Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА ГІБРИДНОЇ IOT-СИСТЕМИ МОНІТОРИНГУ ТА АДАПТИВНОГО КЕРУВАННЯ ОХОЛОДЖЕННЯМ GPU-КЛАСТЕРУ

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-512

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

АЛЕКПЕРОВ В.Е.
(ПРІЗВИЩЕ та ініціали)

Керівник КУДЕРМЕТОВ Р.К.
(ПРІЗВИЩЕ та ініціали)
Рецензент РОМАНОВСЬКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
_____
«14» квітня 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АЛЕКПЕРОВА Віталія Едуардовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка гібридної IoT-системи моніторингу та адаптивного керування охолодженням GPU-кластеру
керівник проєкту (роботи) к.т.н., доцент, КУДЕРМЕТОВ Р.К.,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від « 14 » квітня 2026 року № 160
2. Строк подання студентом проєкту (роботи) 18.05.2026 р.
3. Вихідні дані до проєкту (роботи) опис предметної області, існуючі методи та алгоритми керування охолодженням, мікроконтролер ESP-32, персональний сервер – обчислювальний компонент

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Опис предметної області;
2) Проєктування системи;
3) Вибір компонентів та технологічного стеку системи;
4) Розробка системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	КУДЕРМЕТОВ Р.К.		
Нормоконтроль	ДЬЯЧУК Т.С.		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	до 18.04.2026	
2	Визначення та аналіз вимог до програмного	до 22.04.2026	
	забезпечення		
3	Розробка специфікації вимог SRS	до 24.04.2026	
4	Розробка описів прецедентів	до 28.04.2026	
5	Вибір компонентів та технологічного стеку системи	до 01.05.2026	
6	Проєктування архітектури програмного	до 05.05.2026	
	забезпечення		
7	Проєктування інтерфейсу користувача	до 12.05.2026	
8	Оформлення пояснювальної записки	до 18.05.2026	
9	Проходження нормоконтролю	до 01.06.2026	
10	Перевірка на наявність академічного плагіату	до 01.06.2026	
11	Проходження рецензування	до 01.06.2026	

Студент(ка) _____ Віталій АЛЕКПЕРОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Равіль КУДЕРМЕТОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 125 с., 13 табл., 24 рис., 4 дод., 20 джерел.

ІНТЕРНЕТ РЕЧЕЙ, АВТОМАТИЗАЦІЯ, СИСТЕМА МОНІТОРИНГУ, ЦЕНТР ОБРОБКИ ДАНИХ, ХМАРНІ ТЕХНОЛОГІЇ, ТУМАННІ ОБЧИСЛЕННЯ, АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС, ВЕБ-СИСТЕМА

Об’єкт розробки – гібридна IoT-система моніторингу та адаптивного керування охолодженням GPU-кластеру.

Мета роботи – запобігання прискореній деградації обчислювального обладнання в локальних дата-центрах та забезпечення його стабільної роботи за рахунок автоматизованого випереджального керування і компенсації негативних впливів мікроклімату.

У першому розділі було проведено аналіз предметної області, досліджено вплив термодинамічних параметрів та неконтрольованого мікроклімату на електронні компоненти, а також обґрунтовано необхідність використання гібридної хмарно-туманної архітектури.

У другому розділі здійснено проєктування системи: визначено функціональні та нефункціональні вимоги, побудовано структуру взаємодії компонентів. Розроблено математичні моделі обчислення зносу обладнання: індекс корозії та механічного зносу вентиляторів, а також описано алгоритми керування.

У третьому розділі обґрунтовано вибір апаратного забезпечення: відібрано необхідні сенсори температури, вологості, концентрації пилу, мікроконтролер як граничний шлюз та виконавчі механізми. Також обрано сучасний стек технологій для реалізації туманного сервера та веб-системи моніторингу.

У четвертому розділі детально описано процес розробки та налаштування програмно-апаратного комплексу. Проведено комплексне тестування системи.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АЦП – Аналогово-цифровий перетворювач

БД – База даних

ОЗП – Оперативний запам'ятовуючий пристрій

СКБД – Система керування базами даних

ЦОД – Центр обробки даних

ШІМ – Широтно-імпульсна модуляція

API – Application Programming Interface, Інтерфейс прикладного програмування

AWS – Amazon Web Services, Хмарна платформа

BSON – Binary JSON, Бінарний формат представлення даних JSON

CI – Corrosion Index, Розрахунковий індекс корозії елементної бази

CSS – Cascading Style Sheets, Каскадні таблиці стилів

FWI – Fan Wear Index, Кумулятивний індекс механічного зносу вентиляторів

GPIO – General-Purpose Input/Output, Інтерфейс вводу/виводу загального призначення

GPU – Graphics Processing Unit, Графічний процесор

HTTP – HyperText Transfer Protocol, Протокол передачі гіпертексту

I2C – Inter-Integrated Circuit, Послідовна шина даних для зв'язку інтегральних схем

IoT – Internet of Things, Інтернет речей

JS – JavaScript, Мова програмування

JSON – JavaScript Object Notation, Текстовий формат обміну даними

MCU – Microcontroller Unit, Мікроконтролер

ML – Machine Learning, Машинне навчання

MQTT – Message Queuing Telemetry Transport, Протокол обміну повідомленнями

NTC – Negative Temperature Coefficient, Термістор з негативним температурним коефіцієнтом

OLED – Organic Light-Emitting Diode, Органічний світлодіод

PGA – Programmable Gain Amplifier, Програмований підсилювач зі змінним коефіцієнтом

PM – Particulate Matter, Дисперсний пил (тверді мікрочастинки у повітрі)

PWM – Pulse-Width Modulation, Широтно-імпульсна модуляція

RAM – Random Access Memory, Оперативна пам'ять

REST – Representational State Transfer, Архітектурний стиль взаємодії компонентів розподіленого додатка

RH – Relative Humidity, Відносна вологість повітря

RPM – Revolutions Per Minute, Кількість обертів вентилятора за хвилину

SCL – Serial Clock, Лінія тактової синхронізації

SDA – Serial Data, Лінія передачі даних

SPI – Serial Peripheral Interface, Послідовний периферійний інтерфейс

SVG – Scalable Vector Graphics, Масштабована векторна графіка

TFT – Thin-Film Transistor, Тонкоплівковий транзистор

TSDB – Time Series Database, База даних часових рядів

TSM – Time-Structured Merge Tree, Деревоподібна структура зберігання часових рядів

UI – User Interface, Інтерфейс користувача

UART – Universal Asynchronous Receiver-Transmitter, Універсальний асинхронний приймач-передавач

URL – Uniform Resource Locator, Уніфікований покажчик ресурсу

ВСТУП

Сучасні локальні обчислювальні інфраструктури, зокрема GPU-кластери, є невід'ємною складовою для вирішення складних задач, таких як робота зі штучним інтелектом, рендеринг та обробка великих масивів даних. На відміну від великих промислових дата-центрів зі спеціалізованими інженерними підсистемами, локальні кластери часто розгортаються в умовах малого бізнесу або дослідницьких лабораторій, де відсутній повноцінний автоматизований кліматичний контроль.

Основна проблема експлуатації таких систем полягає у високій динаміці зміни термодинамічних параметрів. Асинхронне завантаження графічних процесорів зумовлює нелінійне тепловиділення та температурну інерцію, що призводить до циклічного термічного розширення та стискання елементної бази. Разом із неконтрольованими факторами навколишнього середовища, такими як перепади вологості та накопичення дисперсного пилу, це спричиняє прискорену деградацію дороговартісного обладнання, підвищує ризик статичних розрядів, корозії контактів та загальної відмови системи.

Для вирішення цих проблем необхідне впровадження сучасних систем інтернету речей , здатних здійснювати безперервний інтелектуальний моніторинг та компенсацію деструктивних факторів. Використання гібридної хмарно-туманної архітектури дозволяє забезпечити миттєву автономну реакцію локального сервера на температурні аномалії за допомогою випереджального керування, паралельно зберігаючи історичні дані та надаючи зручний веб-інтерфейс для глобального контролю. Впровадження такої системи мінімізує експлуатаційні ризики та продовжує життєвий цикл апаратного забезпечення шляхом зміщення зносу на допоміжні кліматичні пристрої.

Метою даної роботи є запобігання прискореній деградації обчислювального обладнання в умовах локальних дата-центрів та забезпечення його стабільної роботи за рахунок розробки гібридної IoT-системи моніторингу та адаптивного керування охолодженням GPU-кластеру.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз структури Дата-центрів

Центр обробки даних – це інфраструктурний об'єкт, призначений для розміщення, живлення, охолодження та мережевого підключення обчислювального обладнання. Попри різноманіття форм і масштабів, будь-який ЦОД вирішує одну й ту ж фундаментальну інженерну задачу: підтримувати умови, за яких апаратне забезпечення працює в межах допустимих параметрів протягом усього строку експлуатації [2].

Класичний промисловий ЦОД є складною інженерною системою, що поєднує серверні стійки, системи безперебійного живлення (ДБЖ), прецизійні кондиціонери (CRAC/CRAH-блоки), засоби протипожежного захисту та структуровану кабельну інфраструктуру [1]. Відповідно до стандарту Uptime Institute, такі об'єкти класифікуються за рівнями надійності від Tier I до Tier IV. Рівні Tier III та Tier IV передбачають повне резервування всіх інженерних підсистем і гарантують доступність обладнання на рівні 99,98% та 99,995% відповідно. Для досягнення таких показників мікроклімат суворо нормується: стандарт ASHRAE TC 9.9 визначає рекомендований діапазон температури на вході в стійку від 18°C до 27°C та відносну вологість від 40% до 60%.

Типова архітектура промислового ЦОД будується за принципом «гарячих» та «холодних» коридорів: холодне повітря подається з боку лицьових панелей стійок, а нагріте відводиться з тильного боку до систем кондиціонування. Ефективність такої організації оцінюється показником PUE – відношенням загального споживання до споживання безпосередньо IT-обладнанням. У передових об'єктів він наближається до 1,1–1,2 [3]. Обслуговується така інфраструктура виділеними командами інженерів, а всі відмови локалізуються і усуваються до того, як вони вплинуть на роботу сервісів.

Паралельно з промисловим сегментом у останні роки сформувався якісно інший клас інфраструктури – локальні обчислювальні вузли, що розгортаються

стартапами, малим бізнесом та дослідницькими командами. Рушієм цього явища стало стрімке поширення відкритих великих мовних моделей (open-source LLM), зростання попиту на генерацію медіаконтенту, автоматизацію бізнес-процесів та рендеринг. Виконання таких задач безпосередньо на хмарних ресурсах стає економічно невигідним при постійному навантаженні, що спонукає малий бізнес до придбання власного GPU-обладнання.

Такий «локальний ЦОД» може займати одне-два суміжних приміщення і на початковому етапі містити менше двадцяти відеокарт. Принципова відмінність від промислового об'єкта полягає не лише в масштабі, а й у характері інфраструктури та компетенціях обслуговуючого персоналу. Інженерні системи тут спрощені: замість прецизійних кондиціонерів – побутові або напівпромислові кліматичні установки, замість спеціалізованих стійок – монтажні рами або навіть відкриті полиці. Профільних фахівців з термодинаміки та систем охолодження, як правило, немає – обладнанням займається один-два системних адміністратори.

При цьому локальний ЦОД принципово не є статичним об'єктом: у міру зростання бізнесу він масштабується – додаються нові вузли, збільшується щільність навантаження, ускладнюється теплова карта приміщення. Кожне таке розширення підвищує вимоги до керування мікрокліматом, однак не завжди супроводжується відповідним розширенням інженерної інфраструктури або штату [6].

Ключова відмінність між промисловим та локальним ЦОД полягає у передбачуваності умов: великий об'єкт розробляється під конкретне теплове навантаження, має підготовлене приміщення і постійно контрольоване середовище. Локальний вузол, натомість, часто розгортається у звичайному офісному або підвальному приміщенні, де температура, вологість і запиленість повітря суттєво змінюються залежно від пори року, роботи вентиляції та інтенсивності використання. Взимку повітря може бути надмірно сухим через опалення, що підвищує ризик статичного розряду; влітку без якісного кондиціонування температура в приміщенні зростає разом із тепловим навантаженням кластера. Рівень запиленості залежить від будівлі та режиму

прибирання і жодним чином не регулюється автоматично.

У промисловому ЦОД з усіма цими факторами справляється спеціалізована інженерна інфраструктура та цілодобовий моніторинг. У локальному – ці фактори залишаються поза контролем, якщо не впроваджена відповідна система автоматизованого спостереження. Саме ця прогалина визначає предметну область цієї роботи: розробка системи, яка надає операторові локального GPU-кластера рівень інформованості та контролю, порівнянний з промисловим стандартом, але реалізований без великих капітальних витрат і складної інфраструктури. Ключові параметри мікроклімату, незалежно від масштабу ЦОД занесемо до таблиці 1.1.

Таблиця 1.1 – Параметри мікроклімату ЦОД

Параметр	Значущість для обладнання	Характер зміни у локальному ЦОД
Температура повітря	Безпосередньо визначає тепловий режим компонентів	Сезонні коливання, залежність від навантаження
Відносна вологість	Корозія при надлишку, статичний розряд при нестачі	Різка зміна при опаленні, дощах, сезонах
Концентрація пилу	Теплова ізоляція радіаторів, провідні містки на платах	Залежить від приміщення, не регулюється автоматично

Відхилення кожного з цих параметрів від допустимих меж запускає деструктивні процеси з різним характером і часовим масштабом [7] – від миттєвого статичного пробою до поступової корозії контактних площадок протягом місяців. Детальний аналіз цих механізмів наведено у наступних підрозділах.

1.2 IoT системи

Інтернет речей – це концепція побудови розподілених мереж фізичних пристроїв, оснащених сенсорами, мікроконтролерами та комунікаційними

модулями, що забезпечують збір даних із реального світу, їхню передачу та автоматизоване реагування на основі отриманої інформації. Сучасні IoT системи пройшли шлях від простих датчиків збору інформації до інтелектуальних екосистем, здатних самостійно аналізувати великі масиви даних та приймати рішення в режимі реального часу. Поняття введено дослідником Кевіном Ештоном у 1999 році і з того часу трансформувалося від теоретичної концепції до ключової технологічної парадигми [2].

