого лінійного програмування та парадигми циклічного опитування, які традиційно вивчаються на початкових етапах підготовки інженерів, до сучасної подієво-орієнтованої архітектури. У такій архітектурній моделі кожна зміна фізичного параметра навколишнього середовища або надходження нового мережевого пакету розглядається як незалежна асинхронна подія, що миттєво ініціює відповідний каскад обчислювальних процесів у програмному ядрі системи. Використання мови програмування Python для реалізації цих алгоритмів дозволяє ефективно застосувати високорівневі абстракції, потужні структури даних та вбудовані механізми асинхронного вводу-виводу, що забезпечує одночасну та незалежну обробку десятків правил без ризику блокування основного потоку виконання програми.

Процес обробки правил міжмашинної взаємодії фундаментально базується на безперервному порівнянні вхідних емпіричних даних із заданими користувачем граничними значеннями або уставками. З математичної точки зору, кожне правило у системі може бути формалізоване як комплексна логічна функція або предикат. Ця функція приймає на вхід набір параметрів, серед яких унікальний мережевий ідентифікатор сенсора, поточне числове або логічне значення вимірюваної величини, оператор математичного порівняння та цільове

порогове значення. На виході такий логічний рушій формує детермінований булевий стан або конкретне числове значення для подальшого керування визначеним виконавчим пристроєм. Впровадження такого механізму на навчальному лабораторному стенді має колосальне дидактичне значення, оскільки дозволяє наочно продемонструвати майбутнім фахівцям концепцію периферійних обчислень, принципи побудови детермінованих скінченних автоматів та алгоритми автоматизованого реагування інфраструктури на нештатні або аварійні ситуації.

Для структурування процесу розробки було проведено декомпозицію поставленої мети на окремі логічні етапи, які відображають ієрархію побудови кіберфізичної системи (рис. 2.2).

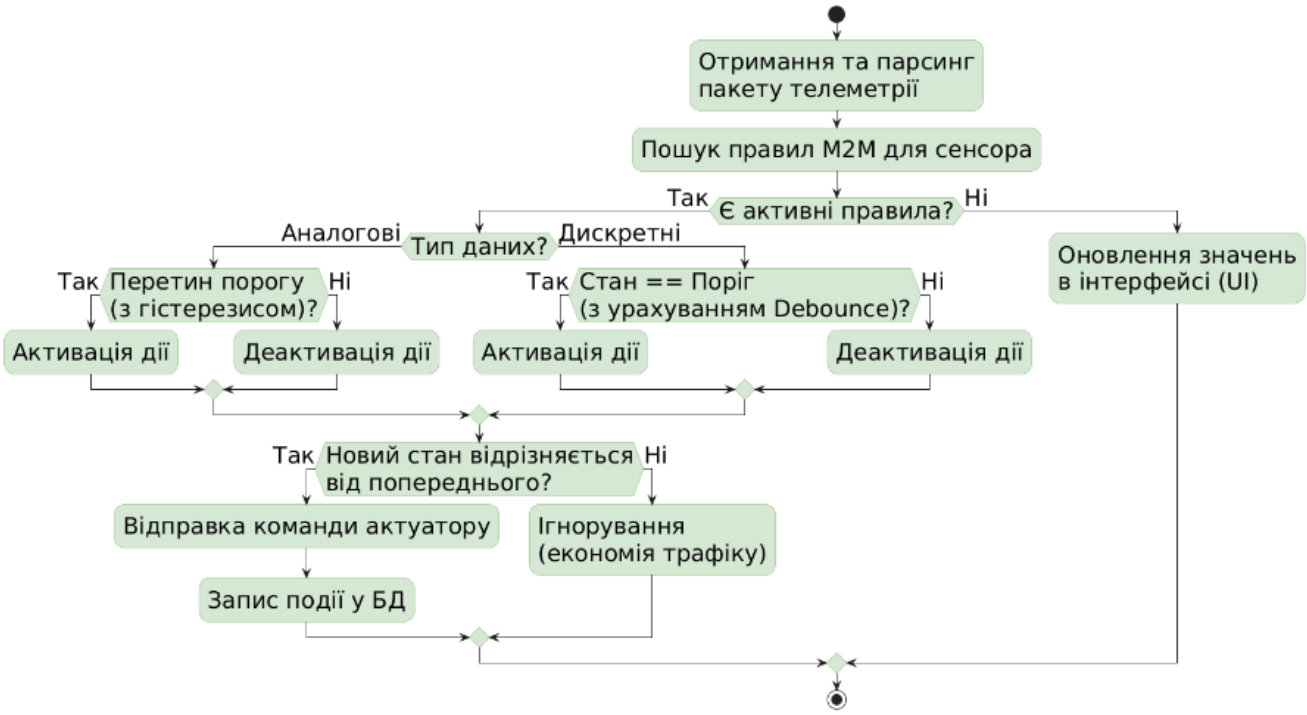


Рисунок 2.2 - Ієрархічна схема кіберфізичної системи

Представлена вище розгорнута блок-схема алгоритму детально візуалізує повний життєвий цикл обробки одного транзакційного пакета телеметричних даних всередині програмного забезпечення стенду, починаючи від моменту його первинного надходження і закінчуючи формуванням можливої фізичної реакції.

Робота цього складного алгоритму розпочинається у момент асинхронного надходження корисного навантаження від комунікаційного шлюзу до центрального сервера обробки. Програмний парсер, реалізований за допомогою вбудованих модулів мови Python, розпаковує вхідний пакет, що найчастіше передається у форматі легковагового обміну даними, і здійснює екстракцію критично важливої інформації. Наступним, надзвичайно важливим кроком з точки зору оптимізації швидкодії обчислювального ядра, є пошук залежностей у резидентній пам'яті сервера. Для досягнення максимальної продуктивності використовується структура даних на основі хеш-таблиці, що реалізована у Python у вигляді стандартного словника. Цей підхід забезпечує складність пошуку алгоритмів взаємодії, наближену до константного часу обчислень, що є критично важливим при масштабуванні системи та збільшенні кількості одночасно підключених датчиків. Якщо для ідентифікованого сенсора не знайдено жодного пов'язаного користувачем логічного правила, програмна система обмежується лише оновленням поточного стану в оперативній пам'яті та трансляцією нових даних на веб-інтерфейс через механізми веб-сокетів, повністю уникаючи марних витрат процесорного часу на подальші логічні перевірки.

У випадку успішного виявлення активного правила, алгоритм переходить до найскладнішої фази ухвалення рішення, яка на концептуальному рівні поділяється на дві окремі гілки залежно від фізичної природи вимірюваного сигналу. Для дискретних сигналів, які генеруються типовими цифровими датчиками стану на кшталт інфрачервоних детекторів руху, магнітних герконів відкриття дверей або оптичних пожежних сповіщувачів, застосовується метод строгої логічної еквівалентності з обов'язковим попереднім програмним фільтруванням. Це фільтрування полягає в алгоритмічному усуненні брязкоту контактів, що гарантує відсутність хибних спрацювань через механічні вібрації сенсорів. Після підтвердження стабільності сигналу програмне ядро перевіряє

точний збіг поточного стану датчика з пороговим значенням тривоги, встановленим у правилі.

Натомість обробка неперервних аналогових величин, таких як температура навколишнього середовища, концентрація вуглекислого газу в приміщенні, відносна вологість або рівень освітленості, становить значно складнішу математичну та інженерну задачу. Використання простого математичного порівняння у таких випадках є не лише недостатнім, але й вкрай шкідливим фактором, що загрожує стабільності та довговічності всього апаратного комплексу. Причиною цього є те, що природне коливання аналогового сигналу безпосередньо навколо заданого порогу, викликане фізичними флуктуаціями середовища або електромагнітними наведеннями на вимірювальні лінії, неминуче призводить до явища високочастотного перемикання станів. У результаті виконавчі пристрої, такі як електромагнітні реле або потужні контактори, починають вмикатися та вимикатися десятки разів на секунду, що призводить до їхнього стрімкого зносу, перегріву та неминучого виходу з ладу за лічені години експлуатації.

Для професійного інженерного вирішення цієї класичної проблеми систем автоматики, у програмному алгоритмі навчального стенду теоретично обґрунтовано та практично імплементовано строгу математичну модель гістерезису. Цей підхід полягає у цілеспрямованому створенні штучної буферної зони, яку також називають мертвою зоною, навколо цільового порогового значення. У межах цієї зони будь-яка зміна логічного стану керуючої системи примусово блокується алгоритмом. Завдяки такій інтелектуальній програмній обв'язці, для активації, наприклад, системи примусової вентиляції, рівень температури повинен чітко перевищити верхню межу гістерезису. Зворотна дія деактивації відбудеться лише тоді, коли показник опуститься значно нижче нижньої межі, повністю ігноруючи будь-які незначні мікроколивання або шуми вимірювання всередині самої буферної зони. Практична реалізація цієї потужної концепції вимагає від програмного забезпечення на мові Python постійного

відстеження та збереження попереднього стану кожного виконавчого механізму в оперативній пам'яті. Це дозволяє логічному рушію порівнювати не лише абсолютні скалярні значення вимірів, але й враховувати вектор напрямку динаміки фізичного процесу в часі.

Наступним надзвичайно важливим теоретичним аспектом алгоритму є використання концепції цифрового двійника для жорсткої оптимізації мережевого трафіку при відправці керуючих команд назад на фізичний рівень. Логіка роботи навчального стенду вимагає, щоб центральний сервер завжди зберігав у своїй оперативній пам'яті останній відомий стан кожного підключеного актуатора. Розроблений алгоритм містить обов'язковий блок перевірки зміни стану обчисленого прапорця активації, який гарантує, що керуючий мережевий пакет буде згенеровано та фактично надіслано через комунікаційний канал до актуатора лише у тому єдиному випадку, коли щойно обчислюваний цільовий стан відрізняється від поточного фізичного стану пристрою, зафіксованого у цифровому двійнику під час попереднього такту обробки. Це елегантне рішення надзвичайно ефективно запобігає засміченню обмеженого радіоефіру або локальної мережі нескінченним потоком дублюючих керуючих команд, що виникали б при кожному новому циклі опитування датчиків.

На завершальному етапі, у разі алгоритмічного підтвердження необхідності виконання нової фізичної дії, сервер формує серіалізований керуючий пакет, ініціює безпечний мережевий запит до відповідного апаратного вузла та вносить максимально деталізований запис про автоматичне спрацювання відповідного правила міжмашинної взаємодії до системного журналу бази даних. Наявність такого структурованого та захищеного хронологічного журналу є критично необхідною складовою для ефективного навчального процесу. Вона забезпечує студентам унікальну можливість проводити глибокий ретроспективний аналіз коректності роботи налаштованих ними логічних алгоритмів, математично досліджувати загальний час реакції

кіберфізичної системи на збурення, а також швидко виявляти, ізолювати та виправляти логічні помилки у власних архітектурних сценаріях автоматизації виробничих процесів.

2.3 Обґрунтування вибору апаратної елементної бази та просторової топології мережі навчального стенду

Проєктування апаратного забезпечення для освітніх цілей у галузі комп'ютерної інженерії суттєво відрізняється від створення комерційних або промислових рішень. Головним критерієм ефективності навчального стенду з Інтернету речей є його дидактична цінність, яка полягає у здатності наочно демонструвати студентам базові принципи функціонування мікроелектронних компонентів, фізику поширення сигналів та логіку роботи протоколів передачі даних. Теоретичний аналіз апаратної бази вимагає пошуку оптимального балансу між продуктивністю обчислювальних ядер, енергоефективністю кінцевих вузлів та відкритістю архітектури для глибокого дослідження. Сучасна парадигма Інтернету речей передбачає гетерогенне середовище, де малопотужні периферійні пристрої взаємодіють із високопродуктивними центральними серверами. Відповідно, апаратна реалізація стенду повинна обов'язково включати як мінімум два класи обчислювальних пристроїв: мікроконтролери для збору первинної інформації та мікропроцесорні системи або персональні комп'ютери для розгортання програмного забезпечення на мові Python, яке виконуватиме роль інтелектуального оркестратора.

Аналізуючи доступні на ринку мікроконтролерні платформи для побудови периферійних сенсорних вузлів, необхідно враховувати їхні архітектурні особливості та наявність інтегрованих комунікаційних модулів. Класичні восьмибітні мікроконтролери, незважаючи на їхню простоту та поширеність у базових курсах програмування, не в повній мірі відповідають сучасним вимогам IoT-систем через відсутність вбудованої підтримки бездротових мереж та

обмежений обсяг оперативної пам'яті, що унеможливлює використання сучасних криптографічних протоколів. Тому теоретично обґрунтованим є перехід до використання тридцятидвохбітних мікроконтролерних архітектур на базі ядер ARM Cortex-M або Xtensa. Такі кристали зазвичай містять на одному кремнієвому чипі не лише потужне арифметико-логічне пристрій, але й повноцінні радіотракти для роботи зі стандартами Wi-Fi та Bluetooth Low Energy. Використання подібних інтегрованих рішень на навчальному стенді дозволяє майбутнім інженерам досліджувати процеси підключення до локальних мереж, узгодження IP-адрес через протокол DHCP та формування пакетів прикладного рівня без необхідності розробки складних апаратних "бутербродів" із зовнішніми мережевими модулями.

Не менш важливою складовою теоретичного аналізу є вибір топології комунікаційної мережі, яка об'єднуватиме всі компоненти стенду в єдиний кіберфізичний простір. Існує декілька класичних просторових топологій, таких як кільце, шина, зірка та комірчаста мережа. Для навчального комплексу, орієнтованого на вивчення взаємодії через Інтернет-протоколи, найбільш доцільною є класична топологія типу "зірка", де центральним вузлом виступає локальний маршрутизатор або бездротова точка доступу, а всі сенсорні вузли та центральний сервер підключаються до неї як незалежні клієнти. Ця топологія забезпечує високу надійність ізольованих експериментів: вихід з ладу або зависання одного мікроконтролера під час студентських лабораторних робіт ніяк не впливає на працездатність інших компонентів мережі. Крім того, така конфігурація дозволяє легко перехоплювати та аналізувати мережевий трафік за допомогою спеціалізованого програмного забезпечення на центральному сервері, що є невід'ємною частиною вивчення дисциплін з мережевої безпеки та системного адміністрування.

Для структурування процесу розробки та глибокого розуміння фізичної взаємодії компонентів на різних рівнях абстракції було розроблено детальну

структурно-топологічну схему апаратного забезпечення навчального стенду, яка відображає ієрархію побудови кіберфізичної системи (рис. 2.3).

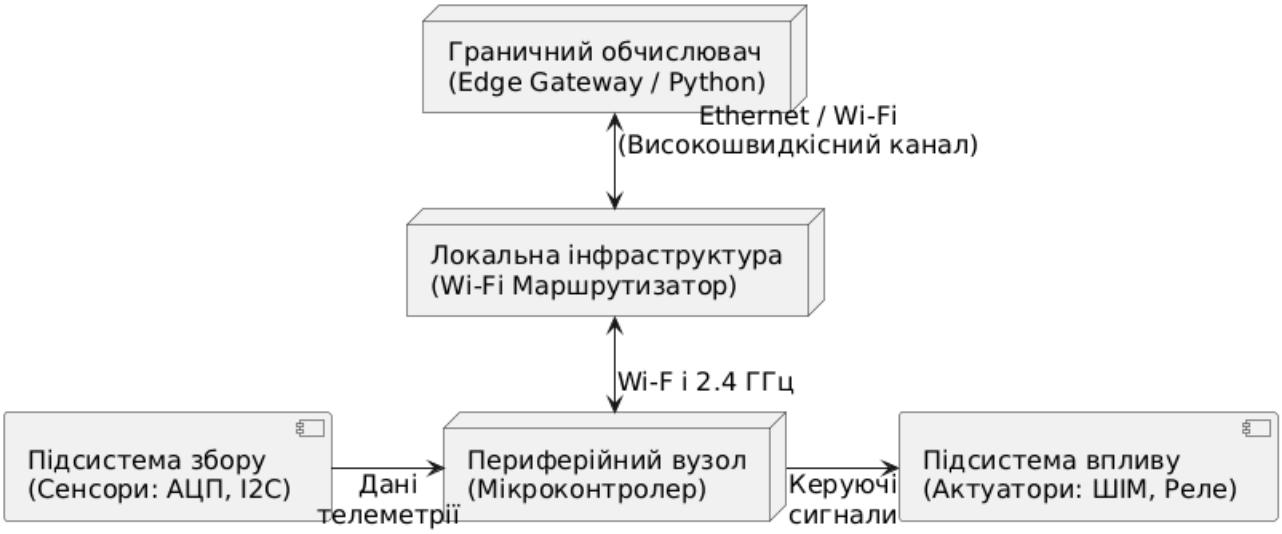


Рисунок 2.3 - Структурно-топологічна схема апаратного забезпечення навчального стенду

Детальний теоретичний розбір представленої вище структурно-топологічної схеми дозволяє простежити всю складність інженерних рішень, закладених у проєкт навчального стенду. Центральним елементом логічної структури, який бере на себе функцію інтелектуального оркестратора, є граничний обчислювач або Edge Gateway. Апаратна база цього вузла вимагає наявності повноцінної операційної системи, що дозволяє розгорнути ізольоване віртуальне середовище для виконання скриптів на мові програмування Python. Завдяки високій продуктивності мікропроцесорного ядра, центральний сервер здатний підтримувати асинхронний мережевий рушій, який одночасно обробляє сотні вхідних підключень від периферійних вузлів. З дидактичної точки зору, саме тут студенти отримують можливість застосовувати на практиці знання з об'єктно-орієнтованого програмування, роботи з базами даних та створення людино-машинних інтерфейсів, абстрагуючись від низькорівневих апаратних переривань, з якими вони стикалися б при програмуванні мікроконтролерів.

Переходячи до аналізу периферійного мікроконтролерного вузла, слід відзначити його ключову роль як мосту між аналоговим фізичним світом та цифровим середовищем передачі даних. Мікроконтролер оснащений спеціалізованим апаратним забезпеченням для взаємодії з різними типами сигналів. Теоретичне вивчення роботи аналого-цифрового перетворювача є критично важливим етапом підготовки фахівців з комп'ютерної інженерії. Студенти повинні розуміти, як безперервний сигнал напруги від резистивного датчика газу надходить на вхід АЦП, де він квантується за рівнем та дискретизується в часі. Якість цієї трансформації напряму залежить від розрядності перетворювача та стабільності опорної напруги. Аналізуючи цифрові підсистеми збору даних, схема ілюструє використання синхронної послідовної шини I2C для зв'язку з інтегральними датчиками мікроклімату. Цей інтерфейс вимагає від розробника глибокого розуміння процесів адресації пристроїв на єдиній шині, ролі підтягуючих резисторів та механізмів підтвердження прийому кожного байта даних, що формує у майбутніх інженерів навички роботи з промисловими стандартами обміну інформацією на рівні друкованої плати.

Блок виконавчих пристроїв або актуаторів, представлений на схемі, демонструє процеси зворотного перетворення цифрової інформації у фізичні дії. Для керування просторовим положенням сервомеханізмів мікроконтролер використовує апаратний генератор широтно-імпульсної модуляції (ШІМ). Теоретичне обґрунтування цього методу полягає у зміні скважності прямокутних імпульсів при незмінній несучій частоті, що дозволяє з високою точністю позиціонувати вал двигуна без необхідності використання складних цифро-аналогових перетворювачів. Керування потужними навантаженнями, такими як системи освітлення чи обігріву, реалізується через блок електромагнітних реле. Тут на перший план виходять питання електробезпеки та захисту мікропроцесорної електроніки від електромагнітних завад. Студенти вивчають необхідність застосування гальванічної розв'язки за допомогою оптопар та

використання зворотних діодів для гасіння високовольтних імпульсів самоіндукції, які виникають у момент відключення індуктивного навантаження котушки реле.

Об'єднавчою ланкою всієї архітектури виступає локальна мережева інфраструктура на базі стандарту Wi-Fi, що працює в діапазоні 2,4 гігагерца. Теоретичний аналіз цього сегмента системи відкриває широкі можливості для дослідження радіофізичних аспектів Інтернету речей. В умовах навчальної лабораторії, де одночасно можуть працювати десятки подібних стендів, радіоефір стає надзвичайно защільненим, що призводить до неминучих колізій пакетів та втрат даних на фізичному рівні. Саме тому архітектура програмного забезпечення на центральному сервері Python повинна будуватися з урахуванням ненадійності транспортного середовища, включаючи механізми автоматичного перепідключення, буферизації повідомлень та перевірки цілісності даних за допомогою контрольних сум. Таким чином, обрана апаратна топологія навчального стенду забезпечує комплексне занурення студентів у проблематику створення стійких кіберфізичних систем, охоплюючи весь спектр технологій від законів Ома на фізичному рівні до складних алгоритмів міжмашинної взаємодії на рівні програмних додатків.

2.4 Проєктування архітектури програмного забезпечення серверної частини та структури бази даних кіберфізичної системи

Ефективне функціонування будь-якого сучасного комплексу Інтернету речей неможливе без надійної, безвідмовної та високопродуктивної серверної частини, яка фактично виконує роль центрального інтелектуального оркестратора всієї кіберфізичної інфраструктури. У контексті проєктування спеціалізованого навчального стенду вибір архітектури програмного забезпечення серверного вузла має ґрунтуватися не лише на критеріях обчислювальної продуктивності, але й на фундаментальних принципах

максимальної модульності, концептуальної масштабованості та повної відкритості сирцевого коду. Це життєво необхідно для того, щоб майбутні інженери могли абсолютно вільно досліджувати приховані процеси обробки інформації, самостійно модифікувати закладену логіку взаємодії та проводити власні наукові чи практичні експерименти в рамках лабораторних практикумів. Використання високорівневої інтерпретованої мови програмування Python як базового та безальтернативного інструменту для розробки серверного ядра зумовлене її надзвичайною синтаксичною гнучкістю, наявністю потужних вбудованих механізмів асинхронного виконання мережевих завдань та величезною екосистемою відкритих спеціалізованих веб-фреймворків.