Популяризація інтернету речей зумовлена розвитком мікроелектроніки та доступністю елементної бази. Сьогодні розробка базової IoT системи доступна не лише великим корпораціям, а й малим бізнесам або окремим розробникам завдяки поширенню бюджетних мікроконтролерів (Arduino, ESP, STM32) та відкритих стандартів комунікації. Це дозволяє створювати гнучкі, кастомізовані рішення під конкретні потреби, що раніше вимагало великих інвестицій у промислове обладнання.

IoT технології стають невід'ємною частиною як повсякденного життя, так і промислових процесів. Можна виділити кілька ключових напрямків:

– промисловий IoT (IIoT) та агрегація даних. У промисловості виникає критична задача збору інформації з тисяч вузлів виробничої лінії. Агрегація даних (температури двигунів, вібрації станків, споживання енергії) дозволяє не лише відстежувати поточний стан, а й будувати предиктивні моделі обслуговування. Це мінімізує простої обладнання та дозволяє виявити аномалії задовго до виходу техніки з ладу [1];

– повсякденне використання та Smart City. Системи «розумного будинку» автоматизують освітлення, опалення та безпеку, забезпечуючи енергоефективність. У масштабах міста IoT керує трафіком, моніторить стан навколишнього середовища та оптимізує роботу комунальних служб;

– агросектор та медицина. В сільському господарстві сенсори вологості ґрунту та аналізатори складу повітря в теплицях дозволяють автоматизувати полив та внесення добрив, підвищуючи врожайність [1,2]. У медицині носимі пристрої безперервно моніторять життєві показники пацієнтів, сигналізуючи про критичні

зміни. Класичний приклад архітектури такої системи відобразимо на рисунку 1.1.

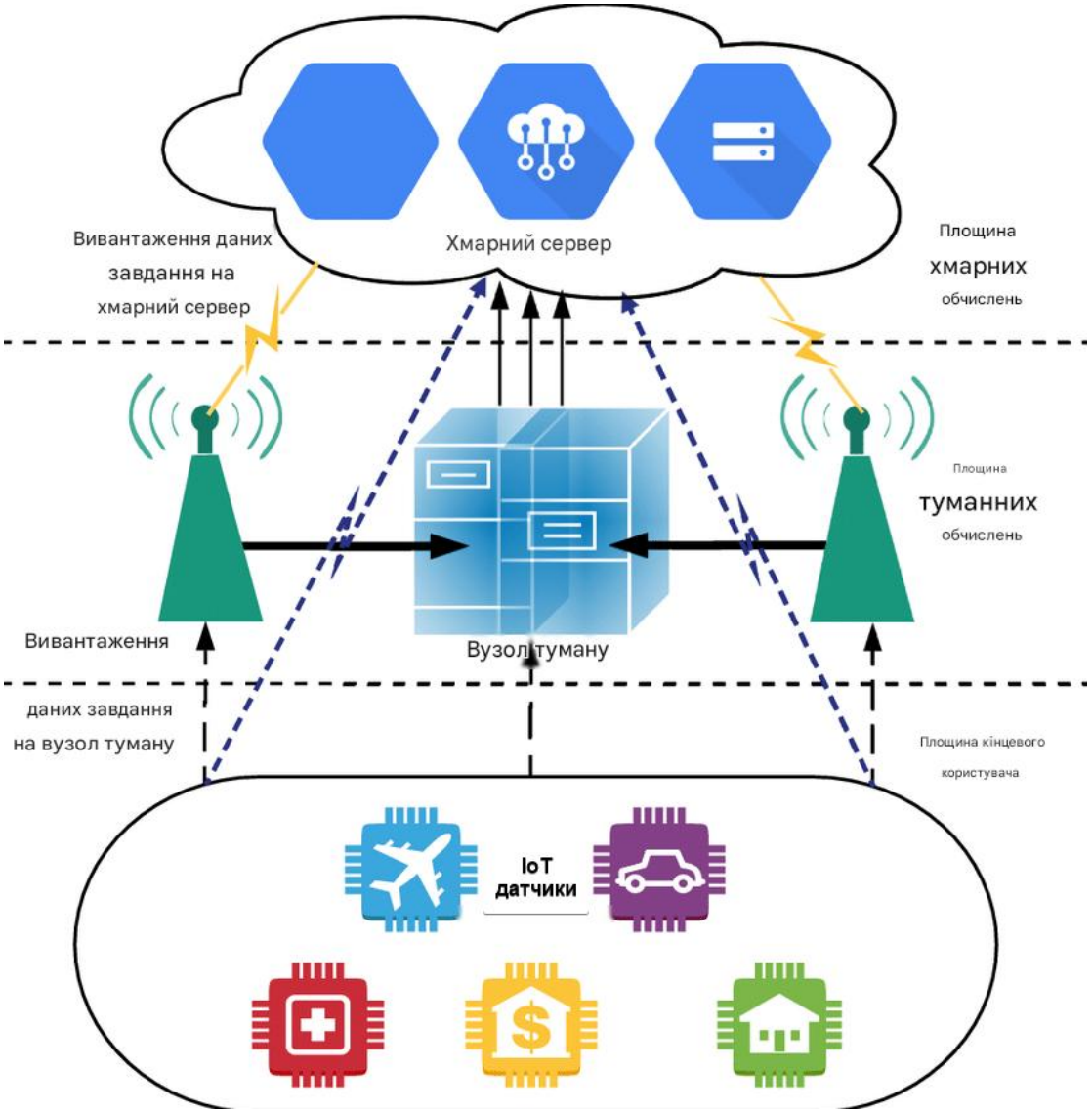


Рисунок 1.1 – Архітектура IoT системи розумної інфраструктури

Типова IoT-система складається з трьох функціональних рівнів:

– рівень сприйняття (Perception Layer) – сенсорні вузли, виконавчі механізми (актуатори) та крайові мікроконтролери, що безпосередньо взаємодіють із фізичним середовищем. На цьому рівні відбувається аналогово-цифрове перетворення сигналів та первинна фільтрація шумів;

– мережевий рівень (Network Layer) – комунікаційна інфраструктура для передачі даних між вузлами. Включає як дротові протоколи (Ethernet, RS-485, I2C,

SPI, UART), так і бездротові (Wi-Fi, Zigbee, LoRaWAN, Bluetooth Low Energy);

– рівень застосувань (Application Layer) – програмне забезпечення для агрегації, аналізу та візуалізації даних, реалізації бізнес-логіки та взаємодії з користувачем.

IoT система стає системою для прийняття інтелектуальних рішень для управління безліччю параметрів одночасно. Замість простого відображення цифр на екрані, система аналізує взаємозв'язки між різними показниками (наприклад, як вологість впливає на ефективність охолодження при певній запиленості) та впроваджує керуючі впливи.

В контексті дата-центрів, і особливо локальних GPU-кластерів, таке інтелектуальне управління дає наступні переваги:

– масштабованість. При додаванні нових обчислювальних потужностей IoT архітектура дозволяє легко інтегрувати нові сенсори та модулі управління в існуючу мережу без повної перебудови системи;

– автоматизація обслуговування. Система бере на себе рутинний моніторинг, звільняючи інженерів від необхідності постійної візуальної перевірки параметрів;

– оптимізація роботи. Автоматичне регулювання систем охолодження та зволоження на основі інтелектуальних алгоритмів дозволяє підтримувати ідеальний мікроклімат при мінімальних витратах електроенергії.

IoT система виступає фундаментом для побудови надійної інфраструктури, де апаратне забезпечення захищене не лише фізично, а й логічно – через безперервний інтелектуальний нагляд.

1.3 Системи обслуговування GPU-кластерів

До останніх подій халвінгу криптовалют значна частина високопродуктивних обчислювальних систем припадала на майнінг-ферми, попит на які невпинно зростав щороку. Характерною особливістю обслуговування таких

ферм була відносно висока відмовостійкість обчислювальних компонентів, що пояснювалося специфікою їхнього навантаження. Відеокарти працювали у статичному режимі, будучи завантаженими майже на 100% цілодобово, відстежувати аномалії було досить просто. Будь-яке зниження обчислювальної потужності або хешрейту окремого графічного процесора на тлі стабільної роботи інших одразу фіксувалося як явне відхилення від норми, що дозволяло операторам швидко реагувати на несправність.

Експлуатація сучасних GPU-кластерів для завдань штучного інтелекту базується на принципово іншій парадигмі – динамічному завантаженні. Розподіл завдань (генерація даних, обробка контексту) є нерівномірним та асинхронним.

Динамічний характер обчислень зумовлює нелінійне тепловиділення. GPU-кластер впливає на загальний температурний баланс приміщення, змінюючи термодинамічні характеристики навколишнього середовища. Конструкція графічних адаптерів включає матеріали з різними коефіцієнтами теплопровідності, що визначає наявність термічної інерції. Циклічні перепади температур під час зміни навантаження спричиняють систематичне теплове розширення та стискання елементної бази [15]. Цей процес виступає самостійним фактором збільшення коефіцієнта зносу компонентів. Додатковим наслідком асинхронного завантаження є виникнення імпульсних стрибків струму, які підвищують навантаження на кабельні лінії локального об'єкта. Запровадження гетерогенних кластерів, до складу яких входять графічні процесори різних архітектур, ускладнює теплову картину. Кожна модель характеризується специфічними температурними лімітами та показниками деградації матеріалів.

Ручний моніторинг описаних динамічних процесів є неефективним. Аналіз температурної інерції та превентивне виявлення апаратних аномалій в умовах кластера виходить за межі можливостей візуального контролю інженерного персоналу. Відсутність автоматизованих систем спостереження класифікується як критичний експлуатаційний ризик. Оптимальним інженерним рішенням для локальних інфраструктур є концепція зміщення зносу на необчислювальні елементи. Концепція передбачає компенсацію термодинамічних навантажень

шляхом автоматизованого управління зовнішніми системами мікроклімату. Впровадження такого підходу вирішує наступні задачі:

– захист інфраструктури живлення. Алгоритмічне згладжування енергетичних та теплових піків знижує ймовірність термічного пошкодження кабельних ліній. Архітектура дозволяє фізично розділити силові магістралі обчислювальних вузлів та систем кліматичного контролю;

– оптимізація експлуатаційних витрат. Спрямування амортизаційного навантаження на допоміжні механічні компоненти (вентилятори, фільтри) уповільнює деградацію високовартісних графічних кристалів;

– мінімізація апаратних ризиків. Збереження ресурсу обчислювальних модулів знижує залежність інфраструктури від логістичних обмежень у періоди дефіциту напівпровідникової продукції.

1.4 Архітектури IoT систем

1.4.1 Граничні обчислення

Ефективність функціонування системи інтернету речей безпосередньо залежить від обраної архітектурної парадигми. Архітектура визначає, на якому рівні ієрархії відбувається збір, обробка, зберігання даних та прийняття керуючих рішень. Загальна топологія сучасної IoT-мережі базується на багаторівневому підході, що дозволяє оптимізувати навантаження на канали зв'язку та мінімізувати затримки в контурах управління. Існує три основні парадигми обчислень в IoT: граничні (Edge), туманні (Fog) та хмарні (Cloud). Кожна з них має свої переваги, обмеження та цільове призначення. Для проектування надійної системи обслуговування локального GPU-кластера необхідно чітко розуміти функціональні межі кожної архітектури.

Граничні обчислення – це концепція, за якої первинна обробка інформації відбувається безпосередньо на периферійних пристроях, що знаходяться на «краю»

мережі, в безпосередній фізичній близькості до джерела даних (сенсорів).

Для збору та оцифрування аналогових чи цифрових сигналів із датчиків обов'язковою умовою є наявність мікроконтролера. Цей обчислювальний модуль виступає шлюзом між фізичним середовищем та цифровою мережею. Архітектура типового крайового вузла є вкрай спрощеною: мікроконтролер циклічно опитує сенсори, отримує сирі дані та виконує базові операції.

Обмеження граничних обчислень – мікроконтролери мають суттєві апаратні обмеження: невеликий обсяг вбудованої пам'яті, низька тактова частота та архітектура, яка не пристосована для багатопотокових або ресурсоємних обчислень. Їхньої обчислювальної потужності достатньо лише для найпростіших завдань:

– фільтрація апаратного шуму: видалення випадкових відхилень у показаннях сенсорів за допомогою базових математичних алгоритмів для отримання чистих метрик;

– локальна візуалізація: виведення відфільтрованих показників на прості локальні дисплеї в режимі реального часу. Ланцюг «сенсор – мікроконтролер – дисплей» є класичною гілкою, доступною для локальних обмежених обчислень.

Для управління мікрокліматом дата-центру виключно граничних обчислень абсолютно недостатньо. Ресурсів мікроконтролера не вистачить для розгортання алгоритмів машинного навчання чи предиктивної аналітики. Граничні обчислення закладають критично важливий фундамент для туманних обчислень. Наявність розгалуженої мережі сенсорних вузлів нівелює необхідність ручного збору даних операторами, делегуючи інженерам виключно функцію моніторингу результатів та прийняття високорівневих рішень.

1.4.2 Туманні обчислення

Туманні обчислення – це проміжний архітектурний рівень, який переносить значну частину обчислювальних потужностей, пам'яті та сервісів з централізованої хмари ближче до крайових пристроїв, безпосередньо в локальну мережу об'єкта.

Fog-нода (локальний обчислювальний вузол) здатна ефективно агрегувати сотні метрик на секунду від усіх мікроконтролерів системи. На цьому рівні

відбувається відстеження аномалій, фіксація критично високих або низьких значень параметрів та формування миттєвих керуючих команд для підвищення відмовостійкості обчислювальних компонентів кластера. Передаючи ці завдання на Fog-ноду, система знімає навантаження з граничних пристроїв, що дозволяє легко масштабувати дата-центр та адаптуватися до зовнішніх умов без ускладнення алгоритмів на мікроконтролерах.

Туманні обчислення дозволяють імплементувати алгоритми на базі Machine Learning для управління мікрокліматом. Наприклад, якщо система виявляє підвищення вологості понад 40%, простий тригер відразу увімкнув би осушувач. Натомість ML-алгоритм враховує інерцію мікроклімату: він аналізує тренди зростання вологості за попередні періоди і приймає рішення з урахуванням прогнозованої динаміки, запобігаючи постійному "смиканню" інженерного обладнання.