Теоретичне обґрунтування програмної архітектури обов'язково починається з визначення парадигми взаємодії між модулем безперервного збору телеметрії, логічним рушієм міжмашинної взаємодії та базовою підсистемою довготривалого збереження накопичених даних. Сучасний світовий підхід до проєктування промислового та навчального програмного забезпечення вимагає категоричної відмови від застарілих монолітних програмних конструкцій на користь гнучкої мікросервісної або жорстко структурованої модульної архітектури. У такій парадигмі кожен окремий логічний компонент системи відповідає виключно за строго визначений спектр математичних, аналітичних чи організаційних завдань і спілкується з іншими суміжними модулями виключно через стандартизовані, добре задокументовані програмні інтерфейси взаємодії. Це дозволяє вносити локальні зміни в один алгоритм без ризику порушити стабільність усієї інфраструктури, що є надзвичайно цінним інженерним досвідом для студентів.

Абсолютною та непорушною основою для прийняття будь-яких автоматизованих рішень у межах побудованої кіберфізичної системи є наявність глибоко структурованого, нормалізованого та математично достовірного сховища історичних і конфігураційних даних. Саме тому системне проєктування архітектури та структури бази даних виступає одним із найважливіших етапів

теоретичного аналізу на стадії системного інжинірингу. Для створюваного навчального стенду, який генерує безперервні високочастотні потоки телеметричних пакетів від різноманітних підключених датчиків, необхідно було на етапі планування вирішити фундаментальну дилему вибору. Вона полягала у виборі між класичними перевіреними часом реляційними базами даних та сучасними часовими рядами або NoSQL документоорієнтованими сховищами, які зараз домінують у сфері великих даних. Зважаючи на першочергову дидактичну та академічну мету поточного проєкту, найбільш виправданим, інженерно надійним та педагогічно доцільним було визнано використання саме реляційної моделі управління даними.

Вибір реляційної парадигми дозволяє майбутнім фахівцям комп'ютерної інженерії безпосередньо застосувати на практиці свої фундаментальні теоретичні знання з архітектури баз даних. Вони отримують змогу наочно вивчити базові та поглиблені принципи нормалізації відношень, детально розібратися з апаратними та програмними механізмами забезпечення транзакційної цілісності інформації за допомогою розгалуженої системи зовнішніх і первинних ключів, а також опанувати класичну мову структурованих запитів на глибокому професійному рівні. Більше того, застосування передової технології об'єктно-реляційного відображення безпосередньо у робочому середовищі Python дозволяє розробникам повністю інкапсулювати складні та багаторядкові SQL-запити всередині лаконічних об'єктних методів класів. Цей сучасний підхід феноменально підвищує загальну читабельність та підтримуваність програмного коду, що суттєво полегшує його подальшу неминучу модифікацію чи розширення в процесі проведення навчальних занять чи підготовки самостійних курсових проєктів здобувачами освіти. Для структурування процесу розробки було проведено декомпозицію поставленої мети на три основні функціональні рівні, які відображають ієрархію побудови кіберфізичної системи та строгі математичні зв'язки між незалежними програмними компонентами ядра (рис. 2.4).

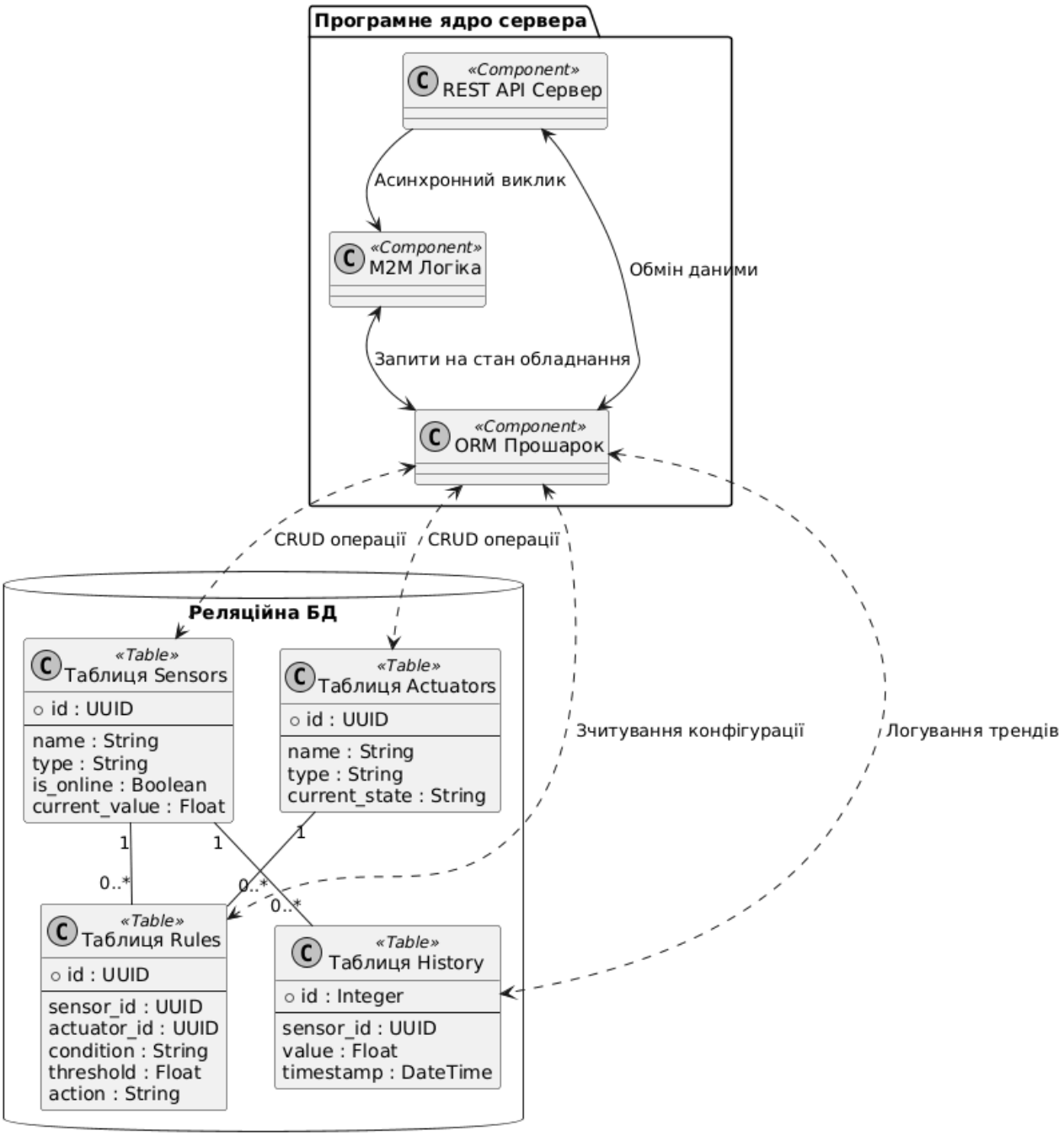


Рисунок 2.4 - Структурна схема бази даних та компонентів серверної частини

Детальний та скрупульозний теоретичний аналіз представленої вище об'єктно-реляційної структурної схеми дозволяє повною мірою усвідомити колосальну комплексність внутрішніх інформаційних та обчислювальних процесів, що безперервно та паралельно відбуваються у серверній частині створеного освітнього навчального стенду. В самій основі архітектури надійного

зберігання первинних даних лежить глибоко математично обґрунтована реляційна модель. У своїй базовій навчальній конфігурації вона складається з чотирьох ключових інформаційних сутностей, нерозривно пов'язаних між собою строгими правилами збереження посилальної та каскадної цілісності. Таблиця аналогових та дискретних датчиків разом із таблицею фізичних виконавчих пристроїв відіграють критично важливу архітектурну роль статичних реєстрів апаратного обладнання. У цих нормалізованих структурах надійно та консистентно зберігається розширена конфігураційна метаінформація про кожен зареєстрований у системі фізичний вузол. Ця інформація включає його унікальний криптографічний мережевий ідентифікатор, людинозрозумілу текстову назву для зручного відображення в графічному інтерфейсі, константний фізичний тип сенсора та актуальний логічний стан підключення до центрального транспортного сервера. Застосування універсальних унікальних ідентифікаторів замість класичних цілочисельних автоінкрементних ключів є беззаперечно визнаною у всьому світі сучасною інженерною практикою. Вона на теоретичному рівні практично повністю унеможливлює виникнення катастрофічних інформаційних колізій при подальшому просторовому масштабуванні інфраструктури системи або при ймовірному фізичному злитті кількох відокремлених навчальних стендів у єдину глобальну розподілену мережу університетської комп'ютерної лабораторії.

Центральною інтелектуальною та керуючою ланкою побудованої реляційної моделі виступає таблиця правил міжмашинної взаємодії. Саме ця сутність логічно поєднує пасивні сенсорні елементи та активні електромагнітні актуатори через класичний механізм перехресних зовнішніх ключів. Ця таблиця фактично та юридично реалізує надзвичайно ефективну програмну парадигму декларативної конфігурації замість прямого, жорсткого програмування поведінки мікроконтролерів. Завдяки свідомому впровадженню такої гнучкої абстрактної структури, будь-яка кардинальна зміна логіки роботи кіберфізичної системи більше взагалі не вимагає виснажливого та тривалого процесу

переписування, відлагодження, тестування та перекомпіляції сирцевого програмного коду самого сервера чи кінцевих периферійних пристроїв. Користувач або студент-дослідник може абсолютно динамічно, безпосередньо в режимі реального часу, додавати або видаляти нові конфігураційні рядки в цю таблицю через інтуїтивно зрозумілий графічний веб-інтерфейс. Одразу після цього потужний логічний рушій на базі Python миттєво та автоматично підхопить нові правила обробки вже у наступному такті аналізу мережевих подій. Це елегантне інженерне рішення надзвичайно яскраво та практично демонструє здобувачам вищої освіти усі незаперечні переваги сучасної сервіс-орієнтованої архітектури, цілком і повністю керованої потоком даних.

Найбільш динамічною, швидкозростаючою та ресурсомісткою частиною розробленої бази даних є монументальна таблиця історії збереженої телеметрії. У цю структуру програмна система абсолютно безперервно, з гарантованою частотою у кілька герц, записує всі найдрібніші зміни фізичних величин навколишнього середовища, які фіксуються апаратною частиною стенду. Теоретичний системний аналіз процесу проєктування цієї конкретної таблиці висуває до архітектора бази даних жорстку вимогу щодо обов'язкового створення складних багатокомпонентних індексів за полем часової мітки та унікальним ідентифікатором датчика. Причина цього полягає в тому, що без наявності таких заздалегідь оптимізованих бінарних структур дерева індексів побудова будь-яких аналітичних графіків за тривалі проміжки навчального семестру неминуче призводитиме до катастрофічної деградації обчислювальної продуктивності центрального сервера через постійне повне сканування гігабайтних таблиць на магнітному чи твердотільному накопичувачі.

Переходячи безпосередньо до детального розгляду самого програмного ядра, реалізованого потужною та лаконічною мовою Python, варто звернути виняткову увагу на визначальну роль механізму об'єктно-реляційного відображення у забезпеченні загальної кібернетичної відмовостійкості системи. Цей передовий архітектурний патерн програмного проєктування створює

надійний, непробивний абстрактний прошарок між об'єктно-орієнтованим кодом бізнес-логіки та низькорівневими реляційними таблицями ядра бази даних. Він повністю автоматично генерує абсолютно безпечні, параметризовані та синтаксично вивірені запити. З точки зору забезпечення жорсткої кібербезпеки в масштабній інфраструктурі Інтернету речей, таке технологічне рішення є не просто бажаним, а критично обов'язковим етапом розробки. Воно на фундаментальному математичному рівні захищає навчальний серверний стенд від найбільш поширених хакерських атак типу впровадження несанкціонованого коду, автоматично та повністю прозоро для студента-програміста екрануючи всі без винятку вхідні текстові параметри. Ці параметри можуть постійно надходити від потенційно скомпрометованих, заражених шкідливим програмним забезпеченням або просто алгоритмічно неправильно налаштованих периферійних мікроконтролерів. Крім виконання надзвичайно важливих безпекових функцій, цей спеціалізований програмний прошарок також ефективно бере на себе вкрай складне та ресурсомістке системне завдання з управління динамічним пулом активних з'єднань із базою даних. Це забезпечує математично оптимальне використання завжди обмежених системних ресурсів оперативної пам'яті головного сервера при одночасному паралельному обслуговуванні сотень чи навіть тисяч інтенсивних мережевих запитів від усього парку підключених периферійних пристроїв.

Ключовим обчислювальним та організаційним компонентом створеного модульного програмного ядра є багатопотоковий асинхронний сервер, який бездоганно виконує критичну функцію єдиної уніфікованої точки входу для абсолютно всієї транзитної інформації у розробленій кіберфізичній системі. Фундаментальна теоретична та інженерна перевага використання класичного стандарту передачі стану представлення полягає у його абсолютно повній та беззаперечній незалежності від типу апаратної архітектури чи операційної системи кінцевого клієнта. Абсолютно не має значення, чи це малопотужний бюджетний мікроконтролер, прошивка якого написана на низькорівневій мові

програмування з ручним керуванням пам'яттю, чи це складний сучасний графічний веб-інтерфейс, реалізований на мові виконання браузерних скриптів. В обох випадках використовуються абсолютно однакові уніфіковані мережеві кінцеві точки та суворо стандартизовані методи прикладного рівня для взаємодії з єдиним головним сервером. Коли цифровий пакет із телеметричними чи діагностичними даними успішно надходить до комунікаційного інтерфейсу, центральний сервер не блокує свою подальшу роботу в очікуванні тривалого процесу запису цієї інформації на повільні носії бази даних. Натомість він миттєво та елегантно передає управління подією спеціалізованому модулю оркестрації через надійний механізм асинхронних неблокуючих викликів. Одразу після цього модуль системної логіки оперативно завантажує у швидку кеш-пам'ять найактуальніші конфігураційні правила для виявленого конкретного датчика, проводить складні обчислення логічних та математичних умов з обов'язковим урахуванням раніше докладно описаного алгоритмічного гістерезису. У разі підтвердження умови він так само миттєво формує відповідні зворотні пакети для максимально швидкого реагування виконавчих пристроїв. Вся ця багатогранна та складна програмно-апаратна архітектура, яка скрупульозно та педантично побудована на строгих теоретичних засадах сучасних комп'ютерних наук, цифрової мікроелектроніки та класичної теорії автоматичного управління, створює винятково потужний, інженерно надійний та надзвичайно дидактично цінний фундамент. Цей фундамент призначений для проведення студентами глибоких лабораторних досліджень поведінки багатокомпонентних мереж, тестування відмовостійкості систем та набуття практичних навичок у рамках академічного навчального процесу з підготовки висококваліфікованих інженерів-програмістів та компетентних фахівців з кіберфізичних систем.

2.5 Висновки до другого розділу

У другому розділі кваліфікаційної роботи було проведено глибокий теоретичний аналіз, системне проєктування та математичне моделювання усіх ключових підсистем розроблюваного навчального апаратно-програмного комплексу з Інтернету речей. Процес проєктування базувався на сучасному методологічному підході до створення кіберфізичних систем, який передбачає чітку декомпозицію складної інфраструктури на незалежні, але тісно взаємопов'язані архітектурні рівні. Дослідження теоретичних засад міжрівневої взаємодії дозволило обґрунтувати необхідність використання розширеної багаторівневої моделі, де чітко розмежовуються фізичні процеси збору первинної інформації, мережеві механізми транспортування пакетів даних та високорівневі алгоритми оркестрації і аналітики. Такий підхід забезпечує не лише високу інженерну надійність створюваного рішення, але й формує ідеальне дидактичне середовище, у якому студенти можуть послідовно та ізольовано вивчати кожен етап життєвого циклу інформації: від виникнення аналогового електричного сигналу на клемах датчика до побудови інтерактивних графіків у веб-браузері.

Значну увагу в межах розділу було приділено розробці та теоретичному моделюванню алгоритмів міжмашинної взаємодії, які становлять основу інтелектуальної поведінки будь-якої сучасної системи автоматизації. Відмова від застарілої парадигми синхронного циклічного опитування на користь асинхронної подієво-орієнтованої архітектури дозволила кардинально знизити обчислювальне навантаження на центральний сервер та мінімізувати затримки в мережі. Для вирішення класичних проблем систем автоматичного керування, зокрема явища високочастотного брязкоту контактів виконавчих механізмів при коливаннях вимірюваних величин поблизу порогових значень, було успішно імплементовано математичну модель гістерезису. Додаткове впровадження концепції цифрового двійника в оперативну пам'ять програмного ядра на базі

мови Python забезпечило надійний механізм фільтрації надлишкового мережевого трафіку, оскільки керуючі команди відправляються на фізичний рівень виключно у випадках фактичної необхідності зміни стану актуатора. Ці інженерні рішення гарантують високу стабільність роботи стенду та значно подовжують експлуатаційний ресурс електромеханічних компонентів.

На основі сформованих алгоритмічних вимог було здійснено комплексний теоретичний аналіз та обґрунтований вибір апаратної елементної бази й просторової топології мережі. Доведено, що для побудови периферійних сенсорних вузлів сучасного навчального комплексу найбільш доцільним є використання високопродуктивних тридцятидвохбітних мікроконтролерних систем з інтегрованими радіотрактами.

Завершальним етапом теоретичного проєктування стала детальна розробка архітектури серверного програмного забезпечення та структури реляційної бази даних. Обґрунтовано використання реляційної моделі управління даними, яка завдяки механізмам нормалізації та підтримці транзакційної цілісності ідеально підходить для фіксації структурованої конфігураційної інформації та безперервних потоків телеметрії. Застосування патерну об'єктно-реляційного відображення в синергії з асинхронним сервером додатків забезпечує не лише високу пропускну здатність системи при обробці вхідних пакетів, але й гарантує бездоганний рівень кібербезпеки, автоматично захищаючи навчальний стенд від впровадження шкідливого коду.

Таким чином, розроблений у другому розділі комплекс взаємопов'язаних теоретичних, архітектурних та алгоритмічних рішень повністю описує логіку функціонування майбутньої кіберфізичної системи. Здійснене проєктування створює вичерпне, науково обґрунтоване та деталізоване технічне підґрунтя для переходу до наступного етапу виконання кваліфікаційної роботи, а саме до безпосередньої фізичної реалізації апаратної частини стенду та написання програмного коду модулів системи керування мовою Python, що буде докладно розкрито у третьому розділі.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНОЇ СИСТЕМИ НАВЧАЛЬНОГО СТЕНДУ З ІНТЕРНЕТУ РЕЧЕЙ

3.1 Опис реалізації модулів апаратного та програмного забезпечення програмно-технічного засобу

На етапі практичної реалізації розроблюваного навчального стенду з Інтернету речей головним інженерним завданням стало безшовне поєднання фізичної апаратної бази з гнучким керуючим програмним забезпеченням. Програмно-технічний засіб створювався як комплексна гетерогенна система, що складається з периферійних мікроконтролерних вузлів для безпосереднього збору даних та центрального сервера, який бере на себе всю обчислювальну і логічну роботу. Вибір апаратних модулів базувався на необхідності забезпечення максимальної репрезентативності фізичних процесів, які вивчатимуться студентами. До складу стенду увійшли різноманітні сенсори, такі як датчики температури, вологості, концентрації вуглекислого газу, детектори чадного газу, ультразвукові далекоміри, а також дискретні датчики відчинення дверей, детектори руху та пожежні сповіщувачі. Ці первинні перетворювачі інформації фізично підключаються до портів вводу-виводу мікроконтролерної платформи, яка здійснює їх первинне опитування та формує мережеві пакети для передачі на центральний сервер.

Крім модулів збору інформації, апаратна частина включає блок виконавчих пристроїв, які дозволяють системі фізично впливати на навколишнє середовище. Реалізація цього блоку охоплює модулі керування системами вентиляції, розумними лампами освітлення, акустичними сиренами тривоги, електромагнітними замками та сервомоторами для керування кутом нахилу механічних елементів, наприклад, віконних жалюзі. Важливим елементом зворотного зв'язку є наявність інформаційного рідкокристалічного дисплея та світлодіодної стрічки, які візуалізують поточний стан системи безпосередньо на фізичному макеті. Усі ці апаратні компоненти функціонують під управлінням

єдиного центрального ядра, програмне забезпечення якого розроблено мовою високого рівня Python.

Програмна реалізація серверної частини базується на використанні сучасного мікрофреймворку Flask. Цей інструмент було обрано завдяки його легковаговій архітектурі, яка ідеально підходить для розробки інтерфейсів прикладного програмування у навчальних цілях. Серверна програма виконує роль інтелектуального оркестратора, який безперервно приймає запити від апаратних вузлів, аналізує отриману телеметрію та формує керуючі команди. Одним із найважливіших архітектурних рішень при написанні програмного коду стало використання об'єктно-орієнтованої парадигми для моделювання цифрових двійників фізичних пристроїв. У вихідному коді системи реалізовано базові класи для датчиків та виконавчих механізмів, які повністю інкапсулюють у собі всі необхідні технічні характеристики обладнання.

При створенні нового екземпляра класу в оперативній пам'яті сервера йому автоматично генерується криптографічно стійкий унікальний ідентифікатор. Цей підхід фундаментально вирішує проблему можливих колізій ідентифікаторів при масштабуванні системи. Програмний об'єкт сенсора містить атрибути для збереження його текстової назви, фізичного типу, поточного числового або логічного значення, а також прапорця мережевої доступності, який дозволяє системі визначати обрив зв'язку з фізичним пристроєм. Аналогічно, об'єкт виконавчого пристрою зберігає свій тип та актуальний стан, який може бути представлений булевим значенням для реле, числовим значенням для кута повороту сервоприводу або текстовим рядком для інформаційного табло. Така об’єктна декомпозиція дозволяє створити абстракцію, де кожен фізичний вузол розглядається як окрема програмна сутність із визначеним набором методів для взаємодії. Завдяки механізмам інкапсуляції, серверна логіка оперує не низькорівневими потоками бітів, а структурованими даними, що значно підвищує читабельність коду та полегшує процес тестування окремих модулів системи. Крім того, використання такої

парадигми відкриває можливості для легкого інтегрування нових типів периферійних модулів у майбутньому шляхом простого наслідування базових класів (рис. 3.1).

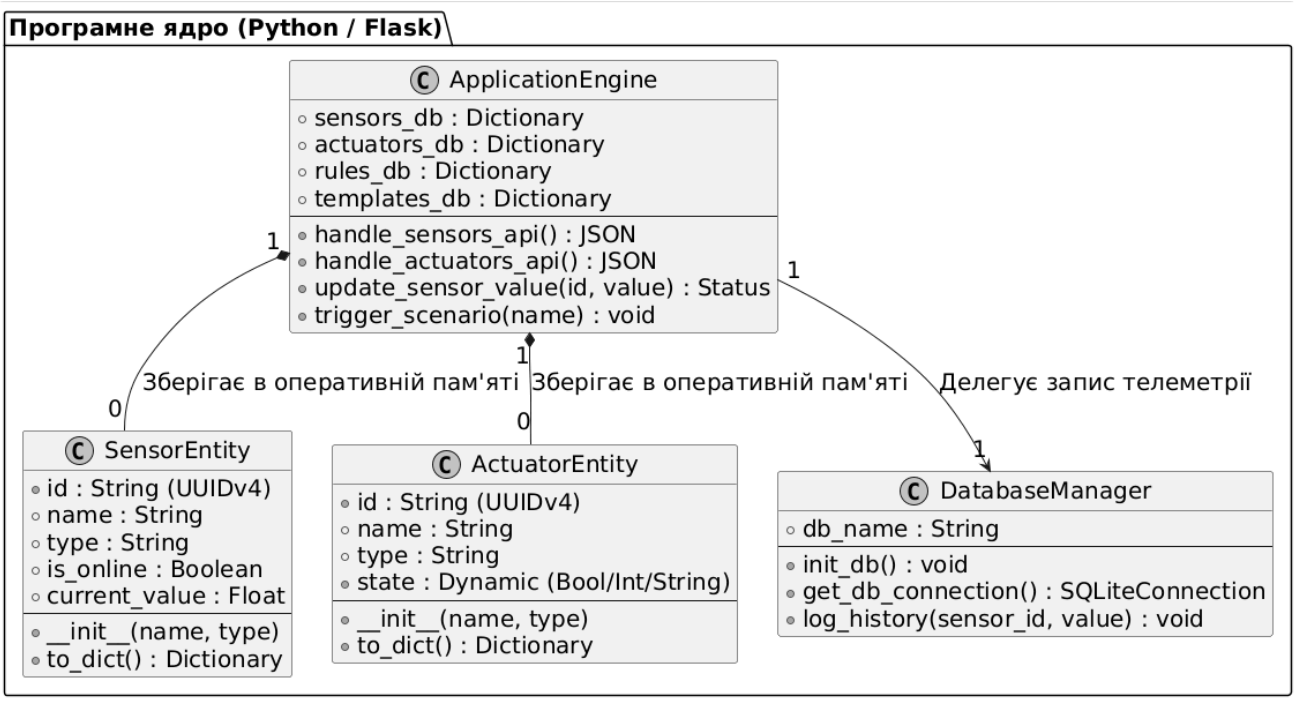


Рисунок 3.1 - Діаграма класів програмного забезпечення серверного вузла системи

Детальний аналіз наведеної діаграми класів розкриває внутрішню механіку обробки інформації у програмному засобі. Головний модуль програми керує кількома глобальними словниками, які виконують роль надшвидкої оперативної бази даних. Зберігання об'єктів датчиків та актуаторів безпосередньо у пам'яті процесора, а не на жорсткому диску, дозволяє досягти мінімальних затримок при виконанні алгоритмів автоматизації. Коли мікроконтролер надсилає новий пакет з даними через мережу, головний рушій системи миттєво знаходить відповідний об'єкт за його унікальним ідентифікатором у словнику та оновлює його поточне значення. Одразу після цього викликається функція перевірки правил взаємодії, яка приймає рішення щодо необхідності впливу на виконавчі механізми. Для взаємодії із зовнішнім світом у класах передбачено спеціальний метод

серіалізації, який трансформує складний об'єкт мови Python у стандартизований формат передачі даних для подальшого відправлення клієнтським додаткам.

Візуалізація процесів моніторингу та керування модулями реалізована за допомогою веб-інтерфейсу, який є невід'ємною частиною програмного комплексу. Користувацький інтерфейс забезпечує графічне представлення всіх цифрових двійників апаратного забезпечення у вигляді інтерактивних карток. Він динамічно адаптується під тип підключеного обладнання, виводячи відповідні одиниці виміру, елементи індикації тривог та інструменти ручного коригування значень (рис. 3.2).

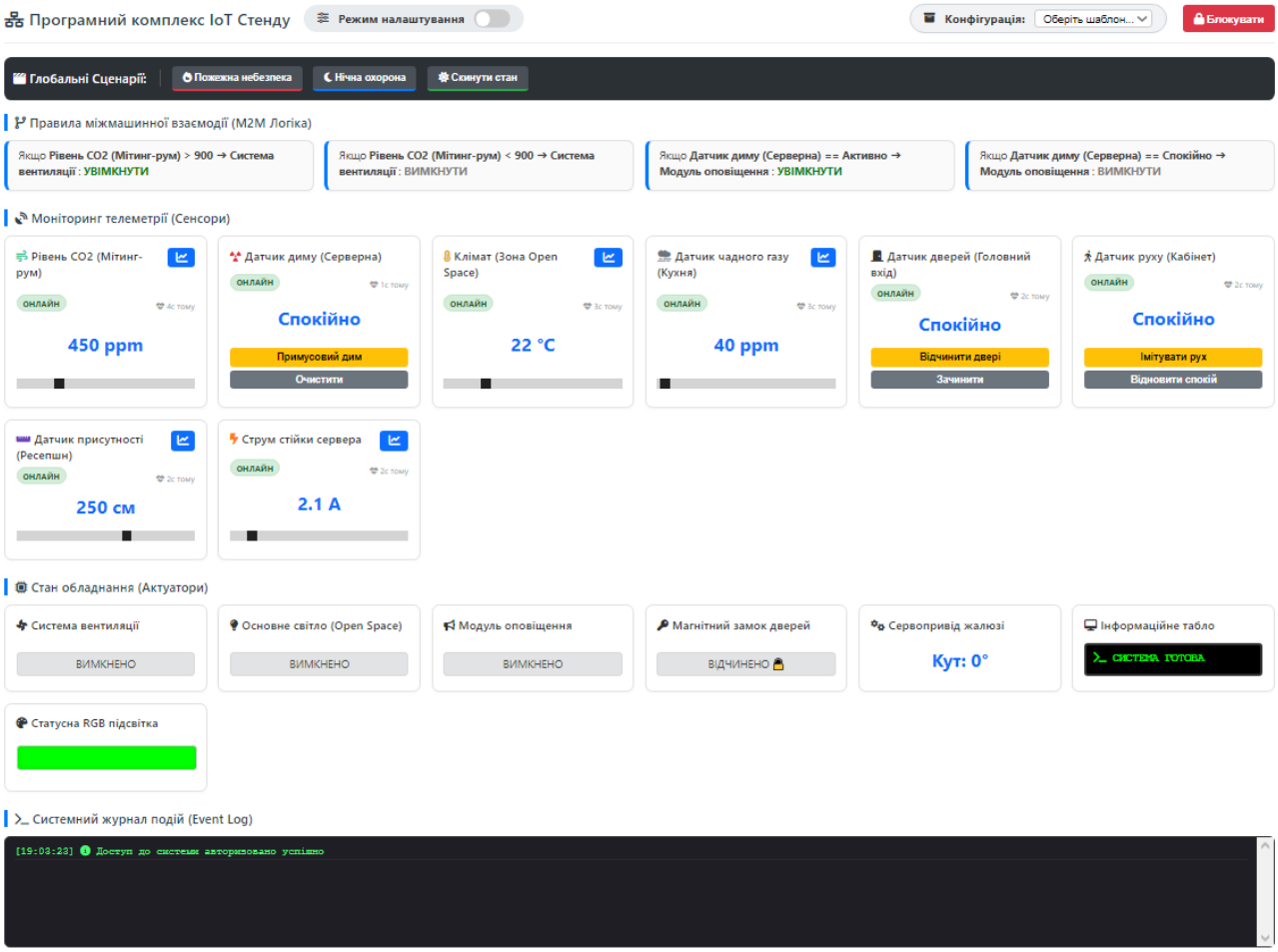


Рисунок 3.2 - Головна панель моніторингу та керування апаратно-програмним комплексом навчального стенду

Цей інтерфейс не лише відображає поточні показники телеметрії, але й надає інженерний доступ до управління мережевим станом кожного вузла. У разі необхідності викладач або студент може програмно імітувати обрив лінії зв'язку для будь-якого датчика, що дозволяє досліджувати поведінку автоматизованої системи в умовах апаратних збоїв.

Для більш глибокого розкриття архітектурних особливостей та інформаційних потоків між модулями апаратного і програмного забезпечення навчального стенду, наведено дві додаткові UML-діаграми. Перша з них є діаграмою компонентів (рис. 3.3), яка наочно ілюструє фізичний та логічний розподіл підсистем між мікроконтролерним рівнем, серверним ядром на базі фреймворку Flask та клієнтським веб-інтерфейсом.

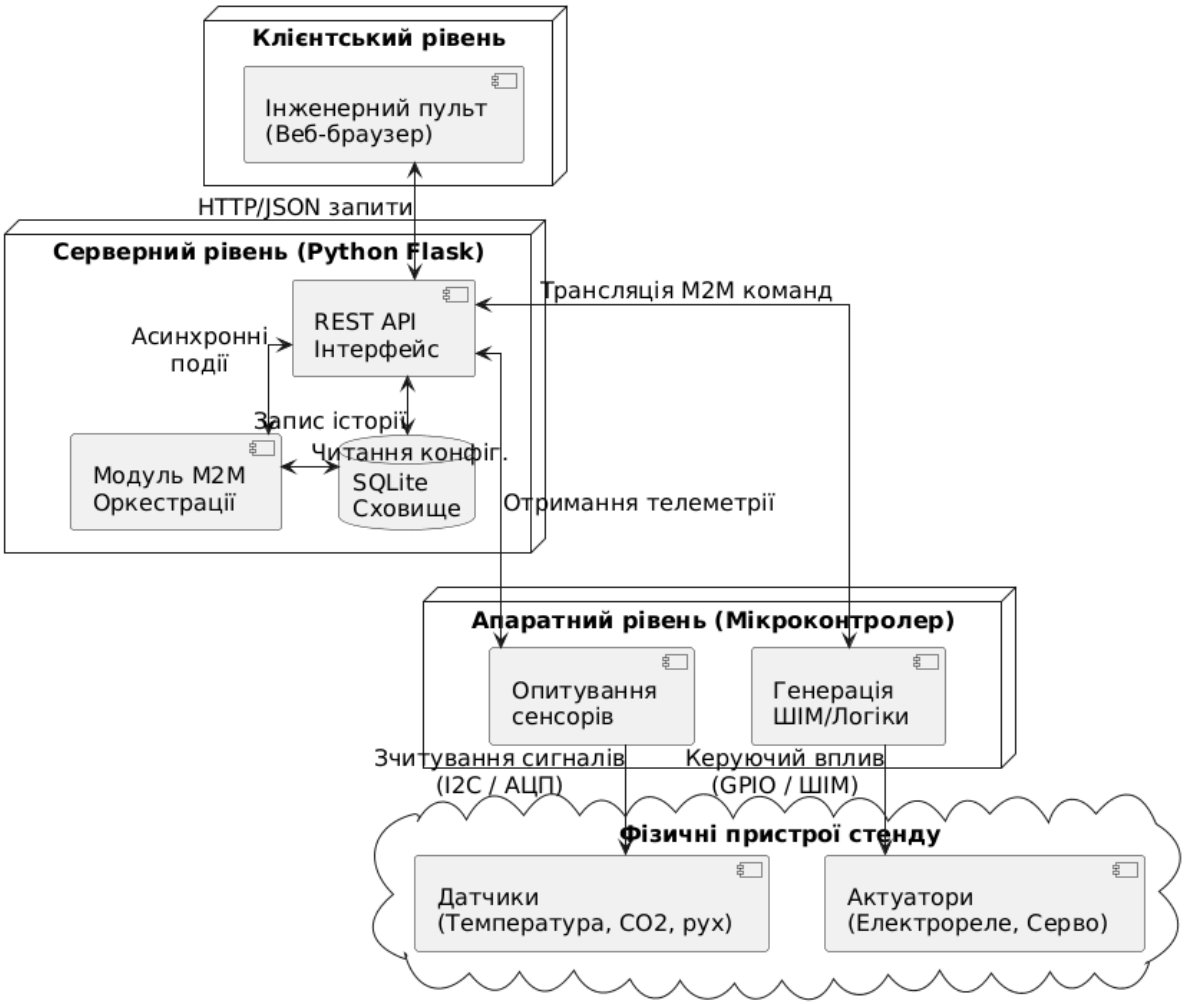


Рисунок 3.3 - Діаграма компонентів програмно-апаратного комплексу

Друга є діаграмою послідовності (рис. 3.4), що детально описує життєвий цикл проходження телеметричного пакету: від моменту його генерації на фізичному датчику до обробки логічним рушієм міжмашинної взаємодії, збереження в базі даних та відправки відповідної керуючої команди на виконавчий механізм з подальшим оновленням графічного інтерфейсу користувача.

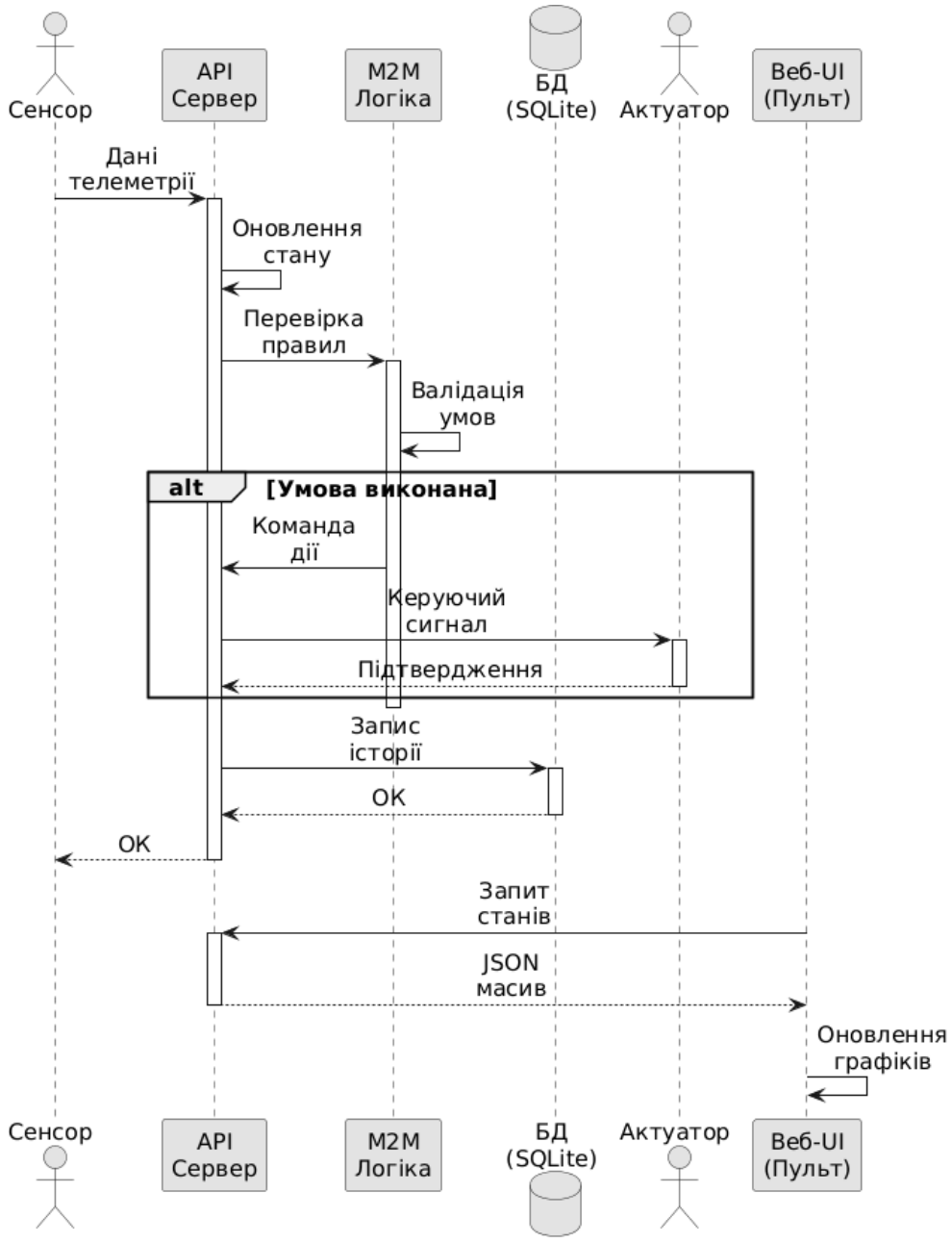


Рисунок 3.4 - Діаграма послідовності обробки телеметрії та спрацювання M2M логіки

3.2 Опис процесу створення та архітектури бази даних кіберфізичної системи

Наступним критично важливим етапом практичної реалізації апаратно-програмного комплексу навчального стенду стало проєктування, структурування та безпосереднє розгортання бази даних. У сучасних кіберфізичних просторах ефективність та надійність збереження інформації безпосередньо впливає на швидкість прийняття автономних рішень логічними модулями, а також на загальну відмовостійкість інфраструктури. Зважаючи на специфіку розроблюваного навчального стенду, який працює з безперервними динамічними потоками телеметрії від різноманітних датчиків, було впроваджено та обґрунтовано гібридну архітектуру даних. Вона ефективно поєднує надшвидке резидентне кешування поточного стану пристроїв безпосередньо в оперативній пам'яті сервера з довготривалою фіксацією часових рядів та аналітичних трендів у реляційній базі даних під управлінням вбудованої системи SQLite. Такий інтегрований підхід дозволив повністю нівелювати затримки на повільні дискові операції введення-виведення під час виконання автоматизованих алгоритмів міжмашинної взаємодії, одночасно забезпечуючи надійне збереження історичного журналу подій для подальшого аналізу.

Обрана система керування базами даних SQLite є ідеальним інструментом для вбудованих та навчальних комплексів комп'ютерної інженерії, оскільки вона не потребує розгортання окремого сервера, є повністю автономною, демонструє високу швидкодію при операціях читання-запису та підтримує загальноприйнятий стандарт мови структурованих запитів SQL. Для структурування інформаційних потоків та забезпечення суворої цілісності даних буловедено декомпозицію архітектури сховища на логічні сутності, які відображають ієрархію побудови кіберфізичної системи (рис. 3.5).

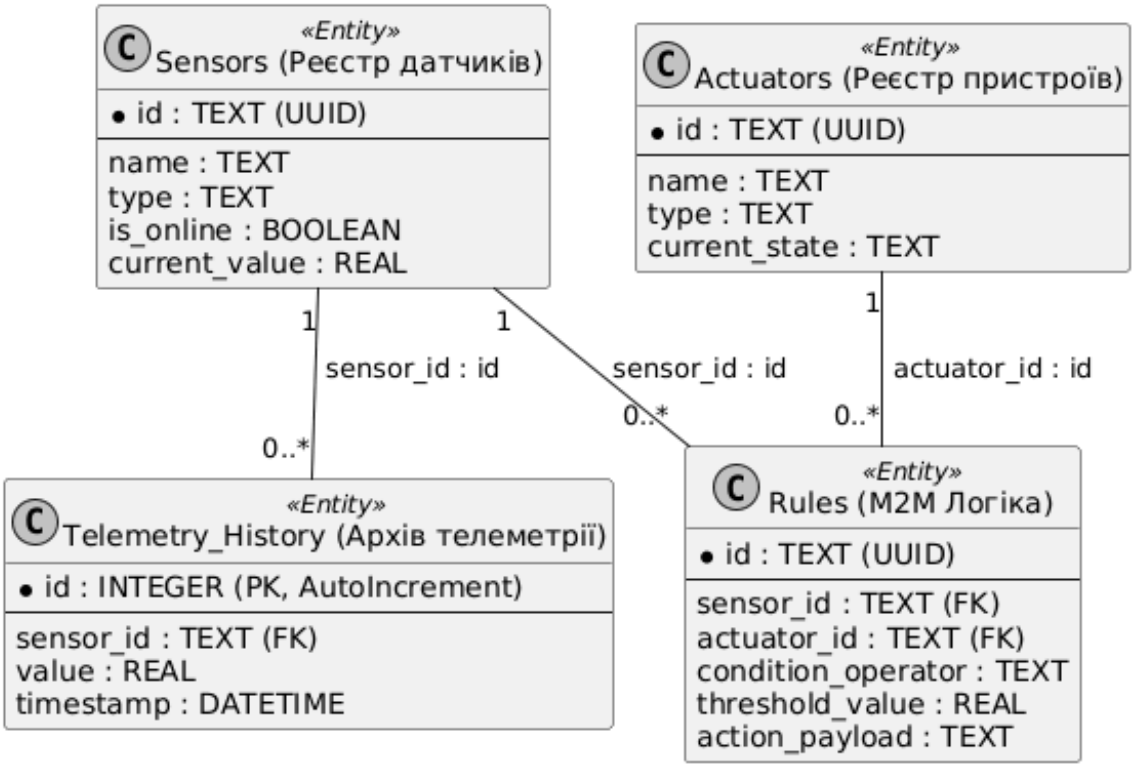


Рисунок 3.5 - Логіко-фізична модель реляційної бази даних кіберфізичної системи

Детальний та послідовний аналіз представленої логіко-фізичної схеми організації даних дозволяє чітко простежити системні взаємозв'язки між усіма апаратними та програмними модулями створеного стенду. Першою фундаментальною сутністю структури є таблиця датчиків, яка виконує роль цифрового реєстру для всіх підключених периферійних засобів вимірювання. Кожен рядок у цій таблиці характеризується унікальним первинним ключем текстового формату, що формується програмно за стандартом UUID версії чотири, повністю виключаючи ймовірність колізій ідентифікаторів у розподіленій мережі. Поля назви та типу дозволяють серверній частині однозначно визначати фізичну природу пристрою, а атрибут поточного значення динамічно оновлюється в оперативній пам'яті сервера під час кожного такту отримання нового пакету. Важливу функцію виконує логічний прапорець мережевої доступності, який сигналізує людино-машинному інтерфейсу про

наявність стабільного з'єднання або раптовий обрив лінії зв'язку з кінцевим мікроконтролером.