Fog-нода локального дата-центру має обмежені ресурси для тривалого зберігання терабайтів даних. Виключно туманної архітектури недостатньо для глибокої візуалізації в реальному часі, довгострокового відстеження трендів та паралельного навчання нових ML-моделей – це швидко призведе до перевантаження локального сервера.

1.4.3 Хмарні обчислення

Хмарні обчислення – це парадигма надання масштабованих обчислювальних ресурсів за вимогою через інтернет. Глобальні лідери індустрії, такі як AWS або Microsoft Azure, будують гігантські платформи з величезним набором сервісів. Однак для малого бізнесу або компактних локальних дата-центрів використання повноцінних комерційних хмар для безпосереднього управління апаратним забезпеченням є нерентабельним.

Головними недоліками хмарного керування в контексті мікроклімату є високі регулярні витрати, надлишковість функціоналу та висока мережева затримка (Latency). Для динамічних параметрів, таких як температура GPU, яка може злетіти до критичних значень за лічені секунди, затримка передачі сигналу на віддалений сервер і назад робить своєчасне запобігання перегріву неможливим. Затримка

нівелює переваги хмарних алгоритмів.

Незважаючи на непридатність для прийняття реактивних керуючих рішень, хмара є ідеальним місцем для довгострокового зберігання (бази даних часових рядів) та глибокої аналітики. Розвантаження туманного рівня шляхом перенесення даних у хмару дозволяє локальній Fog-ноді працювати тривалий час без апаратного розширення.

У хмарному середовищі зручно реалізовувати:

– базу даних для довгострокового зберігання історії всіх параметрів (Big Data);

– глобальну веб-систему для графічної візуалізації даних, ручного управління та моніторингу;

– швидку агрегацію телеметрії для відстеження багатомісячних трендів;

– запуск симуляцій різних кліматичних умов для тестування та підготовки до масштабування дата-центру.

1.5 Огляд існуючих аналогів

Проектування архітектури гібридної IoT-системи та визначення технічних параметрів мережевої взаємодії, затримок передачі даних, пропускної здатності та обсягів телеметрії потребує аналізу існуючих промислових рішень. Аналіз систем-аналогів дозволяє визначити ефективні підходи до автоматизованого керування мікрокліматом серверної інфраструктури та адаптувати їх для локальних GPU-кластерів малого й середнього масштабу.

Система керування охолодженням дата-центрів Google базується на використанні алгоритмів машинного навчання та багаторівневої IoT-архітектури. Телеметрія збирається з великої кількості сенсорів із періодичністю близько п’яти хвилин. До складу телеметричних даних входять температурні показники, швидкість роботи насосів, параметри вентиляційних систем та інші технічні

метрики. Дані передаються у хмарне середовище для подальшої обробки нейронними мережами (Deep Neural Networks), які виконують прогнозування температурних змін та рівня енергоспоживання. Після завершення аналізу сформовані керуючі команди передаються до локального вузла керування (Edge-рівень), де додатково перевіряються локальними алгоритмами безпеки перед застосуванням до обладнання.

Недоліки підходу для локального GPU-кластера:

– передача великого обсягу телеметрії до хмарного середовища та подальша обробка нейронними мережами створює значну затримку між моментом зміни температури та виконанням керуючої дії. Для локального GPU-кластера, у якому температура кристала може досягати критичних значень протягом кількох секунд, затримка хмарної обробки є неприйнятною;

– обробка великої кількості метрик із використанням моделей машинного навчання потребує значних обчислювальних ресурсів. Для локальної серверної інфраструктури малого масштабу використання подібної архітектури є економічно недоцільним.

Платформа AWS IoT Greengrass реалізує модель периферійних обчислень (Edge Computing) із перенесенням логіки обробки даних безпосередньо на локальні пристрої. Локальне виконання алгоритмів керування (Local Inference) дозволяє зменшити затримку реакції системи порівняно з централізованими хмарними рішеннями. Для оптимізації передачі телеметрії використовується механізм буферизації та пакетування даних (Stream Manager), що дозволяє передавати інформацію до хмарного середовища агрегованими блоками без перевантаження мережевого каналу.

Недоліки підходу для локального застосування:

– використання AWS IoT Greengrass створює залежність інфраструктури від комерційної екосистеми Amazon (Vendor Lock-in);

– експлуатація системи потребує постійного використання хмарних сервісів AWS IoT Core, CloudWatch та мережевих ресурсів, що збільшує операційні витрати локального дата-центру.

Формування архітектурних та технічних вимог до IoT-систем безпосередньо залежить від специфіки їх застосування. Згідно з дослідженнями продуктивності систем інтернету речей у різних галузях, зокрема в агропромисловому та індустріальному секторах, вибір парадигми обчислень кардинально впливає на кінцеві метрики системи. Наприклад, в агрокультурі для довгострокового аналізу ґрунту ефективною є Cloud-архітектура через потребу накопичення великих обсягів Big Data. Натомість для індустріального IoT (IIoT), де критичною є миттєва реакція виконавчих механізмів (actuators), гібридне поєднання туманних та граничних обчислень дозволяє гарантовано знизити затримку передачі даних з типових хмарних 200-300 мс до значень менше 50-100 мс. Окрім того, локальне фільтрування даних на периферії значно зменшує навантаження на пропускну здатність мережі (throughput) та скорочує використання енергії.

1.6 Постановка проблеми та обґрунтування обраної IoT-архітектури

Функціонування локальних обчислювальних інфраструктур, зокрема GPU-кластерів, характеризується високою динамікою зміни термодинамічних параметрів. Відсутність спеціалізованого промислового кліматичного обладнання у приміщеннях невеликих дата-центрів призводить до того, що фактори навколишнього середовища безпосередньо впливають на експлуатаційний ресурс апаратного забезпечення.

Основна проблема у відсутності автоматизованого інструментарію для відстеження та компенсації температурних інерцій, коливань вологості та накопичення пилу в умовах динамічного обчислювального навантаження. Ручне керування допоміжними кліматичними системами є неефективним і не здатним запобігти процесам прискореної деградації електронних компонентів [6].

Для формалізації задачі визначено ключові фактори впливу, алгоритмічний контроль яких є обов'язковим для забезпечення стабільної роботи кластера та

передбачені деградації компонентів та аномалій, які занесено до таблиці 1.2.

Таблиця 1.2 – Фактори впливу на GPU-кластер

Фактор впливу	Фізичний наслідок для обладнання	Вимоги до часу реакції системи	Наслідок ігнорування проблеми
Температура (динамічні перепади)	Термічне розширення та стискання матеріалів (інерція)	Критичний (мілісекунди / секунди)	Деградація пайки BGA-компонентів, термічний пробій транзисторів живлення.
Відносна вологість	Зміна діелектричної проникності середовища	Помірний (хвилини / години)	Корозія струмопровідних доріжок (при надлишку), статичний розряд (при нестачі).
Концентрація пилу	Зниження ефективності тепловідведення радіаторів	Довгостроковий (дні / тижні)	Утворення теплоізоляційного шару, локальний перегрів, замикання контактів.

Вирішення описаної проблеми вимагає розробки комплексної системи моніторингу та керування. Ключовим інженерним завданням на початковому етапі проектування є вибір оптимальної топології обчислень. Враховуючи гетерогенні вимоги до часу реакції на зміну різних мікрокліматичних параметрів, необхідно проаналізувати застосовність базових парадигм IoT для управління дата-центром.

Аналіз підтверджує, що жодна з базових архітектур в ізольованому вигляді не здатна комплексно задовольнити технічні вимоги:

– імплементація виключно граничних обчислень не дозволяє реалізувати логіку прийняття рішень через жорсткі апаратні обмеження мікроконтролерів;

– ізольовані хмарні обчислення формують ризик мережевої затримки, що є

критичним для запобігання різким температурним аномаліям. Це робить експлуатаційну безпеку локального об'єкта залежною від безперебійного підключення до Інтернету;

– туманні обчислення здатні забезпечити алгоритмічну автономність та швидкість реакції, проте обмеження місткості локальних серверів перешкоджають тривалому накопиченню телеметрії для графічного аналізу та розгортанню зручного веб-середовища.

Порівняльний аналіз архітектур наведено у таблиці 1.3.

Таблиця 1.3 – Порівняльний аналіз архітектур IoT систем

Критерій порівняння	Гранична (Edge)	Туманна (Fog)	Хмарна (Cloud)	Гібридна (Fog + Cloud)
Мережева затримка (Latency)	Мінімальна (<1 мс)	Низька (1-10 мс)	Висока (50-200+ мс)	Адаптивна (локально <10 мс, глобально >50 мс)
Стійкість до втрати зв'язку	Повна автономність	Повна автономність	Повна втрата керованості	Автономне керування зберігається
Розподіл логіки та обчислень	Відсутній (лише базові)	Локальна аналітика	Візуалізація та ручне упр.	Оптимальний (Fog - упр., Cloud - віз.)
Зберігання даних (Big Data)	Неможливе	Короткострокове	Довгострокове (БД)	Довгострокове (БД у хмарі)
Економічна доцільність	Висока	Висока	Низька (залежність від трафіку)	Оптимальна (баланс ресурсів)

Відповідно, єдиним технічно обґрунтованим рішенням є імплементація гібридної хмарно-туманної архітектури. У такій структурі крайові пристрої у ролі Edge-шлюзів виконують функцію первинного збору даних та апаратної фільтрації шумів. Вузол туманного рівня (локальний обчислювальний сервер) виступає ядром керування: він приймає відфільтровані дані, виконує пошук аномалій за критичними порогами, реалізує алгоритми підтримки мікроклімату та генерує

керуючі впливи. Усі локальні процеси та системні сповіщення локалізуються на цьому рівні, гарантуючи стійкість дата-центру до відмови зовнішніх мереж. Хмарний рівень розвантажує локальний сервер від завдань накопичення даних та візуалізації. У хмарі розміщується база даних часових рядів та глобальна веб-система. Веб-середовище слугує платформою для графічного відображення, аналізу трендів, симуляції умов для підготовки до масштабування та ініціації команд ручного управління інженером. Важливо, що ініційовані в хмарі команди не керують апаратним забезпеченням напряму, а передаються на туманний вузол для подальшої диспетчеризації локальними алгоритмами. Вибір гібридної архітектури з таким розподілом ролей формує концептуальну основу для проектування системи, мінімізуючи експлуатаційні ризики для локальних GPU-кластерів. Визначення конкретних вимог до такої системи є предметом наступного розділу.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Визначення та аналіз вимог до системи

2.1.1 Вимоги до апаратного рівня

Виходячи з результатів досліджень [1], вимоги до розроблюваної системи керування мікрокліматом GPU-кластера були відповідним чином адаптовані. Ми імплементували промисловий підхід Edge/Fog-обчислень для мінімізації трафіку та досягнення швидкої реакції на перегрів, проте згладили апаратні вимоги до серверного ядра, адаптувавши їх під використання бюджетних відкритих технологій, що робить рішення рентабельним для малого бізнесу.

Визначення конкретних вимог здійснюється на основі аналізу предметної області (Розділ 1) та операційних сценаріїв кожного з модулів. Аналіз розглянутих систем у розділі 1.5 демонструє, що логіка керування охолодженням повинна розміщуватись локально на рівні Fog-сервера. Затримка обробки даних повинна

залишатись мінімальною, оскільки використання виключно хмарної обробки призводить до запізнення реакції системи та виникнення теплового тротлінгу GPU до моменту активації алгоритмів охолодження.

Практика використання великої кількості телеметричних метрик та локальної буферизації даних підтверджує доцільність агрегування інформації перед передачею мережею. Передача телеметрії невеликими пакетами дозволяє зменшити навантаження на мережеву інфраструктуру та забезпечити стабільну роботу системи зберігання часових рядів.

Розроблюваний програмно-апаратний комплекс базується на відкритому стеку технологій. Використання відкритих програмних компонентів дозволяє реалізувати локальну автономність Fog-рівня, масштабовану аналітику та інтеграцію з хмарними сервісами без використання комерційних платформ і без залежності від окремого постачальника технологій.

До апаратного рівня належать сенсорні модулі, виконавчі механізми та Edge-шлюз.

Функціональні вимоги:

– безперервне циклічне зчитування температури кожного GPU та температури навколишнього повітря;

– зчитування відносної вологості повітря та концентрації дисперсного пилу у приміщенні;

– зчитування поточного рівня завантаженості кожного обчислювального вузла;

– апаратна фільтрація шумів аналогових сигналів до формування пакету телеметрії;

– формування структурованого пакету телеметрії та його циклічна передача до туманного сервера;

– прийом керуючих команд від туманного сервера та їх виконання: генерація ШІМ-сигналів для вентиляторів, комутація кліматичного обладнання;

– збереження останніх прийнятих команд для автономної роботи у разі втрати зв'язку.

Нефункціональні вимоги:

– апаратна придатність до роботи в умовах підвищеної температури та запиленості приміщення;

– забезпечення частоти передачі телеметрії, достатньої для своєчасного прийняття рішень туманним сервером;

– підтримка підключення сенсорного рівня для щонайменше 16 обчислювальних вузлів без зміни архітектури;

– сумісність з програмним емулятором: архітектура повинна допускати повну заміну реального апаратного рівня програмним аналогом через єдиний мережевий інтерфейс без модифікацій туманного сервера.

2.1.2 Вимоги до туманного сервера

Туманний сервер є центральним виконавчим вузлом системи, розміщеним у локальній мережі поруч із кластером.

Функціональні вимоги:

– прийом, структурна валідація та маршрутизація вхідних пакетів телеметрії від Edge-шлюзів;

– запис усіх метрик у базу даних часових рядів з прив'язкою до мітки часу та ідентифікатора пристрою;

– реалізація алгоритму адаптивного охолодження: розрахунок ШІМ-сигналу вентилятора на основі температури GPU, рівня завантаженості та теплового тренду з урахуванням гістерезису;

– реалізація випереджального керування: підвищення оборотів при прогнозованому зростанні навантаження до досягнення критичного температурного порогу;

– реалізація алгоритму контролю мікроклімату: активація зволожувача або осушувача залежно від комбінації показників вологості та пилу;

– розрахунок кумулятивних індексів зносу: індексу ризику корозії та індексу зносу вентиляторів;

– генерація подій сповіщень трьох рівнів критичності: критичний, попереджувальний та екологічний;

– підтримка режимів роботи автоматичного та ручного з перемиканням без перезапуску сервера;

– надання програмного інтерфейсу (API) для веб-системи та для Edge-шлюзів.