Другою важливою сутністю виступає таблиця виконавчих пристроїв або актуаторів, яка логічно структурована подібно до реєстру датчиків, проте орієнтована на фіксацію поточних командних станів та керуючих сигналів. Головною архітектурною особливістю цієї сутності є уніфіковане текстове поле поточного стану, яке на рівні бізнес-логики сервера інтерпретується динамічно залежно від фізичного типу пристрою. Для дискретних реле вентиляції, освітлення чи звукового сповіщення стан набуває логічного значення увімкнено або вимкнено, для сервомеханізмів жалюзі -цілого числа у градусах повороту вала, для інформаційного символьного табло -текстового рядка повідомлення, а для статусної підсвітки -шістнадцяткового коду палітри кольорів. Таке інженерне рішення забезпечує високу гнучкість системи при додаванні нових типів обладнання без необхідності фізичної перебудови структури таблиць.

Логічним єднальним вузлом реляційної моделі виступає таблиця правил міжмашинної взаємодії, яка реалізує зв'язки типу один до багатьох між реєстрами сенсорів та актуаторів за допомогою механізму зовнішніх ключів. Ця сутність надійно зберігає конфігураційні предикати, які безпосередньо керують автономною поведінкою всього комплексу. Поле оператора умови фіксує знаки математичного порівняння, поле порогового значення визначає числову межу спрацювання автоматики, а поле корисного навантаження містить конкретний стан, який має бути примусово переданий на виконавчий пристрій у разі істинності логічного виразу. Завдяки такій структурі зміна поведінки кіберфізичної системи відбувається шляхом банальної модифікації рядків у таблиці правил безпосередньо через графічний веб-пульт, повністю усуваючи потребу в ручному перепрограмуванні сервера.

Найбільш динамічною та швидко зростаючою частиною розробленого сховища є таблиця архіву історії телеметрії, яка призначена для тривалого накопичення часових рядів вимірювань аналогових та дискретних датчиків. На

відміну від вищеописаних реєстрів поточного стану, які перезаписують свої поля при кожному новому пакеті від мікроконтролера, таблиця історії функціонує виключно в режимі додавання нових рядків. Кожен запис містить зовнішній ключ посилання на первинний датчик, числове зафіксоване значення та точну часову мітку, яка автоматично генерується вбудованими механізмами бази даних у локальному форматі часу. Для оптимізації обсягу сховища та унеможливлення переповнення дискового простору під час тривалих цілодобових лабораторних випробувань стенду, у програмному коді сервера реалізовано алгоритм автоматичного очищення, який примусово обмежує максимальний розмір таблиці історії, видаляючи найстаріші рядки при перевищенні встановленого ліміту у тисячу записів.

На етапі безпосереднього фізичного розгортання бази даних у програмному середовищі Python за допомогою стандартного модуля взаємодії з SQLite створюється відповідна таблиця історії. Процес ініціалізації супроводжується виконанням структурованого SQL-запиту, який створює таблицю лише у разі її початкової відсутності в директорії проєкту. Для забезпечення максимальної швидкодії підсистеми аналітики при побудові графіків трендів, на рівні бази даних теоретично обґрунтовано та практично створено індекси за полем часової мітки та унікальним ідентифікатором сенсора. Це дозволило скоротити час пошуку історичних даних асинхронним сервером до мінімальних значень, забезпечуючи плавну та безперервну візуалізацію графіків часових рядів у веб-інтерфейсі пульта оператора.

На даному етапі опису практичної реалізації, для наочної верифікації структури створених таблиць та успішного виконання SQL-скриптів ініціалізації, у роботі доцільно навести графічне підтвердження відображення бази даних у візуальному менеджері.

3.3 Проєктування та розробка програмного забезпечення інтерактивного людино-машинного інтерфейсу

Для забезпечення зручної взаємодії оператора з апаратними компонентами та логічним ядром системи було розроблено повноцінний веб-інтерфейс, який функціонує як центральний інженерний пульт кіберфізичної системи. Клієнтська частина побудована з використанням класичного стеку технологій фронтенд-розробки. Структурування візуальних блоків виконано за допомогою мови гіпертекстової розмітки, адаптивний дизайн та стилізація реалізовані через каскадні таблиці стилів, а вся асинхронна логіка взаємодії з сервером написана мовою сценаріїв JavaScript. Такий архітектурний підхід дозволяє розгортати пульт керування на будь-якому сучасному пристрої, оснащеному веб-браузером, без необхідності встановлення спеціалізованого програмного забезпечення чи драйверів.

Критично важливим аспектом розробки систем промислової та домашньої автоматизації є базове розмежування доступу. У розробленому програмному забезпеченні цей принцип реалізовано шляхом створення віртуального екрану блокування пульта управління (рис 3.6). При першому завантаженні сторінки або при ініціалізації ручного блокування весь робочий інтерфейс перекривається темним напівпрозорим шаром, у центрі якого розташовується модуль авторизації. Логіка роботи цього модуля перехоплює натискання віртуальних клавіш, маскує введені символи для запобігання візуальному перехопленню пароля та надсилає запит на валідацію після натискання клавіші підтвердження.

У разі успішної перевірки введеного коду на стороні сервера, система знімає блокування, відкриваючи оператору повний доступ до головної панелі моніторингу та керування. Якщо ж автентифікаційні дані виявляються хибними, інтерфейс генерує візуальне попередження та автоматично очищує поле вводу. Такий механізм є критично необхідним в умовах відкритого навчального

середовища, оскільки надійно захищає виконавчі пристрої та конфігурацію логіки системи від несанкціонованого втручання сторонніх осіб.

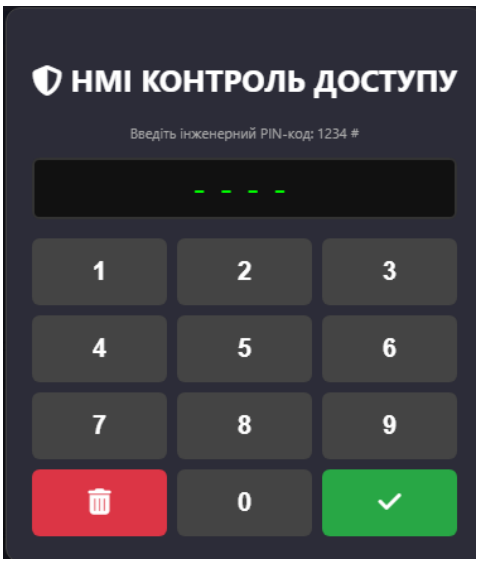


Рисунок 3.6 - Компонент графічного інтерфейсу для введення інженерного ПІН-коду

Після успішної авторизації скрипт приховує екран блокування та відкриває доступ до головної інформаційної панелі. Архітектура робочого простору поділена на логічні функціональні зони. Основою моніторингу є панель телеметрії, яка генерується абсолютно динамічно. Клієнтський додаток використовує функцію періодичного опитування, яка кожні півтори секунди ініціює асинхронний мережевий запит до сервера. Отриманий масив даних обробляється браузером, після чого скрипт перемальовує відповідні блоки об'єктної моделі документа. Завдяки цьому оператор бачить зміну показників температури, вологості чи стану задимлення в режимі реального часу. Особливої уваги заслуговує алгоритм візуалізації якості зв'язку. Програмний код аналізує прапорець мережевої доступності кожного датчика і, у разі його активності, генерує пульсуючий індикатор серцебиття з випадковим інтервалом часу, що створює ефект безперервної фонової взаємодії з фізичним обладнанням. Керування виконавчими пристроями реалізовано через окремий блок

інтерфейсу, який також генерується динамічно залежно від конфігурації обладнання. Програмний скрипт розпізнає фізичний тип актуатора та виводить відповідний елемент управління (рис 3.7).

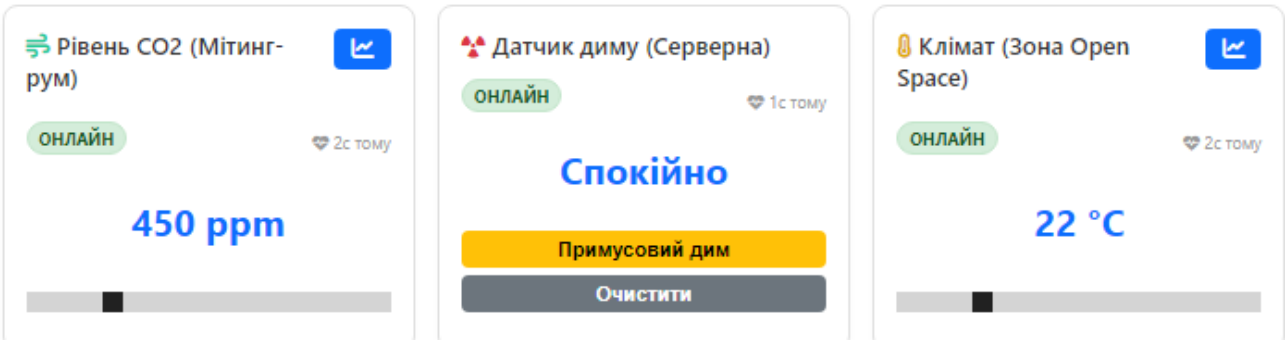


Рисунок 3.7 - Динамічні компоненти відображення стану цифрових та аналогових сенсорів

Для звичайних реле відображаються інформаційні бейджі поточного стану з кольоровим кодуванням, для сервомоторів виводиться точний кут повороту, а для RGB-підсвітки демонструється поточний згенерований колір. Усі зміни станів супроводжуються анімаційними переходами, що покращує сприйняття інформації оператором.

Особливою розробкою в межах клієнтської частини є інженерний режим конфігурації (рис. 3.8), який активується відповідним перемикачем. Цей режим інкапсулює в собі функціонал конструктора правил міжмашинної взаємодії. Форма конструктора написана таким чином, щоб мінімізувати ймовірність помилки користувача. Скрипт динамічно підвантажує у випадаючі списки всі активні сенсори та актуатори. Якщо користувач обирає дискретний датчик, наприклад детектор руху, система автоматично приховує поле вводу числових значень і пропонує обрати лише логічний стан спокою або тривоги. Сформоване правило пакується у стандартизований об'єкт і відправляється на сервер для збереження в базі даних.

Для поглибленого аналізу фізичних процесів у системі реалізовано модуль візуалізації історичних трендів. Цей модуль побудовано з використанням

спеціалізованої бібліотеки обробки графіки. При активації функції аналітики для конкретного датчика, клієнтський скрипт робить запит до історичної бази даних сервера, отримує масив останніх вимірювань та малює лінійний графік на віртуальному полотні.

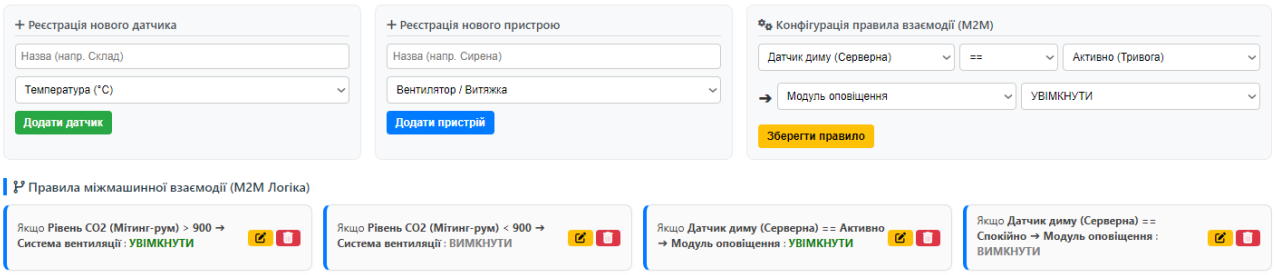


Рисунок 3.8 - Інтерфейсний модуль конструктора логіки автоматизованого керування

Якщо для обраного датчика існує активне правило з числовим порогом спрацювання, скрипт додатково розраховує координати та накладає поверх графіка пунктирну лінію уставки. Це дидактичне рішення дозволяє майбутнім інженерам наочно досліджувати швидкість реакції автоматики на перетин критичної межі фізичного параметра.

Усі події в системі, починаючи від успішної авторизації і закінчуючи спрацюванням алгоритмів пожежної тривоги, перехоплюються функцією глобального логування. Ця функція створює текстові записи з точними часовими мітками та розміщує їх у вікні системного журналу в нижній частині екрану. Використання кольорового кодування для помилок, попереджень та інформаційних повідомлень дозволяє оператору миттєво оцінювати загальний стан здоров'я кіберфізичної інфраструктури без необхідності детального вивчення кожного окремого пристрою. Крім того, накопичений архів подій автоматично синхронізується з реляційною базою даних, що забезпечує можливість подальшого ретроспективного аналізу інцидентів, фільтрації записів за типом та ефективного налагодження створених студентами правил міжмашинної взаємодії.

3.4 Практичні випробування та експериментальна перевірка роботи навчального комплексу

Завершальним і найбільш критичним етапом створення будь-якої кіберфізичної системи є проведення комплексу практичних випробувань та експериментальної перевірки працездатності всіх закладених інженерних рішень. Для розробленого навчального стенду з Інтернету речей процес тестування мав подвійну мету. З одного боку, необхідно було підтвердити технічну надійність програмно-апаратного комплексу, відмовостійкість серверного ядра на базі мікрофреймворку Flask та коректність роботи алгоритмів міжмашинної взаємодії. З іншого боку, випробування мали довести високу дидактичну ефективність розробленого рішення, підтвердивши, що стенд дозволяє студентам наочно спостерігати фізичні процеси, моделювати аварійні ситуації та зручно аналізувати результати своїх конфігураційних дій через графічний веб-інтерфейс. У зв'язку з цим програма експериментальних досліджень була побудована за принципом поступового ускладнення завдань, починаючи від ізольованого тестування окремих програмних модулів і закінчуючи комплексним навантажувальним тестуванням усієї інфраструктури в умовах імітації апаратних збоїв.

На початковому етапі експериментальних досліджень було проведено верифікацію роботи прикладного програмного інтерфейсу серверної частини, який відповідає за прийом телеметрії. Оскільки в реальних умовах навчальної лабораторії до сервера можуть одночасно підключатися десятки мікроконтролерних макетів, обчислювальне ядро мовою Python повинно ефективно обробляти високочастотні потоки мережевих запитів. Для перевірки пропускної здатності було використано спеціалізоване програмне забезпечення для генерації синтетичного HTTP-трафіку. Експеримент полягав у паралельному надсиланні кількох сотень запитів на оновлення значень віртуальних датчиків за частки секунди. Результати випробувань показали, що завдяки використанню

резидентних словників в оперативній пам'яті для кешування станів цифрових двійників, сервер успішно обробляє масові оновлення без виникнення помилок блокування бази даних. Асинхронна архітектура дозволила уникнути втрати пакетів, гарантуючи, що кожна зміна показника температури чи рівня задимлення буде миттєво зафіксована системою та передана до логічного рушія.

Наступним кроком стала перевірка найскладнішого інтелектуального компонента системи – модуля міжмашинної взаємодії та реалізованої в ньому математичної моделі гістерезису. Для проведення цього експерименту в інженерному режимі веб-пульта було спеціально створено конфігураційне правило, яке пов'язувало аналоговий датчик рівня вуглекислого газу з виконавчим механізмом системи вентиляції. Порогове значення спрацювання було встановлено на рівні дев'ятисот частинок на мільйон. Випробування проводилися шляхом ручного імітування поступового зростання концентрації газу за допомогою інтерактивного повзунка у веб-інтерфейсі. Під час плавної зміни показника від восьмисот до вісімсот дев'яноста дев'яти одиниць система залишалася в стані спокою. У момент перетину встановленої межі логічний рушій миттєво згенерував керуючу команду, і цифрова індикація вентилятора змінила стан на активний. Найважливіша частина експерименту стосувалася перевірки алгоритму захисту від брязкоту контактів. При імітації незначних коливань показника безпосередньо навколо порогового значення програмне ядро успішно ігнорувало ці флуктуації, утримуючи вентилятор в увімкненому стані доти, поки рівень газу не опустився значно нижче заданої межі гістерезису.

Для наочної візуалізації методології проведення комплексного тестування працездатності алгоритмів автоматизації під час проведення лабораторних робіт було розроблено відповідну блок-схему експерименту (рис. 3.9).

Процес експериментального тестування, відображений на наведеній блок-схемі, повністю відповідає методичним вказівкам, які будуть надаватися студентам для перевірки власних проєктних рішень. Важливою складовою цього

процесу є не лише візуальний контроль індикаторів на екрані, але й обов'язкова верифікація подій через системний журнал.

Це дозволяє студентам аналізувати часові затримки між виникненням критичної події на сенсорі та ініціацією відповідної реакції актуатором, що критично для розуміння природи реального часу в кіберфізичних системах. У межах експерименту дослідники мають можливість варіювати порогові значення умов автоматизації, спостерігаючи за стабільністю роботи алгоритмів при перехідних процесах у мережі.

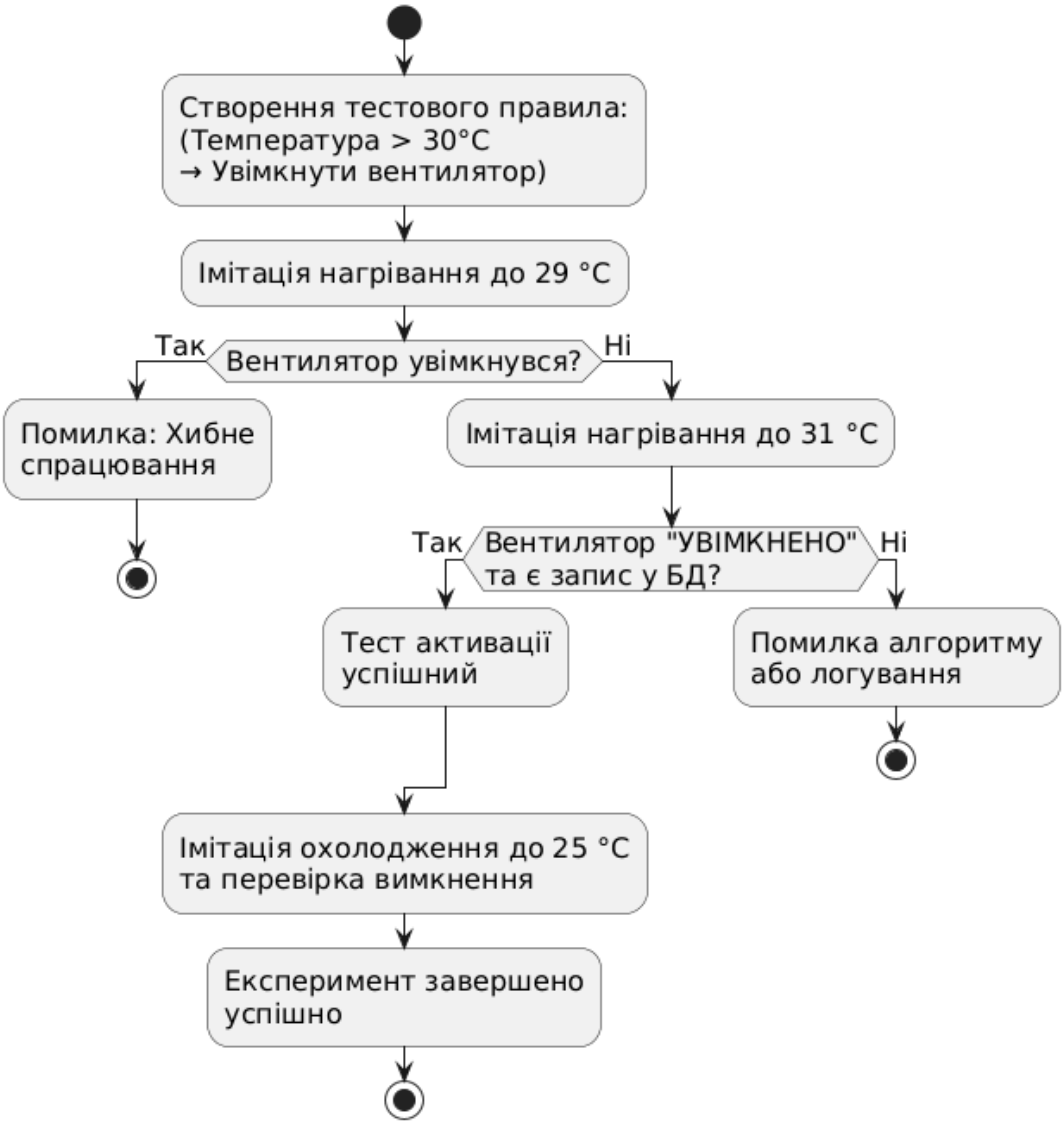


Рисунок 3.9 - Блок-схема проведення експериментального тестування логіки M2M

Під час випробувань було доведено, що кожне автономне рішення серверного ядра автоматично генерує детальний текстовий запис із точною часовою міткою. Журнал подій дозволив інженерам-дослідникам ретроспективно проаналізувати ланцюжок подій (рис 3.10), під час складних макро-сценаріїв. Наприклад, під час тестування макросу пожежної тривоги журнал чітко зафіксував послідовність дій: спочатку штучну модифікацію показників датчиків диму та температури, потім каскадне спрацювання локальних правил безпеки, і нарешті активацію сирени та розблокування електромагнітних замків дверей.