Нефункціональні вимоги:

– розміщення у локальній мережі для мінімізації мережевої затримки між шлюзом та сервером;

– асинхронна обробка вхідних запитів без блокування циклу виконання;

– збереження функціональності алгоритмів керування при недоступності хмарної бази даних;

– масштабованість: підключення нових Edge-шлюзів без модифікації ядра системи;

– розгортання у контейнерному середовищі для відтворюваності та ізоляції залежностей;

– автогенерація документації API без ручного оновлення.

2.1.3 Вимоги до веб-системи

Веб-система забезпечує глобальний доступ оператора та інженера до даних і керування.

Функціональні вимоги:

– відображення поточного стану кожного GPU (температура, завантаженість, ШІМ, оберти) у вигляді дашборду з автоматичним оновленням;

– відображення поточних показників мікроклімату та стану кліматичного обладнання;

– перегляд хронологічного журналу сповіщень усіх рівнів критичності;

– побудова графіків часових рядів за обраним діапазоном з підтримкою накладання різних параметрів на одному полотні;

– відображення динаміки індексів корозії та зносу вентиляторів для підтримки предиктивного обслуговування;

– ініціація команд ручного управління: перемикання режимів, встановлення ШІМ окремих вентиляторів, вибір профілю середовища;

– передача всіх керуючих команд виключно через туманний сервер; пряма взаємодія з апаратним рівнем відсутня.

Нефункціональні вимоги:

– автоматичне оновлення даних без перезавантаження сторінки;

– коректне відображення на різних роздільних здатностях екрану;

– строга типізація структур даних API-відповідей для запобігання помилкам інтеграції;

– чітка URL-маршрутизація для кожної функціональної сторінки.

2.2 Розробка специфікації вимог SyRS

2.2.1 Введення

2.2.1.1 Призначення

Документ є специфікацією системних вимог до комплексу програмно-апаратного моніторингу мікроклімату та адаптивного управління охолодженням GPU-кластера. Документ призначений для використання розробниками програмного забезпечення, проєктувальниками апаратної частини та інженерами з налагодження системи.

2.2.1.2 Область дії

Програмно-апаратний комплекс має назву «Система інтелектуального моніторингу та охолодження GPU-кластера». Система призначена для безперервного збору параметрів мікроклімату (температура, вологість, запиленість) у приміщенні локального дата-центру, автоматичного управління зовнішніми вентиляторами та кліматичним обладнанням, зберігання телеметрії та надання оператору засобів моніторингу і ручного управління через веб-інтерфейс. Система забезпечує контроль кластера до 16 графічних процесорів з можливістю маштабування та мінімізує необхідність фізичної присутності персоналу у серверному приміщенні.

2.2.2 Загальний опис

2.2.2.1 Контекст програмного продукту

Система не є автономним продуктом, а функціонує як трирівневий апаратно-програмний комплекс. Апаратний рівень збирає метрики з датчиків та передає їх на туманний сервер; туманний сервер виконує алгоритми керування та надсилає команди назад до апаратного рівня; хмарний рівень зберігає телеметрію та надає оператору веб-інтерфейс. Команди, ініційовані оператором через веб-інтерфейс, не взаємодіють з апаратним рівнем напряму, а передаються виключно через туманний сервер.

Система також передбачає режим програмного емулятора апаратного рівня, що дозволяє тестувати алгоритми туманного сервера без підключення реального обладнання та симулювати різні кліматичні умови для підготовки до масштабування.

2.2.2.2 Функції програмного продукту

Основними функціями системи є:

– моніторинг у реальному часі. Безперервний збір температури кожного GPU, температури приміщення, відносної вологості та концентрації пилу з відображенням на дашборді оператора;

– адаптивне охолодження. Автоматичний розрахунок і подача ШІМ-сигналів вентиляторів на основі теплового стану кожного GPU, рівня завантаженості та алгоритму гістерезису;

– контроль мікроклімату. Автоматична активація осушувача або зволожувача повітря залежно від комбінації вологості та запиленості з метою запобігання корозії та статичним розрядам;

– аналітика зносу. Безперервне обчислення індексів CI та FWI для реалізації концепції предиктивного обслуговування;

– система сповіщень. Генерація та доставка сповіщень трьох рівнів: критичного (загроза перегріву), попереджувального (підвищена температура) та екологічного (пил або вологість поза нормою);

– ручне управління. Надання інженеру можливості перемикання між

автоматичним та ручним режимами, встановлення ШІМ окремих вентиляторів, симуляції кліматичних сценаріїв;

– архів та тренди. Зберігання телеметрії та відображення графіків за обраний часовий діапазон, у тому числі накладання температури та ШІМ для аналізу роботи алгоритму гістерезису.

2.2.2.3 Характеристики користувача

Система орієнтована на дві категорії користувачів. Оператор – персонал з базовим технічним рівнем, що здійснює моніторинг стану обладнання та реагує на сповіщення, орієнтуючись на дашборд. Інженер-адміністратор – фахівець з глибоким розумінням термодинаміки обчислювальних систем, що аналізує тренди зносу, налаштовує режими роботи, виконує ручне управління та готує систему до масштабування.

2.2.2.4 Обмеження

Апаратні обмеження:

– датчики температури мають апаратну похибку вимірювання; крайовий шлюз повинен враховувати шум при формуванні пакету;

– виконавчі пристрої (вентилятори, осушувач, зволожувач) підключаються до Edge-шлюзу через реле та ШІМ-виходи; напруга живлення та струмове навантаження визначають вибір відповідних виконавчих модулів;

– edge-шлюз повинен підтримувати один або кілька послідовних інтерфейсів зв'язку з сенсорами (I2C, UART, SPI, 1-Wire) та дротовий або бездротовий інтерфейс для передачі даних у локальну мережу;

– компоненти апаратного рівня повинні зберігати працездатність при температурі навколишнього середовища до 55°C та у запиленому середовищі.

Мережеві обмеження:

– затримка у локальній мережі між Edge-шлюзом та туманним сервером не повинна перевищувати 100 мс для стабільної роботи алгоритму охолодження;

– пропускна здатність мережі повинна бути достатньою для передачі JSON-пакетів телеметрії розміром до 10 КБ з частотою не менше одного разу на 30 секунд;

– при недоступності туманного сервера або хмарного рівня відповідна підсистема продовжує роботу в автономному режимі.

Програмні обмеження:

– конфігурація порогів, кількості GPU та адрес сервісів зберігається у зовнішніх файлах конфігурації і не вимагає зміни вихідного коду;

– туманний сервер повинен функціонувати ідентично незалежно від джерела телеметрії – реального апаратного рівня або програмного емулятора;

2.2.2.5 Допущення та залежності

Передбачається стабільне мережеве з'єднання між усіма компонентами системи. Ефективність алгоритму охолодження залежить від температури навколишнього повітря у приміщенні. Зміни у протоколі обміну між апаратним рівнем та туманним сервером можуть вплинути на обидві сторони, тому архітектура передбачає фіксовану схему JSON-пакетів. Система залежить від безперервної роботи туманного сервера: його відмова унеможливлює автоматичне керування охолодженням.

2.2.3 Специфічні вимоги

2.2.3.1 Зовнішні інтерфейси

Апаратні інтерфейси. Датчики температури підключаються до Edge-шлюзу через один з послідовних протоколів (I2C, 1-Wire або UART). Датчик вологості та пилу підключається через I2C або UART залежно від обраної моделі. Для управління вентиляторами використовуються ШІМ-виходи мікроконтролера (0–100%). Кліматичне обладнання (осушувач, зволожувач) комутується через релейні виходи. Edge-шлюз підключається до локальної мережі через дротовий (Ethernet) або бездротовий (Wi-Fi 2.4/5 ГГц) інтерфейс.

Програмні інтерфейси. Взаємодія між Edge-шлюзом та туманним сервером, а також між веб-системою та туманним сервером здійснюється за протоколом HTTP з використанням архітектурного стилю REST. Тіло запитів і відповідей – формат JSON. Як альтернатива може розглядатися протокол MQTT для асинхронної публікації телеметрії.

Інтерфейс користувача. Веб-система реалізується як багатосторінковий веб-

додаток, що функціонує у сучасних браузерах. Основні сторінки: головний дашборд, сторінка мікроклімату, журнал подій, сторінка аналізу архівних даних, сторінка трендів деградації, панель адміністрування. Стан обладнання позначається кольоровою індикацією відповідно до рівня критичності.

2.2.3.2 Функціональні вимоги

Апаратний рівень:

– інтервал зчитування сенсорів: не рідше 1 разу на 5 секунд;

– інтервал відправки пакету телеметрії: не рідше 1 разу на 30 секунд;

– температурний діапазон сенсорів: від −10°C до +150°C; абсолютна похибка – не більше ±0,5°C; роздільна здатність – не гірше 0,1°C; рівень власного шуму (σ) – не більше 0,3°C;

– діапазон вимірювань вологості: 0–100% RH; концентрації пилу: 0–500 мкг/м³;

– максимальна кількість GPU в одному кластерному вузлі: 16;

– при втраті зв'язку з туманним сервером: утримання останніх прийнятих ШІМ-команд або перехід у безпечний режим (PWM = 100%).

Туманний сервер:

– час від прийому запиту телеметрії до відправки відповіді з керуючими командами: не більше 200 мс;

- базова крива залежності ШІМ від температури GPU: лінійна по сегментах (до 30°C – 20%; 30–50°C – 20–40%; 50-70°C – 40-70%; 70–85°C – 70–100%; понад 85°C - 100%);

– при навантаженні GPU понад 80% – мінімальний ШІМ не нижче 50% (випереджальне керування) ;

– гістерезис: зниження ШІМ дозволяється не раніше ніж через 60 секунд після стабілізації температури; крок зниження – не більше 10% за один цикл;

– пороги сповіщень: критичний – температура GPU ≥ 120°C; попереджувальний – ≥ 90°C; екологічний – пил > 50 мк/м³ або вологість поза діапазоном 40–60% протягом понад 1 години;

– активація зволожувача: при RH < 40% у поєднанні з пилом > 30 мкг/м³;

– активація осушувача: при RH > 60% у поєднанні з пилом > 30 мкг/м³;

– при значенні індексу корозії CI > 1,5 базовий ШІМ компенсаційно збільшується; максимальний штраф до ефективності охолодження – 4% при CI = 3,0.

Веб-система:

– автоматичне оновлення даних дашборду: не рідше ніж кожні 5 секунд;

– для кожного GPU відображаються: температура з точністю до 0,1°C, рівень завантаженості, значення ШІМ та RPM;

– графіки часових рядів підтримують вибір довільного часового діапазону та накладання щонайменше двох параметрів на одному полотні;

– команди ручного управління (режим, ШІМ) передаються на туманний сервер; час відгуку інтерфейсу на дію оператора – не більше 200 мс.

2.2.3.3 Вимоги до бази даних

База даних часових рядів повинна забезпечувати:

– потокову запис телеметрії з частотою щонайменше 100 точок на хвилину;

– тегування записів для фільтрації за ідентифікатором пристрою та номером GPU;

– мову запитів з підтримкою агрегації (середнє, мінімум, максимум) та фільтрації за часовим діапазоном;

– можливість розгортання у контейнерному середовищі;

– зберігання повної телеметрії щонайменше 30 діб; агрегованих даних – без обмеження терміну.

Структура вимірювань повинна охоплювати: температури GPU та приміщення; стан вентиляторів (ШІМ, RPM); показники мікроклімату (вологість, пил); стан кліматичного обладнання; події сповіщень; значення CI та FWI.

2.2.3.4 Атрибути якості системи

Система продовжує виконання критичних функцій охолодження при втраті з'єднання з будь-яким рівнем, вищим за поточний. Некоректні вхідні пакети відхиляються із поверненням структурованої помилки без впливу на стан системи.

Токени доступу до бази даних зберігаються у змінних середовища поза

репозиторієм вихідного коду. Всі вхідні пакети підлягають схемній валідації до обробки.

Туманний сервер та база даних розгортаються у контейнерному середовищі. Зміна порогів, кількості GPU або адрес сервісів здійснюється через файл конфігурації без перезапуску вихідного коду.

Туманний сервер автоматично генерує та надає інтерактивну документацію API без ручного оновлення.

2.3 Структура системи

Концептуальна архітектура системи базується на гібридній хмарно-туманній моделі з чітким розподілом обчислювального навантаження між апаратним, туманним та хмарним рівнями. Такий підхід забезпечує мінімальну затримку при виконанні критичних функцій охолодження, зберігаючи при цьому можливості глобального моніторингу та зберігання великих обсягів історичних даних. Фізична та логічна взаємодія компонентів системи представлена на рисунку 2.1 у вигляді діаграми концептуальної архітектури.

Апаратний рівень є фізичним фундаментом системи, що розміщується безпосередньо у приміщенні з обчислювальними вузлами. Логічно він складається з трьох блоків: сенсорного, виконавчого та обчислювального (Edge-шлюз). Сенсори безперервно збирають інформацію про температурний стан GPU та мікроклімат приміщення, передаючи сирі дані на Edge-шлюз через апаратні інтерфейси зв'язку. Edge-шлюз виконує роль локального агрегатора: він здійснює первинну цифрову фільтрацію шумів датчиків, формує єдиний структурований пакет телеметрії та маршрутизує його до туманного рівня через локальну мережу. У зворотному напрямку шлюз отримує абстрактні команди та транслює їх у фізичні сигнали (ШІМ-модуляція для вентиляторів та релейне замикання для кліматичного обладнання). Фізична близькість цих компонентів до об'єкта керування забезпечує

миттєву реакцію апаратної частини на логічні рішення сервера.