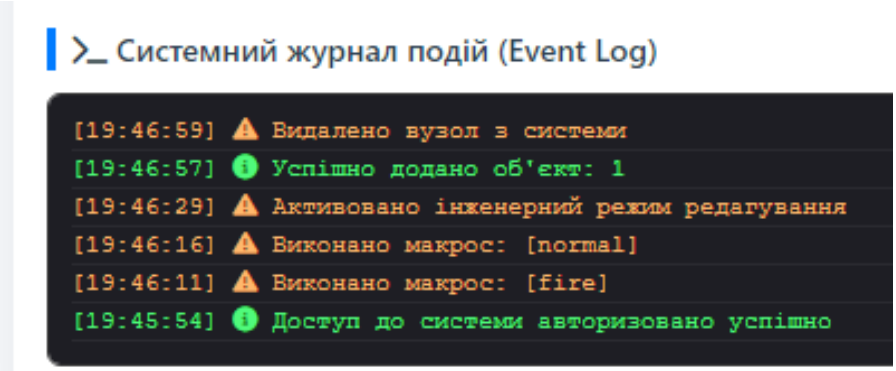


Рисунок 3.10 - Результат ланцюжка подій

Окремий масив експериментів було присвячено дослідженню відмовостійкості системи та її поведінки в умовах нестабільного мережевого з'єднання, що є надзвичайно актуальною проблемою для бездротових сенсорних мереж. Для імітації таких умов у програмному забезпеченні клієнтської частини було передбачено спеціальні інструменти примусового розриву з'єднання віртуальних датчиків. Під час тестування оператор ініціював збій зв'язку для критично важливого вузла, наприклад, пожежного сповіщувача. Програмне ядро системи миттєво відреагувало на припинення надходження телеметрії зміною статусу цифрового двійника. На веб-інтерфейсі картка відповідного датчика змінила своє візуальне оформлення: індикатор серцебиття зупинився, з'явився червоний значок обриву лінії, а елементи ручного керування були примусово

заблоковані скриптом, щоб запобігти відправці команд в нікуди. Після програмного відновлення з'єднання сервер автоматично відновив потік даних, актуалізував графіки та перевірив стани всіх залежних виконавчих пристроїв без необхідності перезавантаження всього комплексу.

Заключним етапом експериментальної перевірки стало тестування підсистеми довготривалого збереження інформації та модуля побудови аналітичних трендів. Випробування полягали у тривалому прогоні системи протягом кількох годин із постійною генерацією випадкових коливань температури та вологості. Метою було переконатися, що база даних SQLite коректно індексує нові записи і не зазнає деградації швидкодії при збільшенні обсягу файлу. Для перевірки результатів було використано модальне вікно графічної аналітики у веб-інтерфейсі. Виклик графіка для обраного датчика відбувався практично миттєво, що підтвердило високу ефективність налаштованих SQL-запитів. Отримані графіки чітко відображали змодельовані раніше флуктуації, а накладена лінія уставки дозволила візуально підтвердити, що алгоритм міжмашинної взаємодії дійсно спрацьовував саме в моменти перетину кривою заданого математичного порогу. Усі ці практичні результати переконливо доводять, що спроєктована та реалізована кіберфізична система функціонує виключно надійно, повністю відповідає поставленому технічному завданню та є потужним інноваційним інструментом для підготовки майбутніх фахівців з комп'ютерної інженерії.

Розроблений апаратно-програмний комплекс, незважаючи на своє першочергове навчально-демонстраційне призначення, володіє архітектурою та функціональними можливостями, які дозволяють екстраполювати його використання на широкий спектр реальних інженерних задач. Створена кіберфізична система не є просто макетом для перевірки теоретичних гіпотез у межах аудиторних занять. Завдяки впровадженню промислових стандартів обміну даними, інтеграції надійного реляційного сховища телеметрії та використанню асинхронного серверного ядра на базі мови програмування

Python, стенд перетворюється на масштабовану платформу. Ця платформа здатна ефективно працювати не лише в тепличних лабораторних умовах, але й слугувати прототипом для розгортання повноцінних систем диспетчеризації на об'єктах критичної інфраструктури, у комерційній нерухомості та в житловому секторі.

Одним із найбільш перспективних і соціально значущих напрямків практичного застосування розробленої системи є її інтеграція у концепцію розумного будинку та автоматизації житлових приміщень. Сучасні тенденції енергозбереження та підвищення рівня комфорту вимагають переходу від розрізнених побутових приладів до єдиної екосистеми управління. Створений комплекс дозволяє реалізувати повноцінний клімат-контроль, об'єднуючи розрізнені датчики температури та вологості з виконавчими механізмами систем опалення, вентиляції та кондиціонування. Наприклад, алгоритм міжмашинної взаємодії може бути налаштований таким чином, щоб пристрій автоматично аналізував показання датчиків якості повітря. При перевищенні допустимої концентрації вуглекислого газу серверне ядро самостійно, без втручання мешканців, ініціює відкриття сервоприводів віконних фрамуг або вмикає примусову припливно-витяжну вентиляцію. Аналогічним чином реалізується керування освітленням: система здатна аналізувати рівень природної освітленості за допомогою фоторезисторів та фіксувати присутність людей через інфрачервоні детектори руху. На основі цих даних оркестратор динамічно регулює яскравість світлодіодних стрічок або вмикає розумні лампи лише в тих зонах, де це фактично необхідно, що призводить до суттєвої економії електричної енергії без жодного зниження рівня комфорту користувачів.

Не менш важливою сферою застосування є забезпечення комплексної безпеки об'єктів. Розроблений стенд містить усі необхідні апаратні та програмні інструменти для створення надійної охоронно-пожежної сигналізації. Завдяки наявності дискретних датчиків відкриття дверей та детекторів руху, система здатна функціонувати в режимі охорони периметра. При несанкціонованому

проникненні на об'єкт логічний рушій миттєво генерує тривожну подію, яка не лише фіксується в системному журналі, але й викликає каскад реакцій: вмикається потужна акустична сирена, активується тривожна світлова індикація, а відповідне попереджувальне повідомлення миттєво виводиться на екран диспетчерського пульта. У розрізі пожежної безпеки інтеграція оптичних датчиків задимлення та сенсорів чадного газу дозволяє системі розпізнати займання на ранніх стадіях. У такому випадку глобальний макрос безпеки, передбачений у програмному коді, здатен автоматично розблокувати електромагнітні замки на евакуаційних виходах, знеструмити вентиляційні канали для запобігання поширенню диму та увімкнути аварійне освітлення.

Промислове застосування розробленої архітектури відкриває можливості для оптимізації технологічних процесів та предиктивного обслуговування обладнання. Модульна будова мікроконтролерного рівня дозволяє замість побутових датчиків підключати специфічні промислові сенсори, наприклад, датчики вібрації, тиску або сили струму. Розміщення таких сенсорів на вузлах промислових верстатів чи насосних станцій дозволяє системі безперервно збирати телеметрію та формувати історичні тренди в базі даних SQLite. Асинхронний сервер мовою Python може бути розширений додатковими модулями статистичного аналізу. Ці модулі здатні виявляти аномальні відхилення у графіках вібрації чи енергоспоживання, які часто передують фізичній поломці механізму. Такий підхід дозволяє диспетчеру своєчасно зупинити обладнання для проведення профілактичного ремонту, уникаючи катастрофічних аварій та тривалих простоїв виробничої лінії.

Окрім описаних сценаріїв, комплекс має величезний потенціал у сфері агропромислового сектору, зокрема для автоматизації тепличних господарств. Кіберфізична система здатна взяти на себе повний контроль за мікрокліматом та режимом поливу рослин. Використовуючи датчики вологості ґрунту, сенсори інтенсивності сонячного випромінювання та термометри, серверне ядро формує динамічні правила зрошення. На відміну від жорстко запрограмованих таймерів,

створена подієво-орієнтована система вмикатиме електромагнітні клапани подачі води виключно тоді, коли показники вологості ґрунту впадуть нижче критичного порогу, розрахованого з урахуванням математичного гістерезису. Одночасно з цим система може керувати сервоприводами для відкриття вентиляційних люків у разі надмірного перегріву повітря всередині теплиці або вмикати додаткові фітолампи при зниженні рівня природного освітлення у похмурі дні.

Усі наведені приклади практичного застосування наочно доводять високу адаптивність та універсальність розробленого програмно-апаратного комплексу. Створена кіберфізична система є не просто абстрактною навчальною моделлю, а повноцінним інженерним інструментом. Вона дозволяє вирішувати реальні завдання автоматизації, моніторингу та диспетчеризації у найрізноманітніших сферах життєдіяльності, що підтверджує практичну цінність та актуальність виконаної кваліфікаційної роботи.

3.5 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи було повністю та успішно реалізовано програмно-апаратну частину навчально-демонстраційного комплексу з Інтернету речей, що дозволило перетворити теоретичні архітектурні концепції на дієвий інженерний інструмент. У ході виконання практичного етапу розробки було спроєктовано та написано сирцевий код серверного ядра кіберфізичної системи з використанням мови програмування Python та мікрофреймворку Flask. Вибір цих програмних засобів дозволив створити гнучке, модульне та легкомасштабоване рішення, яке повністю позбавлене зайвих монолітних абстракцій і є максимально прозорим для вивчення студентами в межах лабораторного практикуму. Важливим досягненням на рівні бекенду стало впровадження гібридної моделі збереження інформації, де поточний стан пристроїв безперервно кешується в оперативній пам'яті сервера у

вигляді оптимізованих словників, а довготривалі часові ряди телеметрії фіксуються у вбудованій реляційній базі даних SQLite, що дозволило мінімізувати затримки обробки даних та оптимізувати навантаження на дискову підсистему.

У межах розділу було детально спроєктовано та впроваджено інтерактивний клієнтський людино-машинний інтерфейс, який виконує функцію центрального інженерного пульта керування. Використання класичного веб-стеку технологій дозволило досягти повної кросплатформності інтерфейсу, роблячи його доступним на будь-яких пристроях оператора.

Завершальним етапом практичної роботи стало проведення всебічних експериментальних випробувань створеного комплексу. Навантажувальне тестування підтвердило здатність асинхронного сервера Flask стабільно обробляти високочастотні потоки мережевих запитів без деградації швидкодії та втрати транзакцій. Серія успішних експериментів з імітації обриву бездротових ліній Wi-Fi довела стійкість клієнтських скриптів і серверної логіки до апаратних збоїв інфраструктури, що супроводжувалося безпомилковою фіксацією всіх системних подій у хронологічному журналі подій подій. Також було проведено детальний аналіз практичних аспектів застосування комплексу, який підтвердив, що розроблена архітектура може успішно слугувати функціональним прототипом для розгортання реальних систем диспетчеризації в житловій автоматизації, охоронно-пожежних комплексах, промисловому моніторингу та агротехнічному секторі. Таким чином, практичні результати третього розділу повністю підтвердили працездатність, надійність та високу дидактичну цінність розробленого програмно-технічного засобу, що дозволяє перейти до фінального узагальнення висновків кваліфікаційної роботи.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було успішно спроєктовано, математично обґрунтовано та практично реалізовано програмно-апаратний комплекс навчального стенду з Інтернету речей, який призначений для поглибленої підготовки майбутніх фахівців у галузі комп'ютерної інженерії. Доведено, що використання сучасної подієво-орієнтованої архітектури у поєднанні з мікросервісним підходом до побудови серверного програмного забезпечення на базі інтерпретованої мови Python дозволяє створити надзвичайно надійну, масштабовану та дидактично цінну кіберфізичну систему. Розроблений комплекс повністю вирішує актуальну проблему відсутності гнучкого лабораторного інструментарію, надаючи здобувачам вищої освіти унікальну можливість наочно досліджувати повний життєвий цикл інформації: від моменту фізичного вимірювання аналогових величин навколишнього середовища до їхньої комплексної програмної обробки, збереження в реляційних базах даних та графічної візуалізації на центральному інженерному пульті оператора.

У першому розділі проведено аналіз ретроспективної еволюції технологій Інтернету речей та концептуальних підходів до побудови навчальних лабораторних стендів для вищих навчальних закладів. Досліджено історичний та технологічний перехід від вивчення абсолютно ізольованих мікроконтролерних систем минулого покоління до розгортання повноцінних кіберфізичних просторів, що нерозривно об'єднують глобальні інформаційні та локальні операційні технології. Встановлено, що класичні методи підготовки інженерів вимагають докорінної модернізації через впровадження комплексних рішень на базі високорівневих мов програмування. Це дозволяє ефективно змістити освітній фокус із рутинного низькорівневого апаратного кодування на вирішення складних архітектурних завдань, оркестрацію великих мережевих потоків та

проєктування інтелектуальних систем безперервної міжмашинної взаємодії в гетерогенних мережах.

У другому розділі проведено глибоке теоретичне проєктування та системний аналіз усіх ключових підсистем створюваного апаратно-програмного комплексу. Обґрунтовано інженерну доцільність застосування трирівневої архітектурної моделі з використанням комунікаційної топології типу зірка для забезпечення максимальної відмовостійкості всієї навчальної інфраструктури лабораторії. Розроблено та строго математично описано алгоритми автономної міжмашинної взаємодії, де для усунення класичної проблеми високочастотного брязкоту контактів виконавчих механізмів було надзвичайно успішно імплементовано програмну модель алгоритмічного гістерезису. Також здійснено детальне проєктування структури реляційної бази даних та науково обґрунтовано необхідність застосування патерну об'єктно-реляційного відображення для гарантування абсолютної транзакційної цілісності телеметричної інформації і захисту розробленої системи від можливих кібернетичних втручань чи помилок на рівні виконання структурованих запитів.

У третьому розділі виконано безпосередню практичну програмно-апаратну реалізацію кіберфізичної системи та проведено комплекс її жорстких експериментальних випробувань. Створено інтерактивний клієнтський людино-машинний інтерфейс з надійним багаторівневим контролем доступу, вбудованим модулем динамічної побудови графічних трендів та спеціалізованим візуального конструювання логічних правил автоматизації. Практичні випробування повністю та беззаперечно підтвердили високу стабільність розробленої системи при обробці інтенсивних потоків телеметрії та імітації апаратних збоїв. Крім того, розгляд перспективних прикладів застосування довів, що створений навчальний комплекс може максимально ефективно використовуватися як повноцінний діючий прототип для розгортання промислових систем диспетчеризації, клімат-контролю та комплексної безпеки на реальних об'єктах критичної чи цивільної інфраструктури.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Юскович-Жуковська В. І., Кот В. В., Лотюк Ю. Г. Віддалений лабораторний стенд для навчання системам Інтернету речей. *Телекомунікаційні та інформаційні технології*. 2025. № 4. С. 48–53.

2. Каращук Я. Ф., Гришко І. А. Лабораторний стенд для дистанційного навчання. *Інновації молоді в машинобудуванні*. 2020. № 2. С. 287–290.

3. Malokhvii E., Molchanov H. Дослідження протоколів передачі даних в умовах Інтернету речей. *Системи управління, навігації та зв'язку. Збірник наукових праць*. 2022. Т. 1, № 67. С. 66–74.

4. Удинська А. Р. *Розробка стенду з аналізу інформаційної безпеки IoT-мереж* : кваліфікаційна робота.

5. Плахтєєв А. П., Бабешко Є. В., Ткаченко В. А., Здоровець Ю. В. *Архітектури та розроблення систем Інтернету/Вебу речей на основі вбудованих платформ*, 2019.

6. Вишневська Д. В. *Технологія Інтернету речей IT-системи «Розумний будинок»* : кваліфікаційна робота, 2021.

7. Фечан А. В., Кушнірук Д. П. Оцінювання ефективності застосування CRDT-технологій у системах моніторингу інтернету речей. *Scientific Bulletin of UNFU*. 2024. Vol. 34, № 4. P. 86–92.

8. Цирульник С. М. Програмно-апаратний комплекс «Arduino Learner Kit». *Електронне наукове фахове видання «Відкрите освітнє е-середовище сучасного університету»*. 2021. № 10. С. 231–240.

9. Грабар І. Г., Іванченко В. М., Ломакін В. О. Програмно-апаратний комплекс для експериментального дослідження полів прискорень. *Вісник ЖДТУ. Серія: Технічні науки*. 2009. Т. 1, № 48. С. 41–47.

10. Polinovskyi V. V., Gerasymenko V. A. Уніфікований програмно-апаратний комплекс автоматизації процесу створення та накопичення сучасних

навчальних матеріалів. *Journal of Information Technologies in Education (ITE)*. 2009. № 4. P. 250–256.

11. Попович К. В., Ходзінська Н. Ю., Чеснік Н. М. Формування математичних компетентностей у майбутніх фахівців з комп'ютерної інженерії. *Концептуальні шляхи розвитку науки та освіти* : матеріали VIII Міжнар. наук.-практ. конф., м. Львів, 9–10 червня 2023 р. Львів : Львівський науковий форум, 2023. С. 90.

12. Zhaliy R. Застосування здоров'язбережувальних інноваційних технологій в освітньому процесі підготовки майбутніх фахівців з комп'ютерної інженерії. *Українська професійна освіта*. 2020. № 7. С. 88–96.

13. Стрюк К. М. Зміст професійної компетентності майбутніх фахівців із комп'ютерної інженерії. *Збірник наукових праць «Педагогічні науки»*. 2016. Т. 2, № 73. С. 118–122.

14. Python W. Python. *Python releases for windows*. 2021. № 24.

15. Van Rossum G., Drake F. L. Jr. *Python tutorial*. Vol. 620. Amsterdam : Centrum voor Wiskunde en Informatica, 1995.

16. Downey A. *Think python*. O'Reilly Media, Inc., 2012.

17. Van Rossum G. *Python 0.*, 1991.

18. Ari N., Ustazhanov M. Matplotlib in python. *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*. IEEE, 2014.

19. Програмне забезпечення мікрокомп'ютерної системи захисту IoT / В. А. Басистий та ін. *Тези доповідей*. 2025. С. 24.

20. Волошанський І. О. *IoT система сервісів комплектації дронів* : кваліфікаційна робота, 2024.

21. Шишкіна В. О. *Аналіз та вдосконалення Інформаційного чат-боту ТНТУ засобами Python та Aiogram* : магістерська робота. Тернопіль, 2023.

22. Лапшин О. О. *Розробка програмного забезпечення для аутентифікації пристроїв інтернету речей на основі методів машинного навчання* : кваліфікаційна робота, 2023.

23. Castro E. *HTML for the world wide web*. Peachpit Press, 2003.

24. Musciano C., Kennedy B. *HTML & XHTML: The definitive guide*. O'Reilly Media, Inc., 2002.

25. Raggett D., Le Hors A., Jacobs I. HTML 4.01 Specification. *IETF HTML WG*. 1997.

26. Musciano C., Kennedy B. *HTML & XHTML*. Anaya Multimedia, 2001.

27. Goodman D. *Dynamic HTML: The definitive reference: A comprehensive resource for HTML, CSS, DOM & JavaScript*. O'Reilly Media, Inc., 2002.

28. Levering R., Cutler M. The portrait of a common HTML web page. *Proceedings of the 2006 ACM symposium on Document engineering*. 2006.

29. Cutler M., Shih Y., Meng W. Using the Structure of HTML Documents to Improve Retrieval. *USENIX Symposium on Internet Technologies and Systems (USITS 97)*. 1997.

30. Research Articles in Simplified HTML: a Web-first format for HTML-based scholarly articles / S. Peroni та ін. *PeerJ Computer Science*. 2017. Vol. 3. P. e132.

31. Enterprise J. *HTML 5 Manual Book*. Elex Media Komputindo, 2015.

32. Jensen S. H., Møller A., Thiemann P. Type analysis for JavaScript. *International Static Analysis Symposium*. Berlin ; Heidelberg : Springer, 2009.

33. Crockford D. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

34. Richards G., Lebresne S., Burg B., Vitek J. An analysis of the dynamic behavior of JavaScript programs. *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2010. P. 1–12.

35. Wirfs-Brock A., Eich B. JavaScript: the first 20 years. *Proceedings of the ACM on Programming Languages*. 2020. Vol. 4, No. HOPL. P. 1–189.

36. Sun K., Ryu S. Analysis of JavaScript programs: Challenges and research trends. *ACM Computing Surveys (CSUR)*. 2017. Vol. 50, No. 4. P. 1–34.

37. Flanagan D. *JavaScript: The definitive guide: Activate your web pages*. O'Reilly Media, Inc., 2011.

38. Toward intelligent machine-to-machine communications in smart grid / Z. M. Fadlullah та ін. *IEEE Communications Magazine*. 2011. Vol. 49, No. 4. P. 60–65.

39. Chen K.-C., Lien S.-Y. Machine-to-machine communications: Technologies and challenges. *Ad Hoc Networks*. 2014. Vol. 18. P. 3–23.

40. Ali A., Shah G. A., Farooq M. O., Ghani U. Technologies and challenges in developing machine-to-machine applications: A survey. *Journal of Network and Computer Applications*. 2017. Vol. 83. P. 124–139.

41. Design an mvc model using python for flask framework development / M. R. Mufid та ін. *2019 International Electronics Symposium (IES)*. IEEE, 2019.

42. Implementation of database using python flask framework / M. Singh та ін. *International Journal of Engineering and Computer Science*. 2019. Vol. 8, No. 12. P. 24890–24893.

43. Vyshnavi V. R., Malik A. Efficient way of web development using python and flask. *Int. J. Recent Res. Asp*. 2019. Vol. 6, No. 2. P. 16–19.

44. Dwyer G., Aggarwal S., Stouffer J. *Flask: building python web services*. Packt Publishing, 2017.

45. Gaspar D., Stouffer J. *Mastering Flask Web Development: Build Enterprise-Grade, Scalable Python Web Applications*. Packt Publishing Ltd, 2018.

46. Introduction to Internet of Things – IOT. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-networks/introduction-to-internet-of-things-iot-set-1/ (дата звернення: 21.05.2026).

47. Architecture of Internet of Things (IoT). *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-networks/architecture-of-internet-of-things-iot/ (дата звернення: 21.05.2026).

48. Arduino Sensor Kit - 37 Sensors. DIYables. URL: https://diyables.io/products/sensor-kit (дата звернення: 21.05.2026).