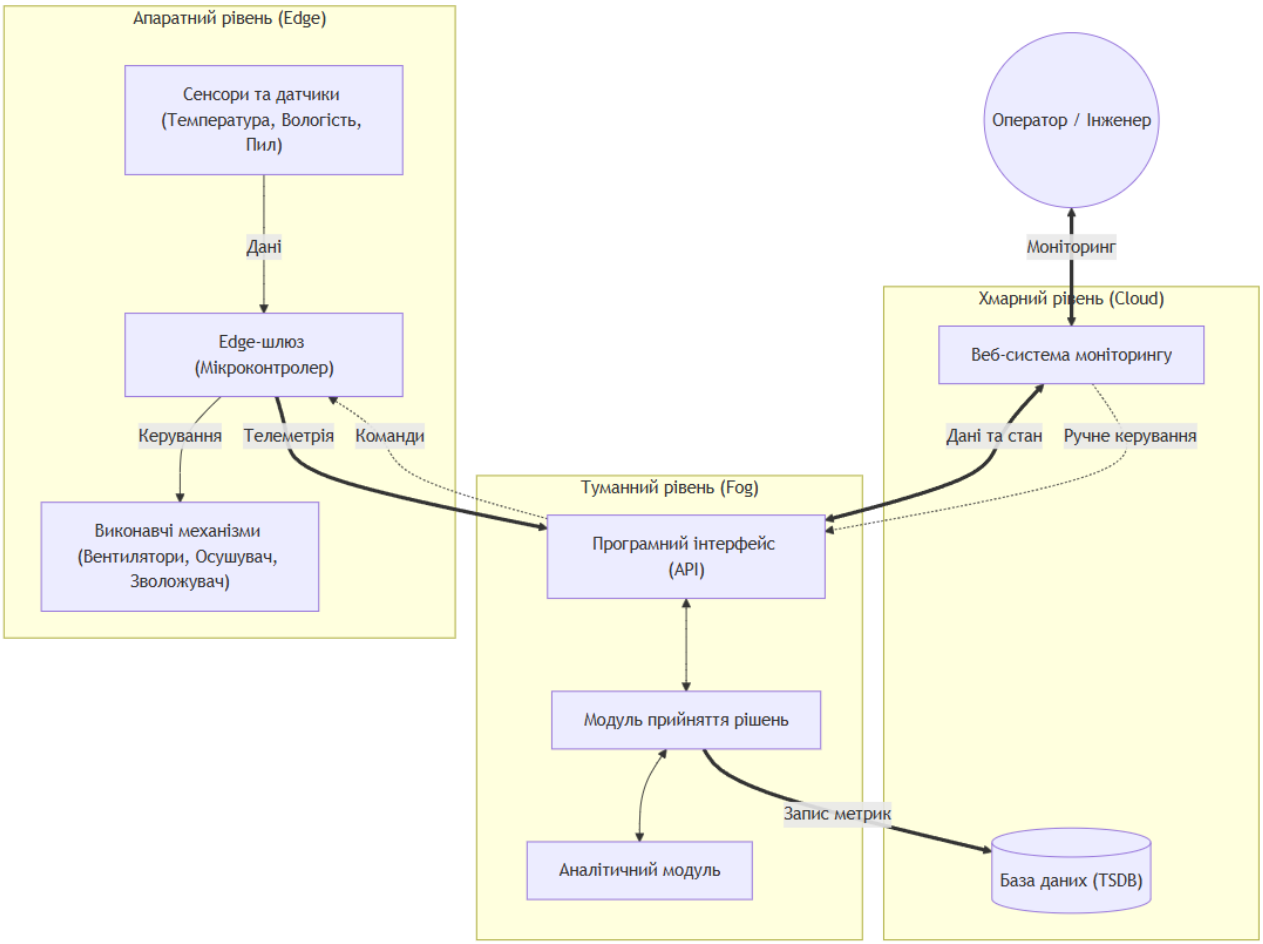


Рисунок 2.1 – Концептуальна архітектура взаємодії рівнів системи

Туманний рівень виступає центральним ядром прийняття рішень. Фізично він реалізований як локальний обчислювальний вузол (сервер), підключений до тієї ж локальної мережі, що й апаратний рівень. Логічно туманний рівень інкапсулює критичну бізнес-логіку системи. Завдяки розміщенню "на краю" мережі (Edge-to-Fog взаємодія), мережеві затримки зведені до мінімуму. Через абстрактний програмний інтерфейс (API) модуль прийняття рішень приймає вхідну телеметрію, проводить її структурну валідацію і застосовує алгоритми адаптивного охолодження та контролю мікроклімату. Враховуючи випереджальне керування та гістерезис, модуль генерує нові керуючі команди для шлюзу. Паралельно

аналітичний модуль обчислює кумулятивні індекси зносу елементної бази. Всі оброблені та валідовані дані безперервно транслюються на хмарний рівень для довгострокового збереження.

Хмарний рівень забезпечує глобальну доступність даних, їх архівування та надання візуального інтерфейсу користувачам. Фізично рівень може розміщуватися на віддалених серверах і логічно складається з двох основних підсистем: бази даних часових рядів (TSDB) та веб-системи. База даних забезпечує високоінтенсивний потоковий запис телеметрії від туманного сервера та виконання агрегаційних запитів для побудови історичних графіків. Веб-система надає графічний інтерфейс для моніторингу та ручного керування системою.

Важливою архітектурною особливістю взаємодії є відсутність прямих логічних чи фізичних зв'язків між веб-системою та апаратним рівнем. Вся взаємодія здійснюється виключно через програмний інтерфейс туманного сервера. Це гарантує, що при відмові веб-інтерфейсу або втраті глобального інтернет-з'єднання локальний контур керування (Edge-Fog) автономно продовжить функціонувати в режимі автоматичної стабілізації температур, гарантуючи безпеку обчислювального обладнання.

Модульна логічна структура дозволяє незалежно замінювати або масштабувати кожен рівень, наприклад, використовувати програмний емулятор замість реального апаратного рівня без жодних змін у туманній або хмарній логіці.

Додатково така архітектура спрощує забезпечення інформаційної безпеки та відмовостійкості системи. Оскільки критичні алгоритми керування та прийняття рішень локалізовані на туманному рівні, зовнішній доступ до апаратного контуру може бути повністю ізольований або обмежений лише через контрольовані API-запити. Це знижує ризики несанкціонованого втручання у процес керування охолодженням та мінімізує залежність від стабільності зовнішньої мережевої інфраструктури.

Крім того, розподіл функцій між рівнями дозволяє оптимізувати навантаження: оперативна обробка телеметрії та реакція на критичні події виконуються локально в реальному часі, тоді як ресурсоємна аналітика,

довгострокове зберігання та візуалізація переносяться на хмарний рівень.

2.4 Функціонування системи

У системі виділено чотири актори – два зовнішніх (люди) та два системних (програмно-апаратних), яких ми відобразили у таблиці 2.1.

Таблиця 2.1 – Опис акторів системи

Актор	Тип	Опис
Оператор	Зовнішній	Персонал з базовою кваліфікацією. Спостерігає за дашбордом, реагує на сповіщення. Не має доступу до функцій ручного управління.
Інженер	Зовнішній	Фахівець з глибоким технічним рівнем. Має повний доступ до системи: перемикання режимів, ручне встановлення ШІМ, вибір профілю середовища, аналіз трендів зносу.
Edge-шлюз	Системний	Мікроконтролер, який циклічно зчитує дані сенсорів, формує пакет телеметрії, відправляє його на туманний сервер та виконує отримані керуючі команди.
Fog-сервер	Системний	Центральний обчислювальний вузол. Приймає телеметрію, виконує алгоритми керування, записує дані у TSDB, формує сповіщення та надає API для веб-системи.

Наведемо перелік прецедентів системи з описом кожного за структурою: назва, первинний актор, передумова, основний потік та результат.

Прецедент П1. Зчитати та передати температурну телеметрію

Актор: Edge-шлюз.

Передумова: Edge-шлюз увімкнений та має мережеве з'єднання з Fog-сервером.

Основний потік:

– edge-шлюз циклічно зчитує температуру кожного GPU та температуру приміщення з інтервалом 5 секунд;

–для кожного GPU зчитується поточний рівень завантаженості;

– виконується апаратна фільтрація шумів аналогових сигналів;

– кожні 30 секунд шлюз агрегує накопичені дані в єдиний структурований JSON-пакет телеметрії;

– пакет надсилається на Fog-сервер через мережевий інтерфейс.

Результат: fog-сервер отримує валідний пакет з температурами, завантаженнями та станом вентиляторів.

Альтернативний потік: при відсутності з'єднання з Fog-сервером пакет втрачається, шлюз продовжує зчитування та повторює спробу відправки в наступному циклі.

Прецедент П2. Зчитати та передати екологічну телеметрію

Актор: Edge-шлюз.

Передумова: Датчики вологості та пилу підключені до шлюзу.

Основний потік:

– edge-шлюз зчитує значення відносної вологості та концентрації дисперсного пилу;

– зчитуються поточні стани виконавчих пристроїв (осушувач, зволожувач);

– формується окремий JSON-пакет екологічної телеметрії;

– пакет надсилається на Fog-сервер одразу після відправки основного температурного пакету.

Результат: fog-сервер отримує актуальні дані про мікроклімат для алгоритму контролю середовища.

Прецедент П3. Виконати алгоритм адаптивного охолодження

Актор: fog-сервер.

Передумова: система знаходиться в режимі auto. Fog-сервер отримав валідний пакет телеметрії від Edge-шлюзу.

Основний потік:

– fog-сервер валідує структуру та діапазони значень вхідного пакету;

– дані записуються у базу даних часових рядів;

– для кожного GPU оновлюється історія температур (останні 3 значення) та визначається тепловий стан: нагрів, охолодження або стабільність;

– на основі поточної температури та рівня завантаженості розраховується цільовий ШІМ за кусковою кривою охолодження;

– застосовується випереджальне керування: якщо навантаження перевищує 80%, мінімальний ШІМ підвищується;

– застосовується компенсація деградації: якщо індекс корозії перевищує порогове значення, ШІМ компенсаційно збільшується;

– застосовується гістерезис: зниження ШІМ блокується протягом 60 секунд після останньої зміни; крок зниження обмежений 10% за цикл;

– сформований пакет керуючих команд зберігається в черзі очікування для Edge-шлюзу.

Результат: edge-шлюз при наступному запиті отримує оновлені ШІМ-команди та застосовує їх до вентиляторів.

Прецедент П4. Виконати алгоритм контролю мікроклімату

Актор: fog-сервер.

Передумова: fog-сервер отримав пакет екологічної телеметрії. Система в режимі auto.

Основний потік:

– fog-сервер перевіряє значення вологості та пилу відносно встановлених порогів;

– якщо вологість нижче нижнього порогу – формується команда активації зволожувача;

– якщо вологість вище верхнього порогу – формується команда активації осушувача;

– якщо вологість у нормі – обидва пристрої деактивуються;

– перевіряється інерційна затримка: активація дозволяється не раніше ніж через 10 хвилин після останньої зміни стану;

– пакет команд зберігається в черзі для Edge-шлюзу.

Результат: edge-шлюз отримує команди управління кліматичним обладнанням та комутує відповідні реле.

Прецедент П5. Сформувати температурне сповіщення

Актор: fog-сервер.

Передумова: fog-сервер обробляє вхідну телеметрію.

Основний потік:

– fog-сервер порівнює температуру кожного GPU з пороговими значеннями;

– при досягненні температурою попереджувального порогу (≥ 90°C) генерується подія рівня Warning;

– при досягненні критичного порогу (≥ 120°C) генерується подія рівня Critical;

– подія записується у базу даних та додається до списку активних сповіщень;

– при нормалізації температури GPU видаляється з переліку активних сповіщень.

Результат: оператор або інженер бачить сповіщення у журналі подій веб-системи.

Прецедент П6. Сформувати екологічне сповіщення

Актор: fog-сервер.

Передумова: fog-сервер обробляє екологічну телеметрію.

Основний потік:

– якщо концентрація пилу перевищує 50 мкг/м³ – генерується негайне сповіщення рівня Critical з повідомленням про необхідність фізичного прибирання;

- якщо вологість знаходиться поза нормальним діапазоном (40–60%) протягом більше ніж 1 годину – генерується сповіщення рівня Warning з описом ризику (статичний розряд або корозія);

- при поверненні параметрів у норму лічильник часу скидається.

Результат: екологічне сповіщення записується у базу даних та відображається у веб-системі.

Прецедент П7. Обчислити індекси деградації обладнання

Актор: fog-сервер.

Передумова: fog-сервер обробляє вхідну телеметрію разом з екологічними даними.

Основний потік:

– на основі поточних значень вологості, пилу та температури приміщення обчислюється приріст індексу корозії (CI) за рівнянням Арреніуса;

– на основі середнього RPM вентиляторів, концентрації пилу та середньої температури GPU обчислюється приріст індексу зносу вентиляторів (FWI);

– значення CI та FWI є кумулятивними і накопичуються протягом часу роботи;

– розраховані значення записуються у базу даних для візуалізації на сторінці трендів;

– при перевищенні порогових значень CI або FWI генеруються відповідні сповіщення.

Результат: інженер має доступ до динаміки зносу елементної бази та може планувати профілактичне обслуговування.

Прецедент П8. Переглянути поточний стан кластера

Актор: оператор.

Передумова: веб-система доступна; Fog-сервер функціонує.

Основний потік:

– оператор відкриває головний дашборд у веб-браузері;

– веб-система виконує циклічний запит до Fog-сервера з інтервалом 5 секунд;

– fog-сервер повертає поточні значення температур, завантаженості, ШІМ та RPM усіх GPU;

– інтерфейс відображає дані у вигляді карток з кольоровою індикацією відповідно до рівня критичності.

Результат: оператор бачить актуальний стан кластера без перезавантаження сторінки.

Прецедент П9. Переглянути параметри мікроклімату

Актор: оператор.

Передумова: веб-система доступна.

Основний потік:

– оператор переходить на сторінку мікроклімату;

– веб-система запитує у Fog-сервера поточні значення вологості, пилу та стан кліматичного обладнання;

– fog-сервер повертає останні відомі значення та статус виконавчих пристроїв;

– інтерфейс відображає параметри мікроклімату та активні екологічні сповіщення.

Результат: оператор контролює параметри середовища та стан осушувача/зволожувача.

Прецедент П10. Переглянути журнал подій

Актор: оператор, інженер.

Передумова: веб-система доступна.

Основний потік:

– користувач переходить на сторінку журналу подій;

– веб-система запитує перелік активних та архівних сповіщень у Fog-сервера;

– інтерфейс відображає хронологічний список подій з рівнем критичності, часовою міткою та описом.

Результат: користувач має повну інформацію про всі зафіксовані аномалії.

Прецедент П11. Аналізувати архівні дані телеметрії

Актор: інженер.

Передумова: у базі даних наявні архівні метрики.