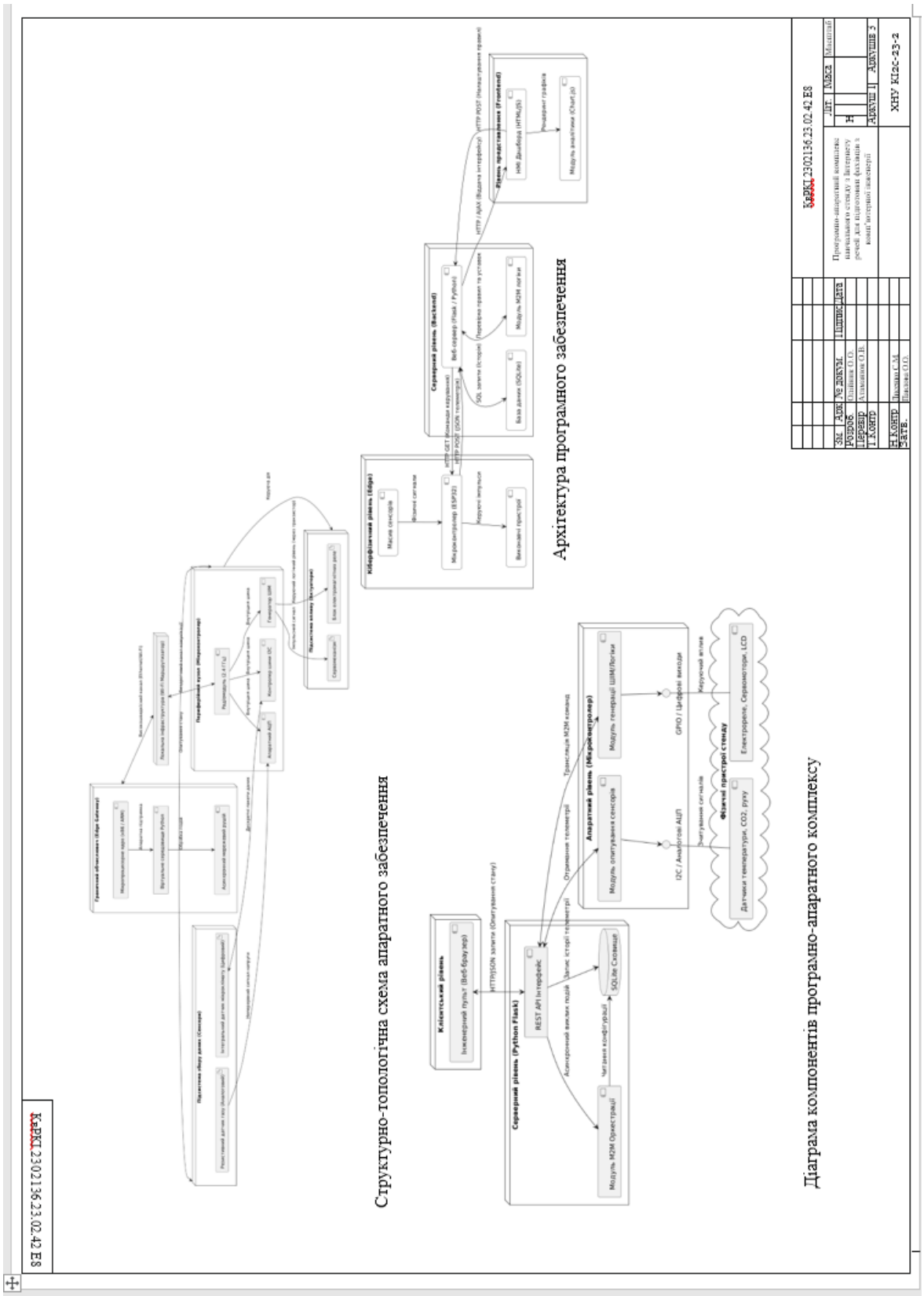
49. Mastering Flask: From Fundamentals to Advanced Concepts. *Medium*. URL: https://medium.com/@sandyjtech/mastering-flask-from-fundamentals-to-advanced-concepts-f1d60ac740d0 (дата звернення: 21.05.2026).

50. Flask Web Development Part 1: Hello World App. *Medium*. URL: https://medium.com/analytics-vidhya/flask-development-part-1-hello-world-app-69cbf9e83abc (дата звернення: 21.05.2026).

ДОДАТОК А

(обов’язковий)

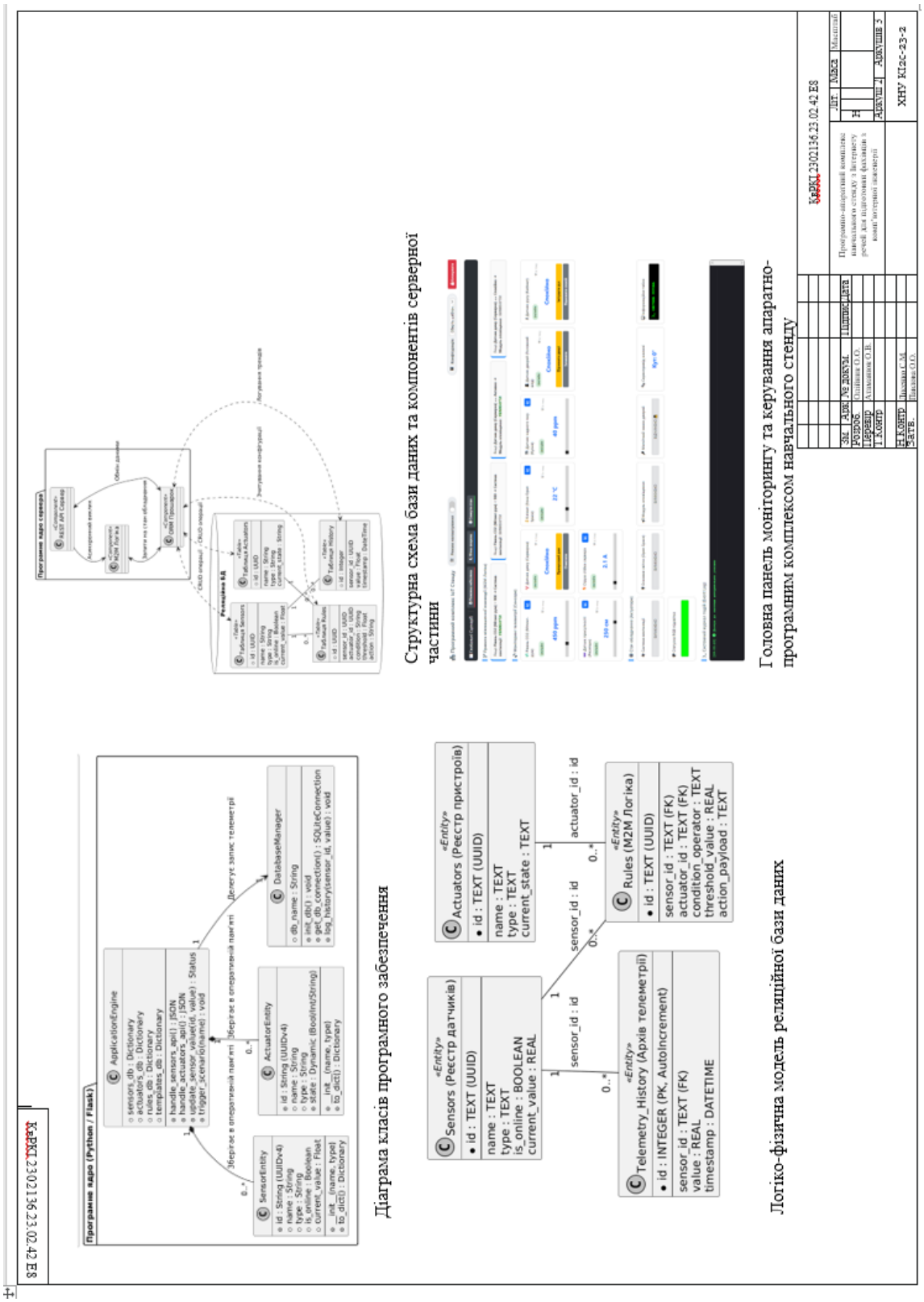
Копія креслення «Апаратна архітектура навчального стенду IoT»



ДОДАТОК Б

(обов’язковий)

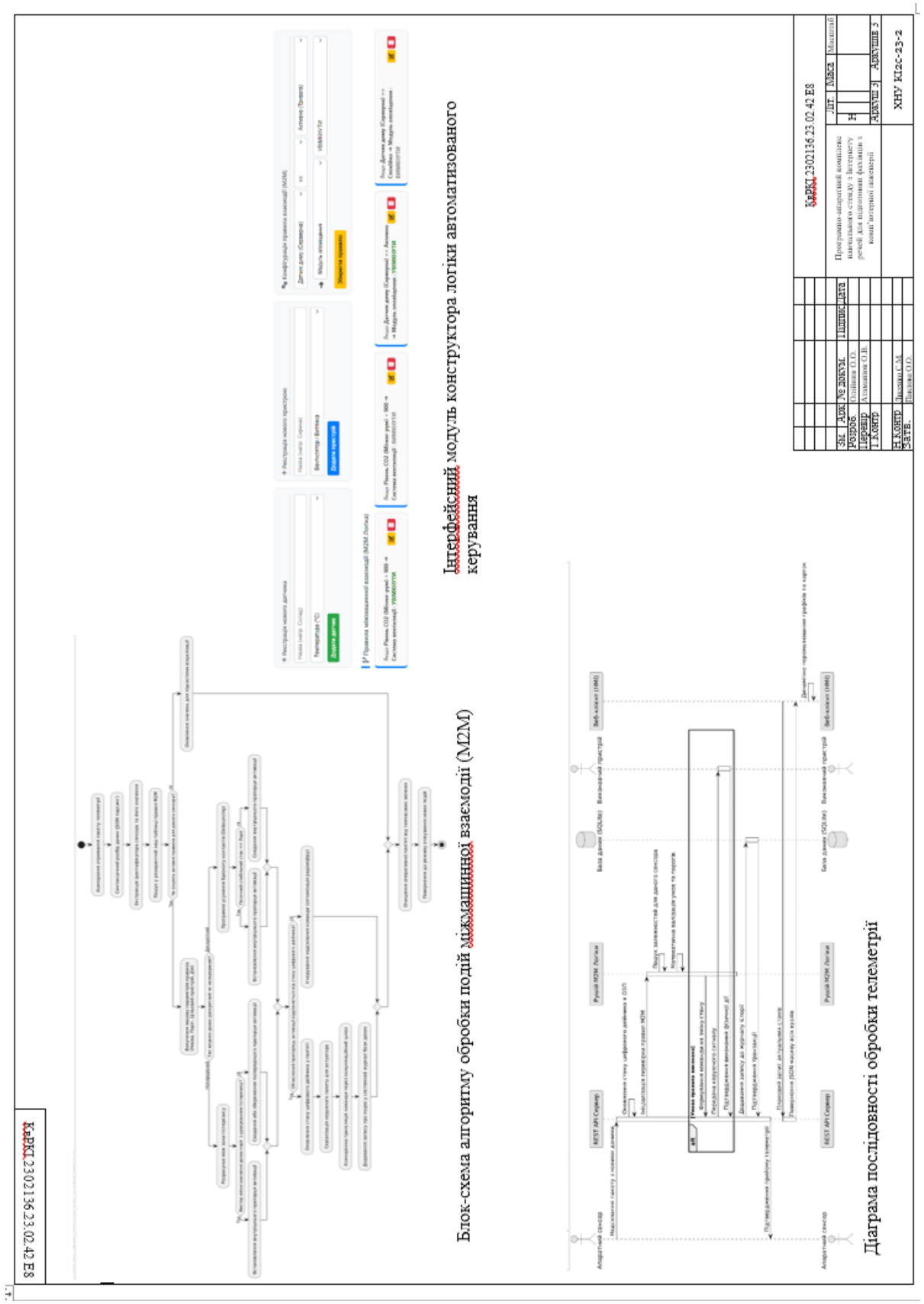
Копія креслення «Програмне забезпечення та серверна інфраструктура»



ДОДАТОК В

(обов’язковий)

Копія креслення «Алгоритмічне забезпечення системи керування»



ДОДАТОК Г

Лістинг програмного коду

app.py

```
from flask import Flask, render_template, request, jsonify
import uuid
import sqlite3
import copy

app = Flask(__name__)
DB_NAME = 'iot_stand.db'

# Сховища в оперативній пам'яті
sensors_db = {}
actuators_db = {}
rules_db = {}
templates_db = {} # Для зберігання шаблонів системних конфігурацій

def init_db():
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS sensor_history
                      (id INTEGER PRIMARY KEY AUTOINCREMENT, sensor_id TEXT NOT NULL,
                      value REAL NOT NULL, timestamp DATETIME DEFAULT (datetime('now', 'localtime')))''')
    conn.commit()
    conn.close()

def get_db_connection():
    conn = sqlite3.connect(DB_NAME)
    conn.row_factory = sqlite3.Row
    return conn

class Sensor:
    def __init__(self, name, sensor_type):
        self.id = str(uuid.uuid4())
        self.name = name
        self.type = sensor_type
        self.is_online = True

        if self.type == 'temperature': self.current_value = 22.0
        elif self.type == 'humidity': self.current_value = 45.0
        elif self.type == 'air_quality': self.current_value = 450.0
        elif self.type == 'gas': self.current_value = 40.0
        elif self.type == 'current': self.current_value = 2.1
        elif self.type == 'ultrasonic': self.current_value = 250.0
```

```
        elif self.type in ['motion', 'smoke', 'door']: self.current_value = 0

    def to_dict(self):
        return {'id': self.id, 'name': self.name, 'type': self.type, 'value': self.current_value, 'is_online': self.is_online}

class Actuator:
    def __init__(self, name, act_type):
        self.id = str(uuid.uuid4())
        self.name = name
        self.type = act_type

        if self.type in ['fan', 'lamp', 'siren', 'lock']: self.state = False
        elif self.type == 'servo': self.state = 0
        elif self.type == 'lcd': self.state = "СИСТЕМА ГОТОВА"
        elif self.type == 'rgb': self.state = "#00ff00"

    def to_dict(self):
        return {'id': self.id, 'name': self.name, 'type': self.type, 'state': self.state}

def seed_data():
    if not sensors_db:
        s_co2 = Sensor("Рівень CO2 (Мітинг-рум)", "air_quality")
        s_smoke = Sensor("Датчик диму (Серверна)", "smoke")
        s_temp = Sensor("Клімат (Зона Open Space)", "temperature")
        s_gas = Sensor("Датчик чадного газу (Кухня)", "gas")
        s_door = Sensor("Датчик дверей (Головний вхід)", "door")
        s_mot = Sensor("Датчик руху (Кабінет)", "motion")
        s_dist = Sensor("Датчик присутності (Ресепшн)", "ultrasonic")
        s_curr = Sensor("Струм стійки сервера", "current")

        for s in [s_co2, s_smoke, s_temp, s_gas, s_door, s_mot, s_dist, s_curr]: sensors_db[s.id] = s

        a_fan = Actuator("Система вентиляції", "fan")
        a_lamp = Actuator("Основне світло (Open Space)", "lamp")
        a_siren = Actuator("Модуль оповіщення", "siren")
        a_lock = Actuator("Магнітний замок дверей", "lock")
        a_servo = Actuator("Сервопривід жалюзі", "servo")
        a_lcd = Actuator("Інформаційне табло", "lcd")
        a_rgb = Actuator("Статусна RGB підсвітка", "rgb")

        for a in [a_fan, a_lamp, a_siren, a_lock, a_servo, a_lcd, a_rgb]: actuators_db[a.id] = a

        rules_db["r1"] = {'id': 'r1', 'sensor_id': s_co2.id, 'condition': '>', 'threshold': 900.0, 'actuator_id': a_fan.id, 'action': True}
        rules_db["r2"] = {'id': 'r2', 'sensor_id': s_co2.id, 'condition': '<', 'threshold': 900.0, 'actuator_id': a_fan.id, 'action': False}
        rules_db["r3"] = {'id': 'r3', 'sensor_id': s_smoke.id, 'condition': '==', 'threshold': 1.0, 'actuator_id': a_siren.id, 'action': True}
        rules_db["r4"] = {'id': 'r4', 'sensor_id': s_smoke.id, 'condition': '==', 'threshold': 0.0, 'actuator_id': a_siren.id, 'action': False}
```

```
init_db()
seed_data()

def check_rules(sensor_id, current_value):
    for rule in rules_db.values():
        if rule['sensor_id'] == sensor_id:
            cond = False
            if rule['condition'] == '>' and current_value > rule['threshold']: cond = True
            elif rule['condition'] == '<' and current_value < rule['threshold']: cond = True
            elif rule['condition'] == '==' and current_value == rule['threshold']: cond = True
            if cond and rule['actuator_id'] in actuators_db: actuators_db[rule['actuator_id']].state = rule['action']

@app.route('/')
def index(): return render_template('index.html')

@app.route('/api/sensors', methods=['GET', 'POST'])
def handle_sensors():
    if request.method == 'POST':
        data = request.json
        if not data.get('name') or not data.get('type'):
            return jsonify({"error": "Назва та тип сенсора обов'язкові для заповнення"}), 400
        new_s = Sensor(name=data['name'], sensor_type=data['type'])
        sensors_db[new_s.id] = new_s
        return jsonify(new_s.to_dict()), 201

    conn = get_db_connection()
    cursor = conn.cursor()
     cursor.execute('DELETE FROM sensor_history WHERE id NOT IN (SELECT id FROM sensor_history ORDER BY id
DESC LIMIT 1000)')
    s_list = []
    for s in sensors_db.values():
        if s.is_online: cursor.execute('INSERT INTO sensor_history (sensor_id, value) VALUES (?, ?)', (s.id, s.current_value))
        s_list.append(s.to_dict())
    conn.commit()
    conn.close()
    return jsonify(s_list)

@app.route('/api/sensors/<s_id>/value', methods=['POST'])
def update_sensor_value(s_id):
    if s_id in sensors_db:
        if not sensors_db[s_id].is_online: return jsonify({"error": "Пристрій офлайн"}), 403
        try:
            new_val = float(request.json['value'])
            if sensors_db[s_id].type in ['motion', 'smoke', 'door'] and new_val not in [0.0, 1.0]:
                return jsonify({"error": "Невалідні дані для дискретного сенсора"}), 400
```

```
            sensors_db[s_id].current_value = new_val
            check_rules(s_id, new_val)
            return jsonify({"status": "success"})
        except ValueError:
            return jsonify({"error": "Значення має бути числом"}), 400
    return jsonify({"error": "Сенсор не знайдено"}), 404

@app.route('/api/sensors/<s_id>/toggle_network', methods=['POST'])
def toggle_sensor_network(s_id):
    if s_id in sensors_db:
        sensors_db[s_id].is_online = not sensors_db[s_id].is_online
        return jsonify({"status": "success"})
    return jsonify({"error": "Сенсор не знайдено"}), 404

@app.route('/api/sensors/<s_id>', methods=['DELETE'])
def delete_sensor(s_id):
    if s_id in sensors_db: del sensors_db[s_id]
    return jsonify({"status": "success"})

@app.route('/api/actuators', methods=['GET', 'POST'])
def handle_actuators():
    if request.method == 'POST':
        data = request.json
        if not data.get('name') or not data.get('type'):
            return jsonify({"error": "Назва та тип виконавчого пристрою обов'язкові"}), 400
        new_a = Actuator(name=data['name'], act_type=data['type'])
        actuators_db[new_a.id] = new_a
        return jsonify(new_a.to_dict()), 201
    return jsonify([a.to_dict() for a in actuators_db.values()])

@app.route('/api/actuators/<a_id>', methods=['DELETE'])
def delete_actuator(a_id):
    if a_id in actuators_db: del actuators_db[a_id]
    return jsonify({"status": "success"})

@app.route('/api/rules', methods=['GET', 'POST'])
def handle_rules():
    if request.method == 'POST':
        data = request.json
        sensor_id = data.get('sensor_id')
        actuator_id = data.get('actuator_id')
        condition = data.get('condition')
        threshold = data.get('threshold')
        action_val = data.get('action')
        rule_id = data.get('id')
```

```
        if not sensor_id or not actuator_id or not condition or threshold is None or action_val is None:
            return jsonify({"error": "Усі поля правила мають бути заповнені"}), 400
        if sensor_id not in sensors_db or actuator_id not in actuators_db:
            return jsonify({"error": "Обраний сенсор або пристрій не існує"}), 400

        if str(action_val).lower() == 'true': action_val = True
        elif str(action_val).lower() == 'false': action_val = False

        if not rule_id:
            rule_id = str(uuid.uuid4())

        rules_db[rule_id] = {
            'id': rule_id, 'sensor_id': sensor_id, 'condition': condition,
            'threshold': float(threshold), 'actuator_id': actuator_id, 'action': action_val
        }
        return jsonify({"status": "success", "id": rule_id})
    return jsonify(list(rules_db.values()))

@app.route('/api/rules/<r_id>', methods=['DELETE'])
def delete_rule(r_id):
    if r_id in rules_db: del rules_db[r_id]
    return jsonify({"status": "success"})

# --- СИСТЕМА ШАБЛОНІВ КОНФІГУРАЦІЙ ---
@app.route('/api/templates', methods=['GET'])
def get_templates():
    return jsonify(list(templates_db.keys()))

@app.route('/api/templates/save', methods=['POST'])
def save_template():
    data = request.json
    name = data.get('name')
    if not name or name.strip() == "":
        return jsonify({"error": "Назва шаблону не може бути порожньою"}), 400

    templates_db[name] = {
        "sensors": copy.deepcopy(sensors_db),
        "actuators": copy.deepcopy(actuators_db),
        "rules": copy.deepcopy(rules_db)
    }
    return jsonify({"status": "success"})

@app.route('/api/templates/load', methods=['POST'])
def load_template():
    data = request.json
    name = data.get('name')
```

```
    if name not in templates_db:
        return jsonify({"error": "Шаблон не знайдено"}), 404

    global sensors_db, actuators_db, rules_db
    sensors_db = copy.deepcopy(templates_db[name]["sensors"])
    actuators_db = copy.deepcopy(templates_db[name]["actuators"])
    rules_db = copy.deepcopy(templates_db[name]["rules"])
    return jsonify({"status": "success"})

@app.route('/api/templates/delete', methods=['POST'])
def delete_template():
    data = request.json
    name = data.get('name')
    if name in templates_db:
        del templates_db[name]
        return jsonify({"status": "success"})
    return jsonify({"error": "Шаблон не знайдено"}), 404

@app.route('/api/scenarios/<name>', methods=['POST'])
def trigger_scenario(name):
    if name == 'fire':
        for s in sensors_db.values():
            if s.type == 'smoke': s.current_value = 1
            elif s.type == 'temperature': s.current_value = 58.0
        for a in actuators_db.values():
            if a.type == 'lock': a.state = False
            if a.type == 'siren': a.state = True
            if a.type == 'fan': a.state = False
            if a.type == 'lcd': a.state = "⚠ ПОЖЕЖНА ТРИВОГА! ЕВАКУАЦІЯ!"
            if a.type == 'rgb': a.state = "#ff0000"
    elif name == 'night':
        for s in sensors_db.values():
            if s.type in ['smoke', 'motion', 'door', 'gas']: s.current_value = 0
        for a in actuators_db.values():
            if a.type == 'lock': a.state = True
            if a.type == 'lamp': a.state = False
            if a.type == 'fan': a.state = False
            if a.type == 'siren': a.state = False
            if a.type == 'lcd': a.state = "🔒 ОБ'ЄКТ ПІД ОХОРОНОЮ"
            if a.type == 'rgb': a.state = "#0000ff"
    elif name == 'normal':
        for s in sensors_db.values():
            if s.type == 'temperature': s.current_value = 22.0
            elif s.type == 'air_quality': s.current_value = 450.0
            elif s.type in ['smoke', 'motion', 'door', 'gas']: s.current_value = 0
        for a in actuators_db.values():
```

```
                if a.type == 'lock': a.state = False
                if a.type == 'lcd': a.state = "🏢 ОФІС ПРАЦЮЄ В НОРМАЛЬНОМУ РЕЖИМІ"
                if a.type == 'rgb': a.state = "#00ff00"
                if a.type == 'siren': a.state = False
                if a.type == 'fan': a.state = False
                if a.type == 'lamp': a.state = True

            for s in sensors_db.values(): check_rules(s.id, s.current_value)
            return jsonify({"status": "success"})

        @app.route('/api/history/<s_id>', methods=['GET'])
        def get_history(s_id):
            conn = get_db_connection()
            rows = conn.execute("SELECT value, strftime('%H:%M:%S', timestamp) as time FROM sensor_history WHERE sensor_id
= ? ORDER BY id DESC LIMIT 25", (s_id,)).fetchall()
            conn.close()
            threshold = None
            for r in rules_db.values():
                if r['sensor_id'] == s_id and r['condition'] in ['>', '<']:
                    threshold = r['threshold']
                    break
            return jsonify({"points": [{"time": row["time"], "value": row["value"]} for row in reversed(rows)], "threshold": threshold})

        if __name__ == '__main__':
            app.run(debug=True, host='0.0.0.0', port=5000)
```