Основний потік:

- інженер переходить на сторінку аналізу та обирає часовий діапазон;

- веб-система запитує у Fog-сервера архівні дані температур GPU та параметрів вентиляторів за обраний період;

- fog-сервер виконує агрегаційний запит до TSDB та повертає набір точок;

- інтерфейс будує інтерактивні графіки часових рядів з можливістю накладання параметрів (наприклад, температура GPU та ШІМ на одному полотні).

Результат: інженер візуально перевіряє ефективність роботи алгоритму гістерезису та випереджального керування.

Прецедент П12. Аналізувати тренди деградації обладнання

Актор: інженер.

Передумова: у базі даних наявні накопичені значення CI та FWI.

Основний потік:

– інженер переходить на сторінку трендів;

– веб-система запитує поточні та історичні значення індексів CI та FWI;

– fog-сервер повертає кумулятивні значення, рівні ризику та модифікатори ефективності;

– інтерфейс відображає графіки динаміки зносу та прогнозні лінії.

Результат: інженер оцінює залишковий ресурс обладнання та планує профілактику.

Прецедент П13. Перемкнути режим керування системою

Актор: інженер.

Передумова: веб-система доступна; Fog-сервер функціонує.

Основний потік:

– інженер на панелі адміністрування обирає режим auto або manual;

– веб-система надсилає команду зміни режиму на Fog-сервер;

– fog-сервер оновлює внутрішній стан режиму та фіксує час зміни і ініціатора (користувач);

– при переході з manual на auto всі раніше встановлені ручні команди скидаються;

– fog-сервер підтверджує зміну режиму та повертає новий стан.

Результат: система працює у обраному режимі. В режимі auto алгоритм керує вентиляторами. В режимі manual очікуються команди від інженера.

Прецедент П14. Встановити параметри ШІМ вручну

Актор: інженер.

Передумова: система знаходиться в режимі manual.

Основний потік:

– інженер на панелі адміністрування задає бажані значення ШІМ для одного або кількох вентиляторів;

– веб-система формує пакет ручних команд та надсилає його на Fog-сервер;

– fog-сервер перевіряє, що система дійсно знаходиться в ручному режимі. Якщо ні – запит відхиляється;

– команди зберігаються в черзі для Edge-шлюзу та застосовуються при наступному циклі опитування;

– дія інженера фіксується в журналі дій користувача.

Результат: вентилятори працюють на заданих інженером оборотах до зміни команди або повернення в автоматичний режим.

Альтернативний потік: якщо система в режимі auto, Fog-сервер повертає помилку з поясненням.

Прецедент П15. Задати профіль кліматичного середовища

Актор: інженер.

Передумова: система працює з програмним емулятором апаратного рівня.

Основний потік:

– інженер на панелі адміністрування обирає один з 9 попередньо визначених кліматичних профілів (комбінації рівня пилу та вологості);

– веб-система надсилає ідентифікатор профілю на Fog-сервер;

– fog-сервер зберігає цільовий ідентифікатор профілю та скидає кумулятивні індекси CI та FWI;

– емулятор при черговому циклі опитування виявляє зміну профілю, переініціалізує сенсори вологості та пилу на нові початкові та рівноважні значення.

Результат: емулятор починає симулювати обрані кліматичні умови. Це дозволяє інженеру тестувати поведінку алгоритмів за різних сценаріїв.

Прецедент П16. Отримати та виконати керуючі команди

Актор: edge-шлюз.

Передумова: edge-шлюз щойно успішно відправив пакет телеметрії.

Основний потік:

– одразу після відправки телеметрії Edge-шлюз виконує запит до Fog-сервера на отримання пакету команд для вентиляторів;

– fog-сервер повертає пакет із ШІМ-значеннями для кожного вентилятора або порожню відповідь, якщо команд немає;

– edge-шлюз інтерпретує отримані команди та генерує відповідні ШІМ-сигнали;

– після цього Edge-шлюз виконує аналогічний запит на отримання екологічних команд (стан осушувача, зволожувача);

– при наявності екологічних команд шлюз комутує відповідні релейні виходи.

Результат: виконавчі механізми працюють відповідно до рішень Fog-сервера.

Альтернативний потік: при відсутності з'єднання шлюз утримує останні отримані команди.

На основі визначених прецедентів побудовано діаграму варіантів використання та занесено до рисунку 2.2. Між прецедентами позначені відношення «include» (обов'язкове включення підпрецеденту) та «extend» (умовне розширення при виконанні спеціальної умови).

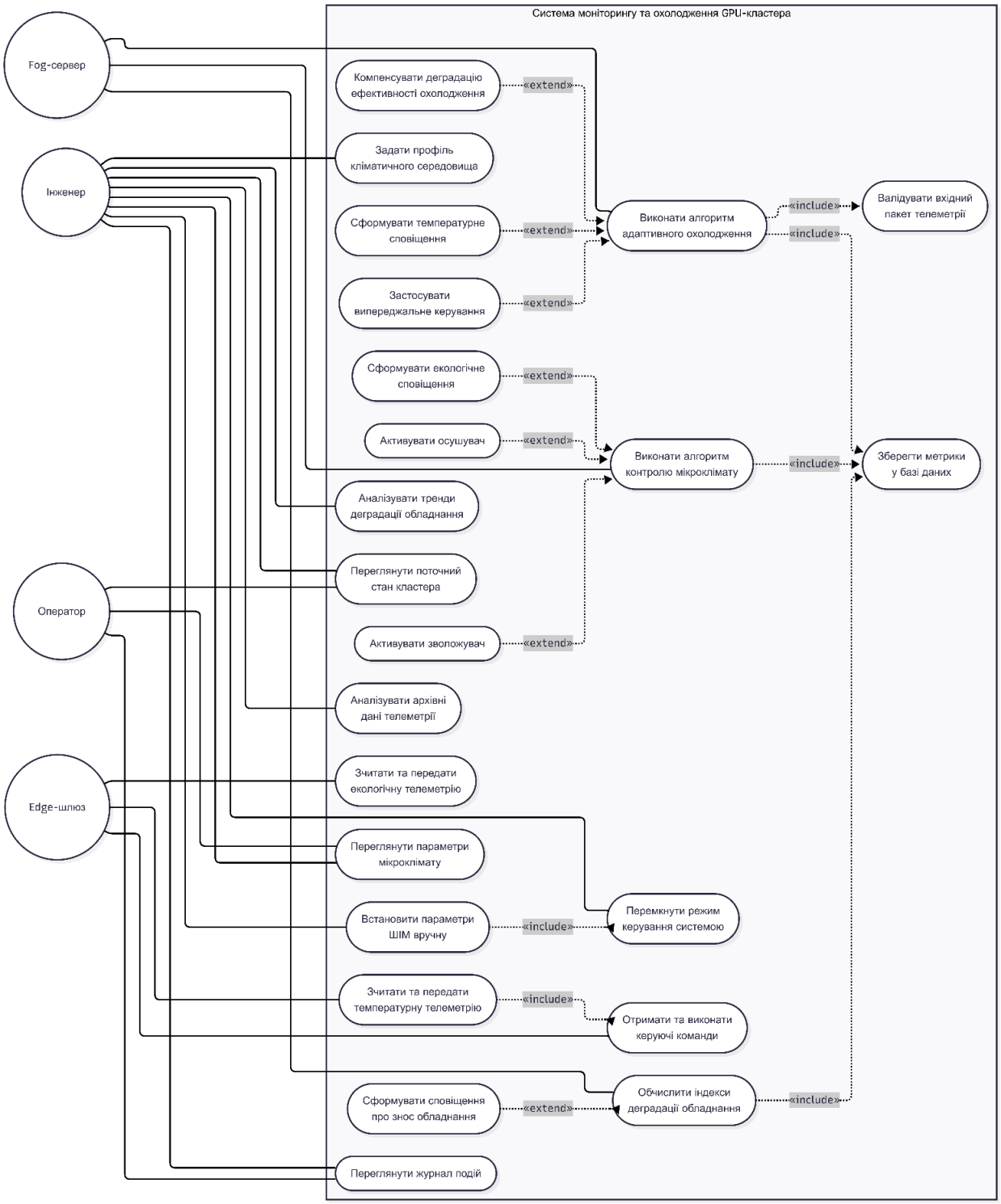


Рисунок 2.2 – Діаграма варіантів використання

Прецеденти системи розподіляються за трьома функціональними групами, діаграми послідовності яких ми відобразили на рисунках 2.3-2.5.

Збір даних (П1, П2, П16) – ініціюються Edge-шлюзом циклічно без участі користувача. П1 включає («include») П16, оскільки запит керуючих команд виконується автоматично після кожної успішної відправки температурної телеметрії.

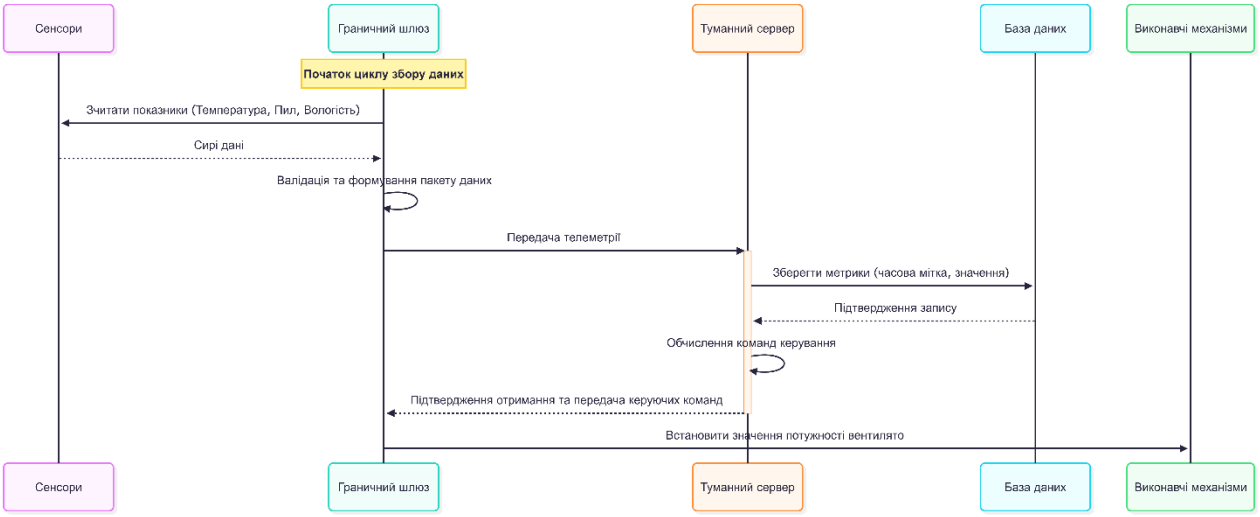


Рисунок 2.3 – Діаграма послідовності збору даних

Автоматична обробка та керування (П1–П7 та підпрецеденти) – виконуються туманним сервером як реакція на вхідну телеметрію. П3 («include») завжди виконує валідацію пакету та запис до TSDB; умовно («extend») розширюється випереджальним керуванням при навантаженні > 80%, компенсацією деградації при CI > 1,5 та формуванням температурних сповіщень при перевищенні порогів. П4 умовно активує осушувач (RH > 60%) або зволожувач (RH < 40%) і формує екологічні сповіщення. П7 умовно розширюється сповіщенням про знос при CI або FWI вище критичного порогу.

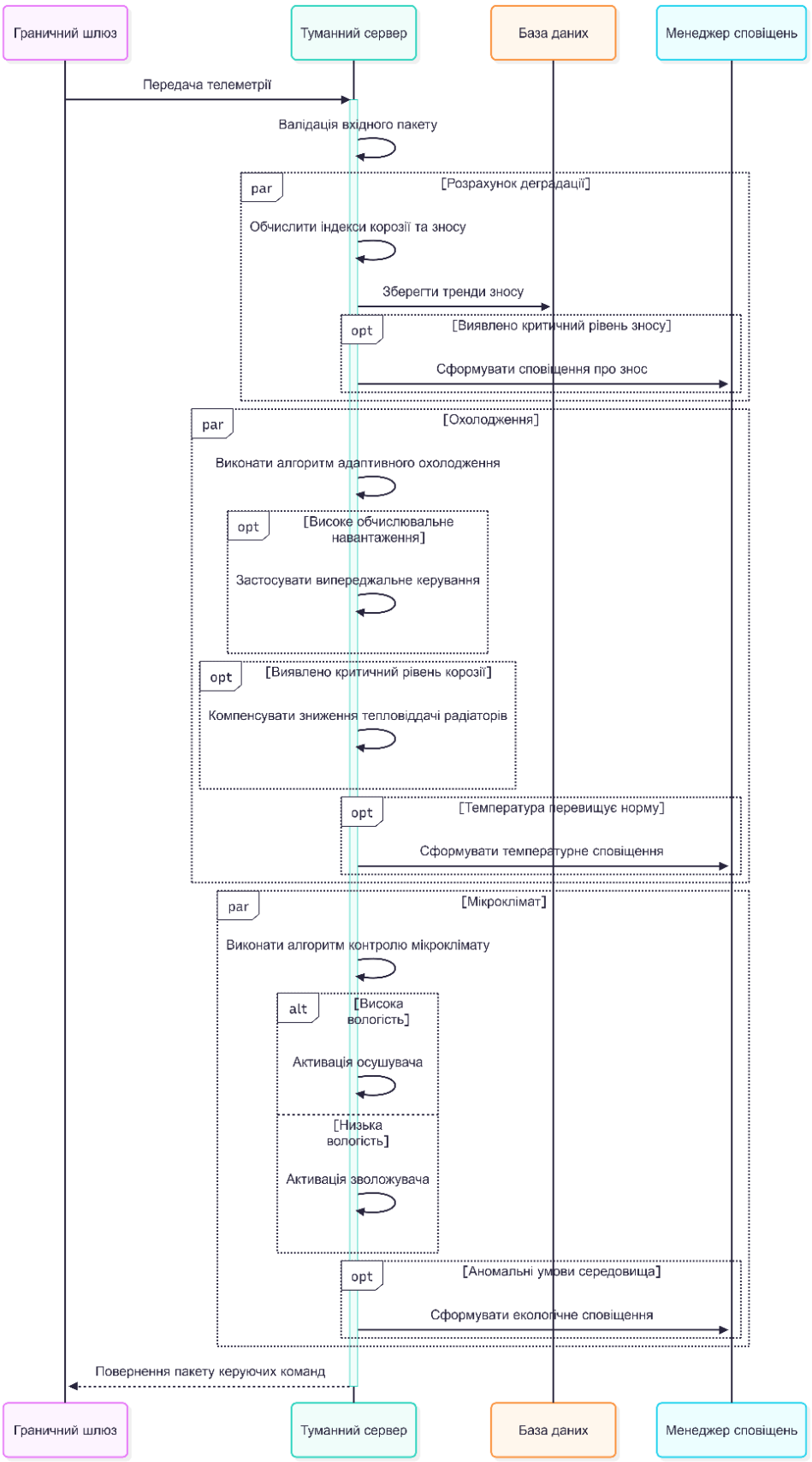


Рисунок 2.4 – Діаграма послідовності автоматичної обробки та керування

Взаємодія користувача (П8–П15) – ініціюються через веб-систему. Усі команди маршрутизуються виключно через Fog-сервер. П14 включає («include») П13, оскільки ручне встановлення ШІМ вимагає попереднього переходу в ручний режим. Інженер має доступ до повного переліку прецедентів взаємодії користувача. Оператор обмежений функціями пасивного моніторингу (П8, П9, П10).

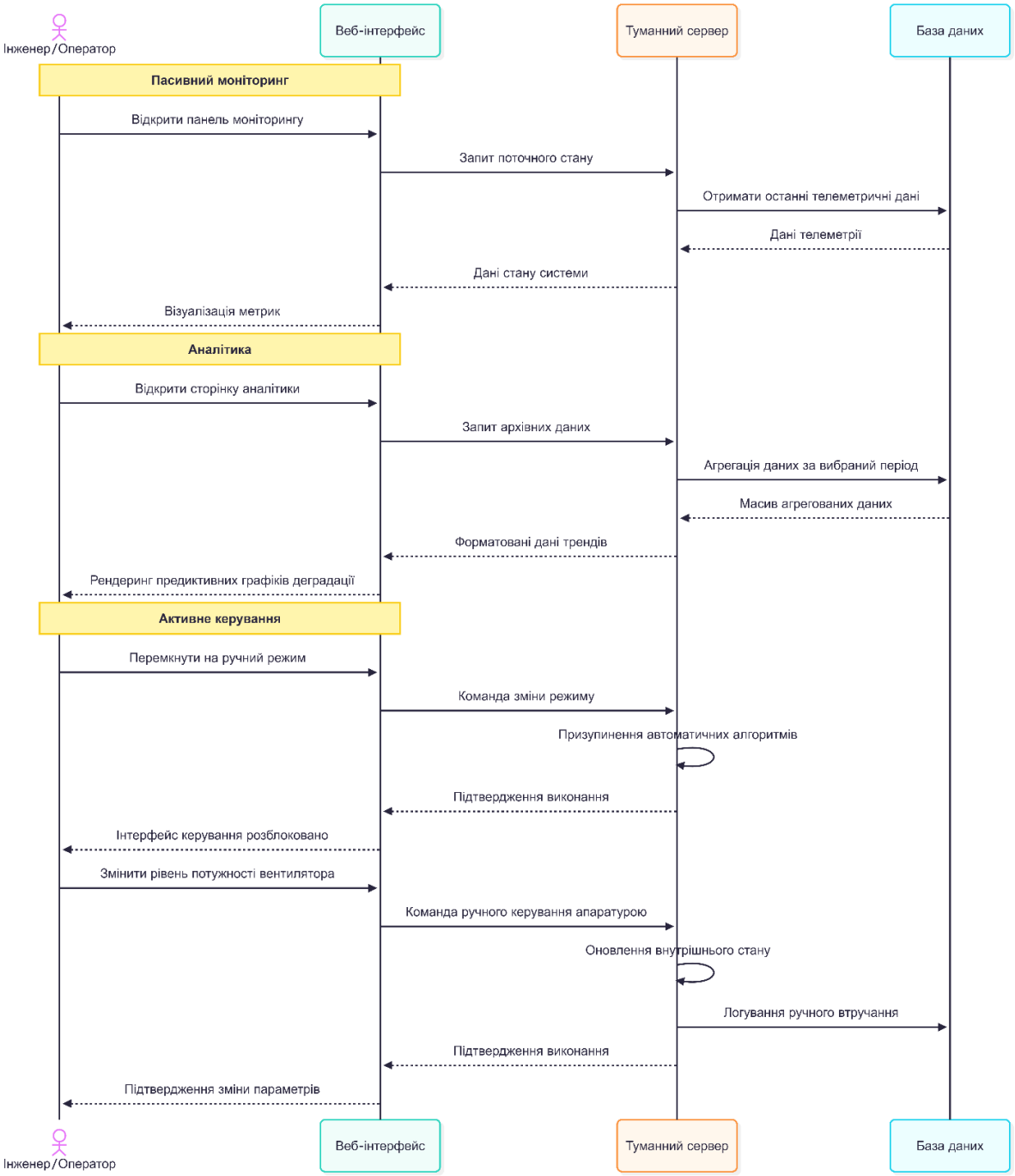


Рисунок 2.5 – Діаграма послідовності взаємодії користувача

2.5 Математичне моделювання об'єкта керування

Проектування архітектури адаптивного керування для обчислювальних кластерів вимагає математичного опису фізичних процесів, що протікають у серверному середовищі. Розроблена математична модель закладена в основу аналітичного ядра туманного сервера для реалізації евристичного машинного навчання та розрахунку керуючих команд [20]. Також ці ж самі рівняння формують логіку програмно-апаратного стенда (Hardware-in-the-Loop), який виступає в ролі цифрового двійника реального кластера, що необхідно для тестуваннння. Це дозволяє здійснювати масштабування всієї системи та тестувати алгоритми предиктивного обслуговування без зміни кон

фігураційних файлів, заощаджуючи сотні годин відлагодження. Математично точна симуляція гарантує збереження дорогоцінних обчислювальних компонентів (графічних процесорів), усуваючи ризики теплового пробою під час тестування експериментальних алгоритмів випереджального керування.

Процес нагрівання та охолодження графічного процесора описується з урахуванням закону охолодження Ньютона та теплової інерції. Швидкість зміни температури залежить від поточної обчислювальної нагрузки, температури навколишнього середовища (приміщення) та інтенсивності обдуву вентиляторами. Ефективне охолодження за дискретний проміжок часу Δt описується рівнянням:

$$Q_{\text{охолодження}} = k_{\text{охолодження}} * PWM * \Delta t * \frac{T_{gpu} - T_{\text{кімнати}}}{50}, \qquad (2.1)$$

де $Q_{\text{охолодження}}$ – ефективна складова зниження температури GPU за такт;

$k_{\text{охолодження}}$ – коефіцієнт швидкості тепловіддачі конкретного радіатора;

PWM – поточне значення керуючого сигналу широтно-імпульсної модуляції (від 0.0 до 1.0);

Δt – дискретний крок часу симуляції;

T_{gpu}– поточна температура графічного процесора (°C);

$T_{кімнати}$– температура навколишнього середовища приміщення (°C).

Нормуючий дільник 50 використовується для масштабування різниці температур до безрозмірного коефіцієнта. Згідно з моделлю, ефективність вентиляторів суттєво знижується при зменшенні градієнта температур $T_{gpu} - T_{кімнати}$, що вимагає підвищення PWM для береження сталої швидкості відводу тепла в умовах підвищеної кімнатної температури.

Пил виступає у ролі теплоізолятора, блокуючи конвективний теплообмін на пластинах радіаторів. Накопичення дисперсних частинок моделюється як асимптотичний процес, що прямує до точки рівноваги. Швидкість осідання пилу є функцією не лише від базової концентрації в приміщенні, але й від інтенсивності повітряного потоку, що генерується вентиляторами охолодження.

Зміна концентрації пилу за такт часу визначається рівнянням:

$$\Delta Пилу = v_{базове} * M_{кулера} * \frac{Пил_{eq} - Пил_{поточне}}{50}, \quad (2.2)$$

де $\Delta Пилу$ – приріст концентрації пилу (мкг/м³);

$v_{базове}$ – базова швидкість осідання пилу у стані спокою;

$M_{кулера}$ – мультиплікатор впливу повітряного потоку (при PWM > 0.8 повітряні потоки піднімають осілий пил, і Mfan зростає до 1.2, прискорюючи забруднення);

$Пил_{макс}$ – рівноважна (максимальна) концентрація пилу для поточного екологічного профілю;

$Пил_{поточне}$ – поточна концентрація пилу в корпусі.

Відносна вологість (ΔRH) середовища змінюється під впливом природних факторів та активної роботи кліматичного обладнання (осушувачів та

зволожувачів). Динаміка зміни вологості також моделюється через наближення до рівноважного стану середовища.

$$\Delta RH = RH_{eq} - RH_{\text{поточне}} * k_{th} * F_{\text{вентиляції}} + E_{hvac} \ , \quad (2.3)$$

де ΔRH – зміна відносної вологості (%);

RH_{eq} – рівноважний рівень вологості приміщення;

$RH_{\text{поточне}}$ – поточний рівень вологості;

k_{th} – базова швидкість зміни вологості середовища;

$F_{\text{вентиляції}}$ – фактор вентиляції приміщення;

E_{hvac} – лінійна компонента зміни від роботи активного кліматичного обладнання (наприклад, +5.0%/год для зволожувача, -5.0%/год для осушувача).

Для реалізації предиктивного обслуговування туманний сервер розраховує два ключових кумулятивних індекси зносу: Індекс ризику корозії (CI) та Індекс зносу вентиляторів (FWI).

Модель CI спирається на рівняння Арреніуса, яке описує залежність швидкості хімічних реакцій (зокрема оксидації) від температури. Модель враховує синергетичний ефект вологості та накопиченого пилу, який утримує вологу на поверхні плат.

Приріст індексу корозії за проміжок часу Δt (в годинах) становить:

$$\Delta CI = \left(\frac{RH}{50}\right) * (1 + 0.1 * \text{Пил}) * e^{\frac{T_{\text{кімнати}}-25}{10}} * \Delta t \ , \quad (2.4)$$

де $e^{\frac{T_{\text{кімнати}}-25}{10}}$ – температурний фактор Арреніуса, що постулює експоненційне прискорення реакції при підвищенні температури вище базових 25°C. При перевищенні значення CI > 1.5 система фіксує деградацію тепловіддачі радіаторів

(через наліт), що апаратно знижує його теплопровідність до 4%, що вимагає компенсаційного збільшення обертів вентиляторів.

Підшипники вентиляторів деградують під впливом трьох факторів: кількості обертів (механічне тертя), потрапляння пилу (абразивний ефект) та високих температур (деградація мастила).

Приріст індексу зносу виражається в еквівалентних годинах роботи:

$$\Delta FWI = \left(\frac{RPM_{середнє}}{RPM_{макс}}\right) * \left(1 + 0.2 * \frac{Пил}{50}\right) * K_{темп} * \Delta t \ , \quad (2.5)$$

де $RPM_{середнє}/RPM_{макс}$ – співвідношення поточних обертів до максимально можливих (навантаження на ротор);

Пил/50 – фактор абразивного впливу пилу;

$K_{темп}$ – температурний коефіцієнт (приймає значення 1.5 при температурі GPU > 70°C, інакше 1.0).

Накопичення значення FWI дозволяє системі завчасно формувати оповіщення про необхідність заміни кулера до моменту його фізичної зупинки.

3 ВИБІР КОМПОНЕНТІВ ТА ТЕХНОЛОГІЧНОГО СТЕКУ СИСТЕМИ

3.1 Компоненти апаратного рівня

3.1.1 Датчик температури графічного процесора

Апаратний рівень системи складається з чотирьох груп компонентів: датчика температури для кожного графічного процесора, комбінованого датчика температури та вологості навколишнього середовища, датчика концентрації дисперсних частинок, а також мікроконтролера, що виконує функцію граничного

шлюзу. Відбір здійснювався за методологією порівняльного аналізу: для кожної позиції розглядалося кілька альтернатив, після чого на підставі ключових технічних критеріїв обиралося рішення з найкращою відповідністю функціональним та нефункціональним вимогам системи.

Ключова вимога до датчика температури GPU – можливість безконтактного або контактного кріплення безпосередньо до корпусу або радіатора відеокарти. Датчик повинен витримувати робочі температури до 100-125°C, які відповідають критичним режимам роботи GPU. Оскільки до одного граничного шлюзу підключається кілька GPU, обов'язковою є підтримка протоколу з адресацією пристроїв на одній лінії зв'язку.

DS18B20 (Maxim Integrated/Dallas Semiconductor) – цифровий датчик температури з інтерфейсом 1-Wire. Відрізняється конструктивом металевого зонду (TO-92 або герметична гільза з нержавіючої сталі), що дозволяє механічно фіксувати його на поверхні радіатора. Підтримує адресацію до 127 пристроїв на одній двопровідній лінії. Діапазон вимірювання: −55°C … +125°C, роздільна здатність: 9–12 біт (налаштовується програмно), похибка: ±0,5°C у діапазоні 0–70°C.

LM35 (Texas Instruments) – аналоговий датчик температури з лінійною характеристикою виходу (10 мВ/°C). Діапазон: −55°C … +150°C. Не підтримує мультидроп-топологію – кожен екземпляр вимагає окремого аналогового входу мікроконтролера, що унеможливлює масштабування при великій кількості GPU. Потребує АЦП з достатньою роздільною здатністю.

NTC-термістор (наприклад, серія 10k NTC MF52) – пасивний резистивний елемент. Характеристика нелінійна, потребує апаратної або програмної лінеаризації. Відсутня самоідентифікація, висока чутливість до паразитної ємності лінії. При масштабуванні значно ускладнює схемотехніку.

До таблиці 3.1 занесемо порівняння характеристик описаних датчиків температури.

Таблиця 3.1 – Порівняння датчиків температури

Характеристика	DS18B20	LM35	NTC-термістор
Діапазон температур	−55 … +125°C	−55 … +150°C	−40 … +125°C (тип.)
Тип виходу	Цифровий (1-Wire)	Аналоговий	Аналоговий
Підтримка мультидроп	Так (до 127 пристроїв)	Ні	Ні
Похибка вимірювання	±0,5°C (0–70°C)	±0,5°C (типова)	±1–3°C (після лінеаризації)
Потреба в АЦП	Ні	Так	Так
Складність монтажу	Низька	Середня	Висока
Захисний корпус (гільза)	Доступний	Недоступний	Доступний

На підставі нефункціональних вимог розділу 2.2, зокрема вимог до масштабованості апаратного рівня та мінімізації кількості провідників між сенсорами і шлюзом, обрано DS18B20. Протокол 1-Wire дозволяє підключити датчики від усіх GPU до єдиної двопровідної лінії без мультиплексорів, що критично при масштабуванні кластера. Конструктив металевої гільзи забезпечує надійний тепловий контакт із радіатором GPU та механічну стійкість. Цифровий вихід виключає вплив перешкод довгої лінії зв'язку, властивий аналоговим рішенням. Схему підключення датчику занесемо до рисунку 3.1.