index.html

```
        <!DOCTYPE html>
        <html lang="uk">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Центральний інженерний пульт IoT</title>

            <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
            <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css">

            <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
        </head>
        <body>

        <div id="lockScreen">
            <div class="keypad">
                <h2><i class="fa-solid fa-shield-halved"></i> HMI КОНТРОЛЬ ДОСТУПУ</h2>
                <p style="color: #aaa; font-size: 11px; margin:0 0 12px 0;">Введіть інженерний PIN-код: 1234 #</p>
                <div class="keypad-display" id="pinDisplay">----</div>
```

```
            <div class="keypad-grid">
                                <button class="keypad-btn" onclick="pressKey('1')">1</button><button class="keypad-btn" onclick="pressKey('2')">2</button><button class="keypad-btn" onclick="pressKey('3')">3</button>
                                <button class="keypad-btn" onclick="pressKey('4')">4</button><button class="keypad-btn" onclick="pressKey('5')">5</button><button class="keypad-btn" onclick="pressKey('6')">6</button>
                                <button class="keypad-btn" onclick="pressKey('7')">7</button><button class="keypad-btn" onclick="pressKey('8')">8</button><button class="keypad-btn" onclick="pressKey('9')">9</button>
                    <button class="keypad-btn" onclick="pressKey('*')" style="background:#dc3545;"><i class="fa-solid fa-trash-can"></i></button><button class="keypad-btn" onclick="pressKey('0')">0</button><button class="keypad-btn" onclick="pressKey('#')" style="background:#28a745;"><i class="fa-solid fa-check"></i></button>
            </div>
          </div>
        </div>

        <div class="container">
          <div class="header">
            <div style="display: flex; align-items: center; gap: 15px;">
              <h2><i class="fa-solid fa-network-wired"></i> Програмний комплекс IoT Стенду</h2>
              <div class="switch-container">
                <i class="fa-solid fa-sliders"></i> Режим налаштування
                <label class="switch">
                  <input type="checkbox" id="editToggle" onchange="toggleEditMode()">
                  <span class="slider"></span>
                </label>
              </div>
            </div>

            <div style="display: flex; align-items: center; gap: 10px;">
              <div class="switch-container" style="background:#f1f3f5; border: 1px solid #ced4da;">
                <i class="fa-solid fa-box-archive"></i> Конфігурація:
                                    <select id="templateSelect" style="width:130px; margin:0; padding:2px 5px;" onchange="loadSystemTemplate()"></select>
                 <button class="btn-delete edit-only" onclick="deleteSystemTemplate()" style="padding:2px 8px; margin:0 5px 0 0;" title="Видалити обраний шаблон"><i class="fa-solid fa-trash-can"></i></button>
                  <input type="text" autocomplete="off" spellcheck="false" id="newTemplateName" placeholder="Новий шаблон" style="width:100px; margin:0; padding:2px 5px;" class="edit-only">
                      <button class="btn-add edit-only" onclick="saveSystemTemplate()" style="padding:2px 8px; margin:0;" title="Зберегти поточний стан як шаблон"><i class="fa-solid fa-floppy-disk"></i></button>
              </div>
              <button class="btn-lock" onclick="lockSystem()"><i class="fa-solid fa-lock"></i> Блокувати</button>
            </div>
          </div>

          <div class="scenarios-panel">
            <h3><i class="fa-solid fa-clapperboard"></i> Глобальні Сценарії:</h3>
```

```
                <button class="btn-scenario btn-scen-fire" onclick="runScenario('fire')"><i class="fa-solid fa-fire"></i> Пожежна небезпека</button>
                <button class="btn-scenario btn-scen-night" onclick="runScenario('night')"><i class="fa-solid fa-moon"></i> Нічна охорона</button>
                <button class="btn-scenario btn-scen-norm" onclick="runScenario('normal')"><i class="fa-solid fa-sun"></i> Скинути стан</button>
            </div>

            <div class="control-panel edit-only-flex" style="display:none;">
                <div class="panel-box">
                    <h4><i class="fa-solid fa-plus"></i> Реєстрація нового датчика</h4>
                    <input type="text" autocomplete="off" spellcheck="false" id="sensorName" placeholder="Назва (напр. Склад)">
                    <select id="sensorType">
                        <option value="temperature">Температура (°C)</option>
                        <option value="humidity">Вологість (%)</option>
                        <option value="air_quality">Якість повітря (CO2 ppm)</option>
                        <option value="gas">Датчик газу (MQ-2 ppm)</option>
                        <option value="current">Споживання струму (A)</option>
                        <option value="ultrasonic">Далекомір (см)</option>
                        <option value="door">Датчик дверей (Геркон)</option>
                        <option value="motion">Датчик руху (PIR)</option>
                        <option value="smoke">Пожежний сповіщувач</option>
                    </select>
                    <button class="btn-add" onclick="addEntity('sensors', 'sensorName', 'sensorType')">Додати датчик</button>
                </div>

                <div class="panel-box">
                    <h4><i class="fa-solid fa-plus"></i> Реєстрація нового пристрою</h4>
                    <input type="text" autocomplete="off" spellcheck="false" id="actName" placeholder="Назва (напр. Сирена)">
                    <select id="actType">
                        <option value="fan">Вентилятор / Витяжка</option>
                        <option value="lamp">Розумна лампа</option>
                        <option value="siren">Сирена Тривоги</option>
                        <option value="lock">Електрозамок</option>
                        <option value="servo">Сервомотор</option>
                        <option value="lcd">LCD Дисплей</option>
                        <option value="rgb">RGB Індикатор</option>
                    </select>
                    <button class="btn-add" style="background-color: #007bff;" onclick="addEntity('actuators', 'actName', 'actType')">Додати пристрій</button>
                </div>

                <div class="panel-box" style="flex: 1.5;">
                    <h4><i class="fa-solid fa-gears"></i> Конфігурація правила взаємодії (M2M)</h4>
                    <input type="hidden" id="editRuleId" value="">
                    <div style="display:flex; gap:5px;">
```

```
                    <select id="ruleSensor" style="width:40%;" onchange="updateRuleUI()"></select>
                    <span id="condContainer" style="width:20%;"></span>
                    <span id="valContainer" style="width:40%;"></span>
                </div>
                <div style="display:flex; gap:5px; margin-top:5px;">
                    <span style="padding-top:6px; font-weight:bold;">➔</span>
                    <select id="ruleAct" style="width:50%;" onchange="updateRuleUI()"></select>
                    <span id="actionContainer" style="width:50%;"></span>
                </div>
                    <button id="btnSaveRule" class="btn-add" style="background-color: #ffc107; color: black; margin-top:8px;" onclick="saveRule()">Зберегти правило</button>
            </div>
        </div>

            <div class="grid-section"><h3><i class="fa-solid fa-code-branch"></i> Правила міжмашинної взаємодії (M2M Логіка)</h3><div class="grid grid-rules" id="rulesContainer"></div></div>
            <div class="grid-section"><h3><i class="fa-solid fa-satellite-dish"></i> Моніторинг телеметрії (Сенсори)</h3><div class="grid" id="sensorsContainer"></div></div>
        <div class="grid-section"><h3><i class="fa-solid fa-microchip"></i> Стан обладнання (Актуатори)</h3><div class="grid" id="actuatorsContainer"></div></div>

        <div class="grid-section">
            <h3><i class="fa-solid fa-terminal"></i> Системний журнал подій (Event Log)</h3>
            <div id="eventLog"></div>
        </div>
    </div>

    <div id="chartModal">
        <div class="modal-content">
            <span class="close-modal" onclick="closeChart()">&times;</span>
            <h3 id="chartTitle" style="margin-top:0; font-size:14px;"><i class="fa-solid fa-chart-line"></i> Тренд аналітики</h3>
            <canvas id="sensorChart" width="400" height="130"></canvas>
        </div>
    </div>

    <script src="{{ url_for('static', filename='js/app.js') }}"></script>

    </body>
    </html>
```

style.css

```
body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; background-color: #f4f6f9; padding: 10px; margin: 0; color: #333;}

.edit-only { display: none !important; }
body.edit-mode .edit-only { display: block !important; }
```

```
body.edit-mode .edit-only-flex { display: flex !important; }

.custom-toast {
  position: fixed; bottom: 20px; right: 20px; background: #dc3545; color: white;
  padding: 15px 20px; border-radius: 8px; box-shadow: 0 10px 25px rgba(220,53,69,0.4);
  font-weight: 600; font-size: 14px; z-index: 10000; display: flex; align-items: center; gap: 10px;
  animation: slideIn 0.3s cubic-bezier(0.175, 0.885, 0.32, 1.275) forwards, fadeOut 0.3s ease-in 2.7s forwards;
}
@keyframes slideIn { 0% { transform: translateX(100%); opacity: 0; } 100% { transform: translateX(0); opacity: 1; } }
@keyframes fadeOut { 0% { opacity: 1; } 100% { opacity: 0; } }

#lockScreen { position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(20, 20, 30, 0.98); z-index: 9999; display: flex; justify-content: center; align-items: center; flex-direction: column; color: white;}
.keypad { background: #2c2c38; padding: 20px; border-radius: 12px; box-shadow: 0 10px 40px rgba(0,0,0,0.8); text-align: center; border: 1px solid #444;}
.keypad-display { background: #111; color: #0f0; font-family: monospace; font-size: 24px; padding: 10px; margin-bottom: 15px; border-radius: 6px; letter-spacing: 8px; height: 25px; display: flex; align-items: center; justify-content: center; border: 2px solid #333;}
.keypad-grid { display: grid; grid-template-columns: repeat(3, 1fr); gap: 8px; }
.keypad-btn { background: #444; border: none; color: white; font-size: 20px; padding: 15px; border-radius: 6px; cursor: pointer; transition: 0.2s; font-weight: bold;}
.keypad-btn:hover { background: #555; }

.container { max-width: 1400px; margin: auto; background: white; padding: 15px; border-radius: 10px; box-shadow: 0 4px 15px rgba(0,0,0,0.05); }
.header { display: flex; justify-content: space-between; align-items: center; border-bottom: 2px solid #eee; padding-bottom: 10px; margin-bottom: 15px;}
.header h2 { margin: 0; font-size: 18px; color: #2c3e50; font-weight: 600;}
.btn-export { background-color: #6f42c1; color: white; padding: 8px 12px; text-decoration: none; border-radius: 4px; font-weight: bold; font-size: 12px; border: none; cursor: pointer;}
.btn-export:hover { background-color: #59339d; }
.btn-lock { background-color: #dc3545; color: white; padding: 8px 12px; border: none; border-radius: 4px; font-weight: bold; cursor: pointer; margin-left: 8px; font-size: 12px;}
.btn-lock:hover { background-color: #c82333; }

.switch-container { display: flex; align-items: center; gap: 10px; background: #e9ecef; padding: 5px 15px; border-radius: 20px; font-weight: bold; font-size: 13px;}
.switch { position: relative; display: inline-block; width: 40px; height: 20px; }
.switch input { opacity: 0; width: 0; height: 0; }
.slider { position: absolute; cursor: pointer; top: 0; left: 0; right: 0; bottom: 0; background-color: #ccc; transition: .4s; border-radius: 20px;}
.slider:before { position: absolute; content: ""; height: 16px; width: 16px; left: 2px; bottom: 2px; background-color: white; transition: .4s; border-radius: 50%;}
input:checked + .slider { background-color: #2196F3; }
input:checked + .slider:before { transform: translateX(20px); }
```

```
.scenarios-panel { background: #2b3035; color: white; padding: 10px; border-radius: 8px; margin-bottom: 15px; display: flex; align-items: center; gap: 12px;}
.scenarios-panel h3 { margin: 0; font-size: 14px; border-right: 1px solid #555; padding-right: 15px; font-weight: 500;}
.btn-scenario { background: #495057; color: white; padding: 6px 12px; border: none; border-radius: 4px; cursor: pointer; font-weight: bold; font-size: 11px; transition: 0.2s;}
.btn-scenario:hover { background: #6c757d; }
.btn-scen-fire { border-bottom: 3px solid #dc3545; }
.btn-scen-night { border-bottom: 3px solid #007bff; }
.btn-scen-norm { border-bottom: 3px solid #28a745; }

.control-panel { display: flex; gap: 15px; margin-bottom: 20px; flex-wrap: wrap; }
.panel-box { background: #f8f9fa; padding: 12px; border-radius: 8px; flex: 1; min-width: 280px; border: 1px solid #e9ecef;}
.panel-box h4 { margin: 0 0 8px 0; font-size: 13px; color: #495057; border-bottom: 1px solid #ddd; padding-bottom: 4px; font-weight: 600;}

input[type="text"], select, input[type="color"] {
    padding: 6px; font-size: 12px; margin-bottom: 8px; width: 100%; box-sizing: border-box;
    border: 1px solid #ccc; border-radius: 4px; transition: border-color 0.3s, box-shadow 0.3s;
}
input:focus, select:focus { outline: none; border-color: #80bdff; box-shadow: 0 0 0 0.2rem rgba(0,123,255,.25); }

button { padding: 6px 10px; border: none; border-radius: 4px; cursor: pointer; font-size: 12px; font-weight: bold;}
.btn-add { background-color: #28a745; color: white;}
.btn-add:hover { background-color: #218838; }
.btn-warn { background-color: #ffc107; color: black; }
.btn-warn:hover { background-color: #e0a800; }
.btn-recover { background-color: #17a2b8; color: white; }
.btn-recover:hover { background-color: #138496; }
.btn-delete { background-color: #dc3545; color: white; margin-top: 5px;}
.btn-delete:hover { background-color: #c82333; }
.btn-chart { background-color: #0d6efd; color: white; width: 32px; border-radius: 4px; padding: 4px;}
.btn-chart:hover { background-color: #0b5ed7; }

.grid-section h3 { font-size: 14px; color: #2c3e50; margin: 0 0 10px 0; border-left: 4px solid #007bff; padding-left: 8px; font-weight: 600;}
.grid { display: grid; grid-template-columns: repeat(auto-fill, minmax(210px, 1fr)); gap: 12px; margin-bottom: 20px; }
.grid-rules { grid-template-columns: repeat(auto-fill, minmax(310px, 1fr)); }

.card { background: #fff; padding: 12px; border-radius: 8px; border: 1px solid #e0e0e0; text-align: center; box-shadow: 0 2px 4px rgba(0,0,0,0.02); display: flex; flex-direction: column; justify-content: space-between;}
.card h4 { margin: 0 0 6px 0; color: #333; font-size: 13px; text-align: left; font-weight: 600;}
.card.offline { background-color: #fff3f3; border-color: #ffcaca; opacity: 0.85; }

.value-text { font-size: 20px; font-weight: bold; margin: 6px 0; }
.val-blue { color: #0d6efd; }
.val-offline { color: #dc3545; }
```

```
.status-badge { font-size: 9px; font-weight: bold; padding: 2px 6px; border-radius: 10px; display: inline-block; margin-bottom: 5px; width: fit-content;}
.status-online { background-color: #d4edda; color: #155724; border: 1px solid #c3e6cb;}
.status-offline { background-color: #f8d7da; color: #721c24; border: 1px solid #f5c6cb;}

.state-on { color: #155724; background-color: #d4edda; border: 1px solid #c3e6cb; padding: 4px 8px; border-radius: 4px; display: inline-block; margin: 5px 0; font-size: 12px; font-weight: bold;}
.state-off { color: #383d41; background-color: #e2e3e5; border: 1px solid #d6d8db; padding: 4px 8px; border-radius: 4px; display: inline-block; margin: 5px 0; font-size: 12px;}
.state-alarm { color: white; background-color: #dc3545; padding: 4px 8px; border-radius: 4px; display: inline-block; margin: 5px 0; font-size: 12px; font-weight: bold; animation: blink 1s infinite;}

.lcd-screen { background: #000; color: #00ff00; font-family: 'Courier New', Courier, monospace; font-size: 12px; padding: 8px; border-radius: 4px; border: 3px solid #333; min-height: 15px; margin: 5px 0; text-align: left; font-weight: bold;}
.rule-text { font-size: 12px; margin-bottom: 5px; line-height: 1.4; color: #444;}
.rule-error { background-color: #fff3f3; border: 1px solid #ffcaca !important; border-left: 4px solid #dc3545 !important;}

#eventLog { background: #1e1e24; color: #a9b7c6; font-family: 'Courier New', monospace; padding: 10px; height: 100px; overflow-y: scroll; border-radius: 6px; font-size: 11px; border: 1px solid #333;}
.log-entry { margin-bottom: 2px; border-bottom: 1px solid #2b2b31; padding-bottom: 2px;}
.log-err { color: #ff6b6b; font-weight: bold; }
.log-warn { color: #ffb86c; }
.log-success { color: #50fa7b; }

#chartModal { display: none; position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0,0,0,0.75); z-index: 1000; justify-content: center; align-items: center; }
.modal-content { background: #fff; padding: 20px; border-radius: 8px; width: 95%; max-width: 750px; position: relative; box-shadow: 0 10px 30px rgba(0,0,0,0.5);}
.close-modal { position: absolute; top: 5px; right: 12px; font-size: 26px; cursor: pointer; color: #666; transition: 0.2s;}
.close-modal:hover { color: #000; }

input[type="range"] { accent-color: #007bff; }
@keyframes blink { 50% { opacity: 0.4; } }
```

app.js

```
function showError(msg, inputId = null) {
    logger("Помилка: " + msg, "err");
    const toast = document.createElement('div');
    toast.className = 'custom-toast';
    toast.innerHTML = `<i class="fa-solid fa-triangle-exclamation"></i> <span>${msg}</span>`;
    document.body.appendChild(toast);
    setTimeout(() => toast.remove(), 3000);

    if (inputId) {
        const el = document.getElementById(inputId);
```

```
            if (el) {
                const oldBorder = el.style.borderColor;
                const oldBoxShadow = el.style.boxShadow;
                el.style.borderColor = '#dc3545';
                el.style.boxShadow = '0 0 8px rgba(220,53,69,0.5)';
                setTimeout(() => { el.style.borderColor = oldBorder; el.style.boxShadow = oldBoxShadow; }, 2500);
            }
        }
    }

    let currentPin = '';
    function pressKey(key) {
        if(key === '*') currentPin = '';
        else if(key === '#') {
            if(currentPin === '1234') { document.getElementById('lockScreen').style.display = 'none'; currentPin = ''; logger("Доступ до системи авторизовано успішно", "success");}
            else { document.getElementById('pinDisplay').innerText = 'ERR'; logger("Помилка авторизації. Невірний код access-key", "err"); setTimeout(() => { currentPin = ''; updateDisplay(); }, 1000); return; }
        } else if(currentPin.length < 4) currentPin += key;
        updateDisplay();
    }
    function updateDisplay() {
        if(document.getElementById('pinDisplay').innerText === 'ERR') return;
        document.getElementById('pinDisplay').innerText = currentPin.padEnd(4, '-').replace(/[0-9]/g, '*');
    }
    function lockSystem() { document.getElementById('lockScreen').style.display = 'flex'; currentPin = ''; updateDisplay(); logger("Консоль заблоковано користувачем", "warn");}

    function toggleEditMode() {
        if(document.getElementById('editToggle').checked) { document.body.classList.add('edit-mode'); logger("Активовано інженерний режим редагування", "warn"); }
        else { document.body.classList.remove('edit-mode'); logger("Повернення до режиму оперативного моніторингу", "success"); }
    }

    function logger(msg, type="info") {
        const logBox = document.getElementById('eventLog');
        const entry = document.createElement('div');
        entry.className = `log-entry ${type==='err'?'log-err':type==='warn'?'log-warn':type==='success'?'log-success':''}`;
        entry.innerHTML = `[${new Date().toLocaleTimeString()}] <i class="fa-solid ${type==='err'?'fa-circle-xmark':type==='warn'?'fa-triangle-exclamation':'fa-circle-info'}"></i> ${msg}`;
        logBox.prepend(entry);
    }

    async function loadTemplateList() {
        const res = await fetch('/api/templates');
```

```
        const list = await res.json();
        const select = document.getElementById('templateSelect');
          select.innerHTML = list.map(t => `<option value="${t}">${t}</option>`).join('') + `<option value="" selected disabled>Оберіть шаблон...</option>`;
    }
    async function saveSystemTemplate() {
        const name = document.getElementById('newTemplateName').value;
         if(!name || name.trim() === "") { showError("Введіть назву для збереження нового шаблону!", 'newTemplateName'); return; }
          const res = await fetch('/api/templates/save', { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ name }) });
                if(res.ok) { logger(`Конфігурацію збережено як шаблон: ${name}`, "success"); document.getElementById('newTemplateName').value = ''; loadTemplateList(); }
    }
    async function loadSystemTemplate() {
        const name = document.getElementById('templateSelect').value;
        if(!name) return;
          const res = await fetch('/api/templates/load', { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ name }) });
        if(res.ok) { logger(`Розгорнуто конфігураційний профіль: ${name}`, "success"); fetchData(); }
    }
    async function deleteSystemTemplate() {
        const select = document.getElementById('templateSelect');
        const name = select.value;
        if(!name) { showError("Оберіть існуючий шаблон для видалення!", 'templateSelect'); return; }
        if(!confirm(`Ви впевнені, що хочете видалити шаблон "${name}"?`)) return;

         const res = await fetch('/api/templates/delete', { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ name }) });
        if(res.ok) {
            logger(`Шаблон конфігурації видалено: ${name}`, "warn");
            select.value = "";
            loadTemplateList();
        } else {
            const errData = await res.json();
            showError(errData.error);
        }
    }

    async function runScenario(name) { await fetch(`/api/scenarios/${name}`, { method: 'POST' }); logger(`Виконано макрос: [${name}]`, "warn"); fetchData(); }

    let currentChart = null, activeChartSensorId = null;
    async function openChart(sensorId, sensorName) {
        activeChartSensorId = sensorId;
        document.getElementById('chartTitle').innerHTML = `<i class="fa-solid fa-chart-line"></i> Аналітика: ${sensorName}`;
```

```
        document.getElementById('chartModal').style.display = 'flex';
        await updateChart();
    }
    function closeChart() { document.getElementById('chartModal').style.display = 'none'; activeChartSensorId = null; if(currentChart) { currentChart.destroy(); currentChart = null; } }
    async function updateChart() {
        if(!activeChartSensorId) return;
        const res = await fetch(`/api/history/${activeChartSensorId}`);
        const serverData = await res.json();
        const labels = serverData.points.map(d => d.time);
        const values = serverData.points.map(d => d.value);
        const thValue = serverData.threshold;

        const datasets = [{ label: 'Поточні дані', data: values, borderColor: '#0d6efd', backgroundColor: 'rgba(13,110,253,0.05)', fill: true, tension: 0.2 }];
        if (thValue !== null) datasets.push({ label: `Уставка (${thValue})`, data: Array(values.length).fill(thValue), borderColor: '#dc3545', borderWidth: 2, borderDash: [5, 5], pointRadius: 0, fill: false });

        if(currentChart) { currentChart.data.labels = labels; currentChart.data.datasets = datasets; currentChart.update(); }
        else { currentChart = new Chart(document.getElementById('sensorChart').getContext('2d'), { type: 'line', data: { labels: labels, datasets: datasets }, options: { animation: false, responsive: true } }); }
    }

    let globalSensors = [], globalActuators = [], globalRules = [];

    function getFAIcon(type) {
        const icons = {
            'temperature': '<i class="fa-solid fa-temperature-three-quarters" style="color:#e0a800;"></i>',
            'humidity': '<i class="fa-solid fa-droplet" style="color:#0dc6ff;"></i>',
            'air_quality': '<i class="fa-solid fa-wind" style="color:#20c997;"></i>',
            'gas': '<i class="fa-solid fa-smog" style="color:#6c757d;"></i>',
            'current': '<i class="fa-solid fa-bolt-lightning" style="color:#fd7e14;"></i>',
            'ultrasonic': '<i class="fa-solid fa-ruler-horizontal" style="color:#6f42c1;"></i>',
            'door': '<i class="fa-solid fa-door-closed"></i>',
            'motion': '<i class="fa-solid fa-person-walking"></i>',
            'smoke': '<i class="fa-solid fa-radiation" style="color:#dc3545;"></i>',
            'fan': '<i class="fa-solid fa-fan text-secondary"></i>',
            'lamp': '<i class="fa-solid fa-lightbulb text-warning"></i>',
            'siren': '<i class="fa-solid fa-bullhorn text-danger"></i>',
            'lock': '<i class="fa-solid fa-key text-success"></i>',
            'servo': '<i class="fa-solid fa-gears text-primary"></i>',
            'lcd': '<i class="fa-solid fa-desktop text-dark"></i>',
            'rgb': '<i class="fa-solid fa-palette"></i>'
        };
        return icons[type] || '<i class="fa-solid fa-microchip"></i>';
    }
```

```
async function addEntity(apiEndpoint, nameId, typeId) {
  const name = document.getElementById(nameId).value;
  const type = document.getElementById(typeId).value;
  if (!name || name.trim() === "") { showError("Поле назви не може бути порожнім!", nameId); return; }
   const res = await fetch(`/api/${apiEndpoint}`, { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ name, type }) });
    if(res.ok) { logger(`Успішно додано об'єкт: ${name}`, "success"); document.getElementById(nameId).value = ''; fetchData(); }
  else { const errData = await res.json(); showError(errData.error); }
}

async function deleteEntity(endpoint, id) { await fetch(`/api/${endpoint}/${id}`, { method: 'DELETE' }); logger(`Видалено вузол з системи`, "warn"); fetchData(); }
async function updateManualValue(id, value) { await fetch(`/api/sensors/${id}/value`, { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ value: value }) }); fetchData(); }
async function toggleNetwork(id) { await fetch(`/api/sensors/${id}/toggle_network`, { method: 'POST' }); fetchData(); }

function updateRuleUI() {
  const s = globalSensors.find(x => x.id === document.getElementById('ruleSensor').value);
  const a = globalActuators.find(x => x.id === document.getElementById('ruleAct').value);
  if (!s || !a) return;

  const cCont = document.getElementById('condContainer');
  const vCont = document.getElementById('valContainer');
  if (['motion', 'smoke', 'door'].includes(s.type)) {
    cCont.innerHTML = `<select id="ruleCond"><option value="==">==</option></select>`;
    let txtOn = s.type === 'door' ? 'Відчинено' : 'Активно (Тривога)';
    let txtOff = s.type === 'door' ? 'Зачинено' : 'Спокійно';
        vCont.innerHTML = `<select id="ruleVal"><option value="1">${txtOn}</option><option value="0">${txtOff}</option></select>`;
  } else {
        cCont.innerHTML = `<select id="ruleCond"><option value=">">&gt;</option><option value="<">&lt;</option></select>`;
    vCont.innerHTML = `<input type="text" autocomplete="off" spellcheck="false" id="ruleVal" placeholder="Введіть число межі">`;
  }

  const aCont = document.getElementById('actionContainer');
   if (a.type === 'servo') aCont.innerHTML = `<input type="text" autocomplete="off" spellcheck="false" id="ruleAction" placeholder="Кут 0-180°">`;
   else if (a.type === 'lcd') aCont.innerHTML = `<input type="text" autocomplete="off" spellcheck="false" id="ruleAction" placeholder="Текст на екран">`;
   else if (a.type === 'rgb') aCont.innerHTML = `<input type="color" id="ruleAction" value="#ff0000" style="height:32px; padding:0;">`;
  else {
```

```
            let tOn = a.type === 'lock' ? 'ЗАКРИТИ' : 'УВІМКНУТИ';
            let tOff = a.type === 'lock' ? 'ВІДЧИНИТИ' : 'ВИМКНУТИ';
                    aCont.innerHTML = `<select id="ruleAction"><option value="true">${tOn}</option><option value="false">${tOff}</option></select>`;
        }
    }

    function editRule(id) {
        const r = globalRules.find(x => x.id === id);
        if(!r) return;
        document.getElementById('editRuleId').value = r.id;
        document.getElementById('ruleSensor').value = r.sensor_id;
        document.getElementById('ruleAct').value = r.actuator_id;
        updateRuleUI();

        document.getElementById('ruleCond').value = r.condition;
        document.getElementById('ruleVal').value = r.threshold;

        if (typeof r.action === 'boolean') document.getElementById('ruleAction').value = r.action.toString();
        else document.getElementById('ruleAction').value = r.action;

        const btn = document.getElementById('btnSaveRule');
        btn.innerHTML = '<i class="fa-solid fa-pen-to-square"></i> Оновити правило';
        btn.style.backgroundColor = "#28a745"; btn.style.color = "white";
        logger(`Завантажено редактор правила`, "info");
    }

    async function saveRule() {
        const rId = document.getElementById('editRuleId').value;
        const sensor_id = document.getElementById('ruleSensor').value;
        const condition = document.getElementById('ruleCond').value;
        const valInput = document.getElementById('ruleVal');
        const threshold = parseFloat(valInput.value);
        let actionVal = document.getElementById('ruleAction').value;
        const a = globalActuators.find(x => x.id === document.getElementById('ruleAct').value);

        if (isNaN(threshold) || valInput.value.trim() === "") { showError("Некоректне значення! Межа повинна бути числом.", 'ruleVal'); return; }
        if (actionVal.trim() === "") { showError("Необхідно вказати дію або значення для пристрою!", 'ruleAction'); return; }
        if (a.type === 'servo') actionVal = parseInt(actionVal);

        const payload = { sensor_id, condition, threshold, actuator_id: a.id, action: actionVal };
        if(rId) payload.id = rId;

        const res = await fetch('/api/rules', { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify(payload) });
```

```
            if(res.ok) {
                logger(rId ? `Логічне правило оновлено` : "Створено нову M2M логіку", "success");
                document.getElementById('editRuleId').value = '';
                document.getElementById('ruleVal').value = '';
                const btn = document.getElementById('btnSaveRule');
                btn.innerHTML = 'Зберегти правило'; btn.style.backgroundColor = "#ffc107"; btn.style.color = "black";
                fetchData();
            } else {
                const errData = await res.json();
                showError(errData.error);
            }
        }

        async function deleteRule(id) { await fetch(`/api/rules/${id}`, { method: 'DELETE' }); logger(`Логічне правило видалено`, "warn"); fetchData(); }

        async function fetchData() {
            const [resSens, resAct, resRules] = await Promise.all([ fetch('/api/sensors'), fetch('/api/actuators'), fetch('/api/rules') ]);
            globalSensors = await resSens.json();
            globalActuators = await resAct.json();
            globalRules = await resRules.json();

            if(activeChartSensorId) updateChart();

            const sSelect = document.getElementById('ruleSensor');
            const aSelect = document.getElementById('ruleAct');
                if(sSelect.options.length !== globalSensors.length) { sSelect.innerHTML = globalSensors.map(s => `<option value="${s.id}">${s.name}</option>`).join(''); updateRuleUI(); }
                if(aSelect.options.length !== globalActuators.length) { aSelect.innerHTML = globalActuators.map(a => `<option value="${a.id}">${a.name}</option>`).join(''); updateRuleUI(); }

            document.getElementById('rulesContainer').innerHTML = globalRules.map(r => {
                const sObj = globalSensors.find(s => s.id === r.sensor_id);
                const aObj = globalActuators.find(a => a.id === r.actuator_id);
                if (!sObj || !aObj) {
                    return `<div class="card rule-error" style="text-align: left; display: flex; flex-direction:row; justify-content: space-between; align-items: center; padding:8px 12px;">
                        <div class="rule-text" style="color:#721c24;"><i class="fa-solid fa-triangle-exclamation"></i> Помилка: вузол відсутній</div>
                        <button class="btn-delete edit-only" onclick="deleteRule('${r.id}')" style="width:auto; padding: 2px 8px; margin:0;"><i class="fa-solid fa-trash-can"></i></button>
                    </div>`;
                }

                let aText = r.action;
```

```
                if (r.action === true) aText = `<span style="color:green; font-weight:bold;">${aObj.type === 'lock' ? 'ЗАКРИТИ' : 'УВІМКНУТИ'}</span>`;
                else if (r.action === false) aText = `<span style="color:gray; font-weight:bold;">${aObj.type === 'lock' ? 'ВІДЧИНИТИ' : 'ВИМКНУТИ'}</span>`;
                  else if (typeof r.action === 'string' && r.action.startsWith('#')) aText = `<span style="color:${r.action}; font-weight:bold;">Колір ${r.action}</span>`;
                else if (typeof r.action === 'string') aText = `Текст "${r.action}"`;
                else aText = `${r.action}°`;

                let conditionTxt = r.condition === "==" ? (r.threshold === 1 ? "Активно" : "Спокійно") : r.threshold;

                 return `<div class="card" style="text-align: left; display: flex; flex-direction:row; justify-content: space-between; align-items: center; background: #fafafa; border-left: 4px solid #007bff; padding:8px 12px;">
                   <div class="rule-text">Якщо <b>${sObj.name}</b> ${r.condition} <b>${conditionTxt}</b> ➔ <b>${aObj.name}</b> : ${aText}</div>
                    <div style="display:flex; gap:3px;">
                                <button class="btn-warn edit-only" onclick="editRule('${r.id}')" style="padding:2px 8px; margin:0;" title="Редагувати правило"><i class="fa-solid fa-pen-to-square"></i></button>
                      <button class="btn-delete edit-only" onclick="deleteRule('${r.id}')" style="width:auto; padding: 2px 8px; margin:0;" title="Видалити правило"><i class="fa-solid fa-trash-can"></i></button>
                    </div>
                  </div>`;
              }).join('');

             document.getElementById('sensorsContainer').innerHTML = globalSensors.map(s => {
                let cardClass = s.is_online ? "card" : "card offline";
                 let badge = s.is_online ? `<div class="status-badge status-online">ОНЛАЙН</div>` : `<div class="status-badge status-offline">ОБРИВ ЛІНІЇ</div>`;
                let sDis = s.is_online ? "" : "disabled";

                let unit = '';
                if (s.type === 'temperature') unit = ' °C';
                else if (s.type === 'humidity') unit = ' %';
                else if (['air_quality', 'gas'].includes(s.type)) unit = ' ppm';
                else if (s.type === 'ultrasonic') unit = ' см';
                else if (s.type === 'current') unit = ' A';

                let valText, controlsHTML, chartBtnHTML = '';
                if (['motion', 'smoke', 'door'].includes(s.type)) {
                   let alertMsg = s.type === 'door' ? 'Відчинено' : (s.type === 'smoke' ? 'ДИМ' : 'Рух');
                   valText = s.is_online ? (s.value == 1 ? alertMsg : 'Спокійно') : "Немає зв'язку";
                   let btn1 = 'Активувати', btn2 = 'Скинути';
                   if (s.type === 'door') { btn1 = 'Відчинити двері'; btn2 = 'Зачинити'; }
                   if (s.type === 'smoke') { btn1 = 'Примусовий дим'; btn2 = 'Очистити'; }
                   if (s.type === 'motion') { btn1 = 'Імітувати рух'; btn2 = 'Відновити спокій'; }
```

```
                controlsHTML = `<div style="display:flex; flex-direction:column; gap:4px; margin: 8px 0;">
                        <button class="btn-warn" onclick="updateManualValue('${s.id}', 1)" ${sDis} style="font-size:11px; padding:4px;">${btn1}</button>
                    <button class="btn-add" onclick="updateManualValue('${s.id}', 0)" ${sDis} style="font-size:11px; padding:4px; background-color:#6c757d;">${btn2}</button>
                </div>`;
            } else {
                valText = s.is_online ? s.value + unit : "Немає зв'язку";
                let max = ['air_quality', 'gas'].includes(s.type) ? 2000 : (s.type==='ultrasonic' ? 400 : (s.type==='current'?20:100));
                let step = s.type==='current' ? 0.1 : 1;
                    controlsHTML = `<input type="range" min="0" max="${max}" step="${step}" value="${s.value}" onchange="updateManualValue('${s.id}', this.value)" style="width:100%; margin:8px 0;" ${sDis}>`;
                chartBtnHTML = `<button class="btn-chart" onclick="openChart('${s.id}', '${s.name}')" title="Аналітика"><i class="fa-solid fa-chart-line"></i></button>`;
            }

            let valColor = (s.value == 1 && ['motion','smoke','door'].includes(s.type)) ? 'val-offline' : 'val-blue';

            let heartbeatHTML = '';
            if (s.is_online) {
                const randomHeartbeat = Math.floor(Math.random() * 4) + 1;
                    heartbeatHTML = `<span style="font-size:9px; color:#999;"><i class="fa-solid fa-heart-pulse"></i> ${randomHeartbeat}с тому</span>`;
            } else {
                heartbeatHTML = `<span style="font-size:9px; color:#dc3545;"><i class="fa-solid fa-heart-crack"></i> втрачено</span>`;
            }

            return `<div class="${cardClass}">
                <div style="display:flex; justify-content: space-between; align-items: start; gap:10px;">
                    <h4 style="margin:0; font-size:13px; font-weight:600; text-align:left;">${getFAIcon(s.type)} ${s.name}</h4>
                    ${chartBtnHTML}
                </div>
                <div style="display:flex; justify-content: space-between; align-items: center; margin-top:5px;">
                    ${badge}
                    ${heartbeatHTML}
                </div>
                <div class="value-text ${valColor}">${valText}</div>
                ${controlsHTML}
                <div class="edit-only-flex" style="display:none; gap:4px; margin-top:5px;">
                    <button class="${s.is_online?'btn-warn':'btn-recover'}" onclick="toggleNetwork('${s.id}')" style="font-size:10px; padding:2px 6px;">${s.is_online?'🛠 Збій Wi-Fi':'🩹 Відновити'}</button>
                    <button class="btn-delete" onclick="deleteEntity('sensors', '${s.id}')" style="margin:0; font-size:10px; padding:2px 6px;"><i class="fa-solid fa-trash-can"></i></button>
                </div></div>`;
        }).join('');
```

```
        document.getElementById('actuatorsContainer').innerHTML = globalActuators.map(a => {
            let stateHTML = '';
            if (a.type === 'servo') stateHTML = `<div class="value-text val-blue" style="margin: 10px 0;">Кут: ${a.state}°</div>`;
            else if (a.type === 'lcd') stateHTML = `<div class="lcd-screen"><i class="fa-solid fa-terminal"></i> ${a.state}</div>`;
             else if (a.type === 'rgb') stateHTML = `<div style="background:${a.state}; width:100%; height:25px; border-radius:4px;
border:1px solid #ccc; margin:10px 0; box-shadow: inset 0 0 5px rgba(0,0,0,0.1);"></div>`;
            else {
                let sClass = a.state ? 'state-on' : 'state-off';
                if (a.type === 'siren' && a.state) sClass = 'state-alarm';
                let txt = a.state ? 'УВІМКНЕНО' : 'ВИМКНЕНО';
                if(a.type === 'lock') txt = a.state ? 'ЗАКРИТО 🔒' : 'ВІДЧИНЕНО 🔓';
                stateHTML = `<div class="${sClass}">${txt}</div>`;
            }

            return `<div class="card">
                <h4 style="text-align:left;">${getFAIcon(a.type)} ${a.name}</h4>
                ${stateHTML}
                <button class="btn-delete edit-only" onclick="deleteEntity('actuators', '${a.id}')" style="margin-top:auto;"><i class="fa-
solid fa-trash-can"></i> Видалити пристрій</button>
            </div>`;
        }).join('');
    }

    setInterval(fetchData, 1500);
    loadTemplateList();
    fetchData();
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олександр ОЛІЙНИК

Співавтор:

Назва: Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії

Експерт: Ольга АТАМАНЮК

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:4.04%

Коефіцієнт подібності 2:0.47%

Мікропробіли: 3

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-30 08:21:31.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-05-30 Доцент Андрій Нічепорук

Дата експерт

Sat May 30 09:10:31 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

	Документ		Сумарний збіг по Базі Даних	
ID: 272838 Назва: БКР Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп’ютерної інженерії Додано в БД: 2026-05-30 Автора: Олександр ОЛІЙНИК Керівники: Ольга АТАМАНЮК Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	110789	631	1816 (2%)	25 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Олійник Олександр Олександрович

Тема: Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___64___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є програмно-апаратного комплексу навчального стенду з Інтернету речей

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проаналізовано еволюцію технологій Інтернету речей та підходів до створення навчальних лабораторних стендів. Обґрунтовано важливість переходу від ізольованих мікроконтролерів до кіберфізичних просторів, що об'єднують інформаційні та операційні технології. Доведено, що впровадження високорівневих мов програмування дозволяє змістити освітній фокус із рутинного апаратного кодування на проектування складної архітектури, оркестрацію мережевих потоків та інтелектуальну міжмашинну взаємодію. У другому розділі спроектовано ключові підсистеми апаратно-програмного комплексу. Обґрунтовано трирівневу архітектуру з комунікаційною топологією «зірка» для забезпечення максимальної відмовостійкості. Розроблено алгоритми міжмашинної взаємодії, де для усунення брязкоту контактів успішно імплементовано програмний алгоритмічний гістерезис. Спроектовано реляційну базу даних із застосуванням патерну ORM, що гарантує абсолютну транзакційну цілісність телеметричної інформації та захист від кібервтручань. У третьому розділі реалізовано кіберфізичну систему та проведено комплекс її експериментальних випробувань. Створено інтерактивний людино-машинний

інтерфейс із надійним контролем доступу, побудовою графічних трендів і візуальним конструюванням логічних правил автоматизації. Практичні випробування підтвердили високу стабільність системи при обробці інтенсивних потоків телеметрії та імітації апаратних збоїв. Доведено, що створений комплекс є ефективним діючим прототипом для розгортання промислових систем диспетчеризації, клімат-контролю та комплексної безпеки.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага аналізу предметної області; недостатньо чітко описано процес складання програмно-апаратного комплексу.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на достатньому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: задовільно (D / 70)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

к.т.н., доц. Ткачук Ю.В.

доцент кафедри ЗІТ

"2" березня 2026 р. _____(підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Олександр ОЛІЙНИК

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2С-23-2

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Програмно-апаратний комплекс навчального стенду з Інтернету речей для підготовки фахівців з комп'ютерної інженерії
Автор Олександр ОЛІЙНИК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: асистент, Ольга АТАМАНЮК

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 4,04%; та системою Anti-Plagiarism складає 0,47%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми _____ Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ Підпис — Ольга АТАМАНЮК (Ім'я, ПРІЗВИЩЕ)