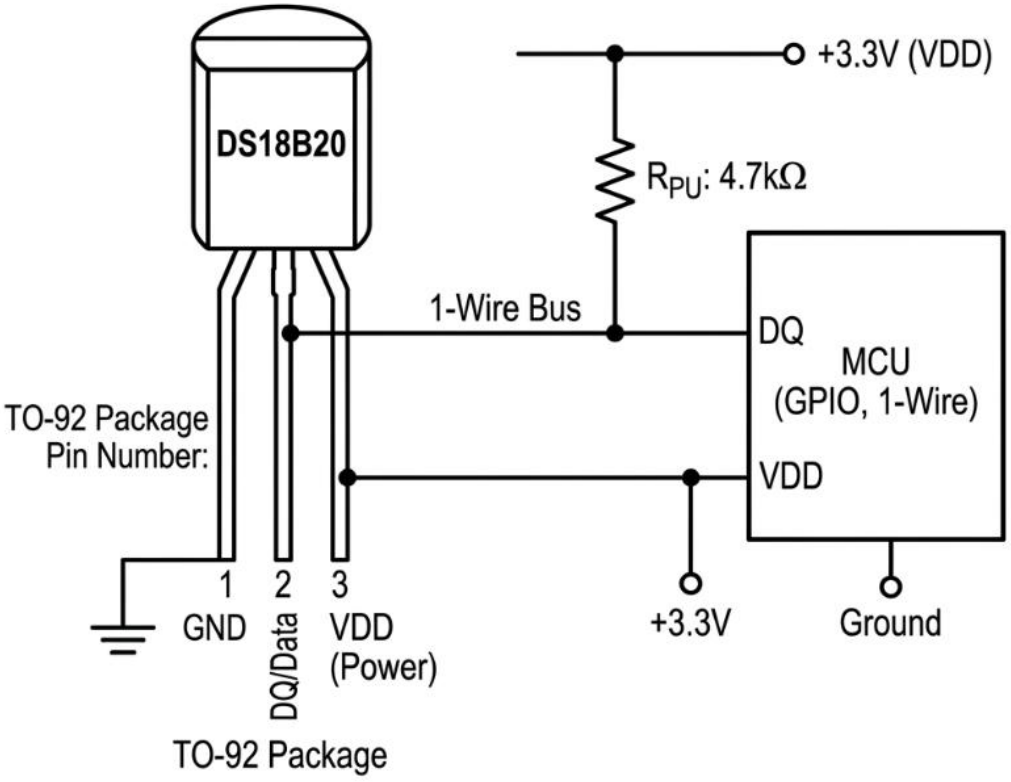


Рисунок 3.1 – Схема підключення датчику температури

3.1.2 Датчик температури та вологості навколишнього середовища

Датчик розміщується в серверному приміщенні та повинен одночасно вимірювати відносну вологість і температуру повітря. Допустимий діапазон вологості для серверного середовища: 40-60% (відповідно до вимог розділу 2.1). Датчик повинен утримувати похибку вологості в межах ±5% RH, щоб система своєчасно виявляла відхилення та активувала виконавчі механізми.

DHT22 (AM2302) – цифровий датчик з однодротовим інтерфейсом (proprietary single-wire). Діапазон вологості: 0–100% RH, похибка: ±2–5% RH. Діапазон температур: −40°C … +80°C, похибка: ±0,5°C. Частота опитування: не частіше 0,5 Гц. Живлення: 3,3–5,5 В.

DHT11 – бюджетний варіант у тому ж конструктиві. Діапазон вологості: 20–80% RH, похибка: ±5% RH. Діапазон температур: 0–50°C, похибка: ±2°C. Частота опитування: не частіше 1 Гц. Значно поступається DHT22 за точністю та діапазоном.

SHT31 (Sensirion) – датчик промислового класу з інтерфейсом I²C. Похибка вологості: ±2% RH, температури: ±0,3°C. Підтримує тактову частоту до 1 МГц. Вища вартість, але забезпечує суттєво вищу точність. Має вбудований нагрівач для видалення конденсату.

BME280 (Bosch) – комбінований датчик температури, вологості та атмосферного тиску з інтерфейсами I²C/SPI. Похибка вологості: ±3% RH, температури: ±1°C. Частота опитування до 157,5 Гц у нормальному режимі. Вбудований фільтр Калмана.

До таблиці 3.2 занесемо порівняння характеристик описаних датчиків температури та вологості.

Таблиця 3.2 – Порівняння датчиків температури та вологості

Характеристика	DHT22	DHT11	SHT31	BME280
Діапазон вологості	0–100% RH	20–80% RH	0–100% RH	0–100% RH
Похибка вологості	±2–5% RH	±5% RH	±2% RH	±3% RH
Діапазон температур	−40 … +80°C	0 … +50°C	−40 … +125°C	−40 … +85°C
Інтерфейс	Single-wire	Single-wire	I²C	I²C / SPI
Частота опитування	0,5 Гц	1 Гц	до 1 Гц	до 157,5 Гц

Відповідно до вимог розділу 2.2 (контроль вологості з порогами 40% та 60% RH), точність похибки ±5% RH є граничною допустимою. Датчик DHT22 забезпечує необхідний діапазон (0-100% RH) і відповідає допустимій похибці при нижчій вартості порівняно з SHT31. Частота опитування 0,5 Гц (один відлік кожні 2 секунди) є достатньою, оскільки динаміка зміни вологості у серверному приміщенні значно повільніша. DHT11 відхилено через обмежений діапазон вологості та температур (нижня межа 0°C виключає експлуатацію у неопалюваних приміщеннях). SHT31 та BME280 перевищують вимоги за точністю, що не

виправдовує їхню вищу вартість у даному застосуванні. Схему датчику температури та вологості занесемо до рисунку 3.2.

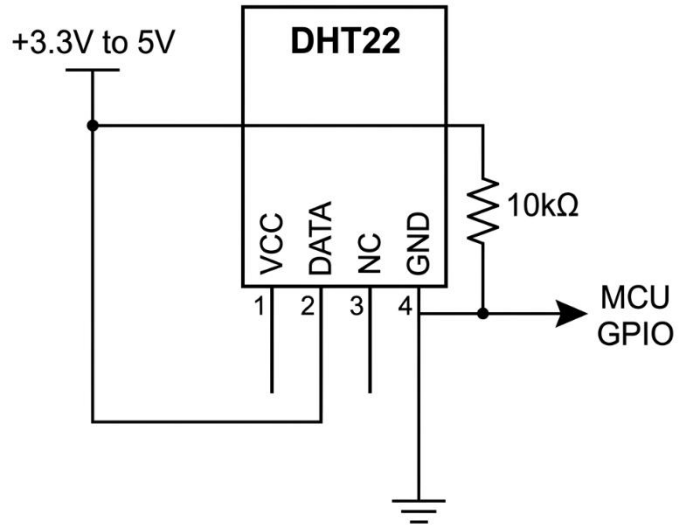


Рисунок 3.2 – Схема датчику вологості на температури

3.1.3 Датчик концентрації дисперсних частинок

Датчик повинен вимірювати концентрацію зважених часток у повітрі (PM – Particulate Matter). Поріг тривоги системи встановлено на рівні 50 мкг/м³, відповідно датчик має забезпечувати вимірювання в діапазоні принаймні 0–200 мкг/м³ з достатньою роздільною здатністю для детектування наближення до критичного порогу.

GP2Y1010AU0F (Sharp) – оптичний датчик на принципі розсіювання інфрачервоного світла. Аналоговий вихід (0–3,5 В), пропорційний концентрації частинок. Діапазон: 0–500 мкг/м³. Не потребує руху повітря через вентилятор – використовує природну конвекцію. Компактний, низьке енергоспоживання. Потребує АЦП на стороні мікроконтролера.

SDS011 (Nova Fitness) – лазерний датчик частинок PM2.5 та PM10. Цифровий інтерфейс UART (9600 бод). Діапазон: 0–999,9 мкг/м³, похибка: ±15%. Вбудований мікровентилятор для активної прокачки повітря. Висока точність, але більший конструктив та рівень шуму.

PMS5003 (Plantower) – цифровий датчик з лазером та активною вентиляцією. Виходи: UART або I²C. Вимірює PM1.0, PM2.5, PM10. Час відгуку до 1 секунди. Висока точність, але значне енергоспоживання та розміри.

До таблиці 3.3 занесемо порівняння характеристик описаних датчиків концентрації дисперсних частинок

Таблиця 3.3 – Порівняння датчиків концентрації дисперсних частинок

Характеристика	GP2Y1010AU0F	SDS011	PMS5003
Принцип вимірювання	ІЧ-розсіювання	Лазер	Лазер
Тип виходу	Аналоговий	UART (цифровий)	UART / I²C
Діапазон вимірювання	0–500 мкг/м³	0–999,9 мкг/м³	0–1000 мкг/м³
Активна вентиляція	Ні	Так	Так
Живлення	5 В	5 В	5 В
Розміри	Компактні	Середні	Великі
Вартість (відносна)	Низька	Середня	Середня

Для даної системи обрано GP2Y1010AU0F. Ключова перевага – відсутність вбудованого вентилятора: у серверному середовищі постійний рух повітря від охолоджувальних систем GPU забезпечує природну прокачку повітря через датчик, роблячи примусову вентиляцію надлишковою. Це усуває необхідність примусової прокачки повітря та дозволяє знизити енергоспоживання підсистеми моніторингу.

Відсутність рухомих механічних компонентів також мінімізує ризик механічного зносу, деградації підшипників і накопичення пилу на лопатях вентилятора, що позитивно впливає на довговічність пристрою при безперервній експлуатації. Відсутність рухомих механічних частин підвищує надійність.

Схему підключення датчика концентрації дисперсних частинок відобразимо до рисунку 3.3.

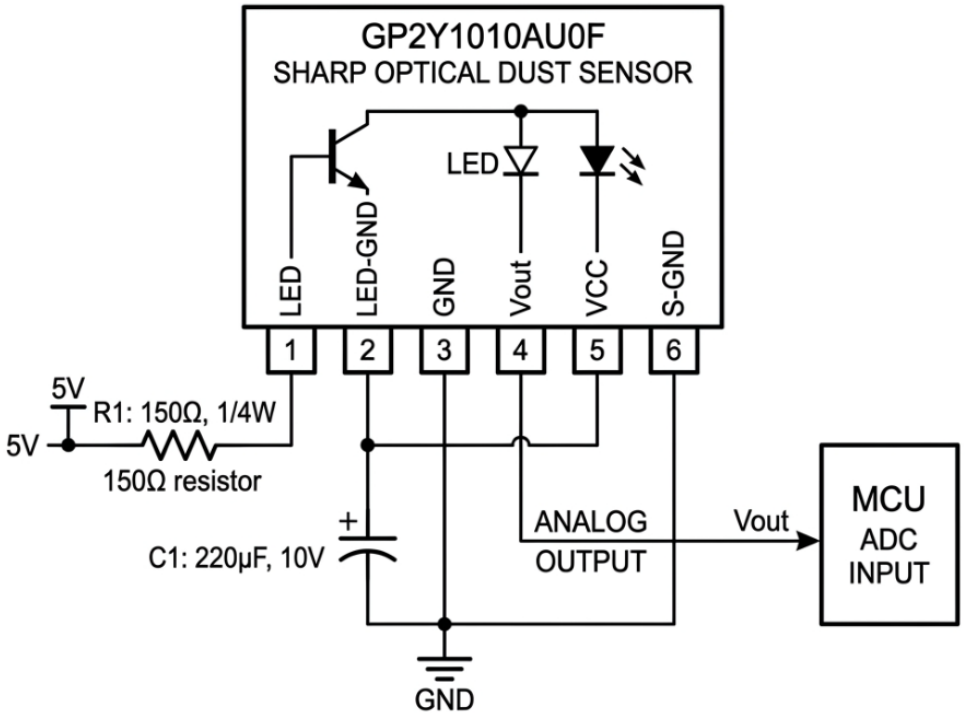


Рисунок 3.3 – Схема підключень датчика концентрації дисперсних частинок

Однак вибір GP2Y1010AU0F зумовлює необхідність обрання зовнішнього аналогово-цифрового перетворювача (АЦП). Причин дві: по-перше, максимальна напруга аналогового виходу датчика становить 3,5 В, що перевищує допустимий вхідний рівень АЦП мікроконтролера ESP32 (3,3 В); по-друге, вбудований АЦП ESP32 має задокументований недолік – значне зниження точності та зростання шуму під час активної роботи радіомодуля Wi-Fi через електромагнітні наводки на аналоговий опорний вузол. Оскільки граничний шлюз у даній системі постійно підтримує Wi-Fi-з'єднання з туманним сервером, використання внутрішнього АЦП ESP32 для вимірювань запиленості є неприйнятним. З огляду на це, необхідне підключення зовнішнього АЦП до ESP32 по інтерфейсу I²C, що описано у підрозділі 3.1.4.

3.1.4 Зовнішній аналогово-цифровий перетворювач

Оскільки вбудований АЦП мікроконтролера ESP32 не може бути використаний для вимірювання аналогового сигналу датчика GP2Y1010AU0F, до схеми граничного шлюзу включено зовнішній АЦП з підключенням по шині I²C.

ADS1115 (Texas Instruments) – 16-бітний АЦП з інтерфейсом I²C. Чотири диференційних або вісім одиночних аналогових входів (у режимі одиночного вимірювання). Програмований підсилювач (PGA): коефіцієнти від ×2/3 до ×16, вхідний діапазон від ±0,256 В до ±6,144 В. Швидкість вибірки: 8–860 вибірок/с. Живлення: 2,0–5,5 В. Підтримує до 4 пристроїв на одній I²C-шині (адресація через пін ADDR).

MCP3204 (Microchip) – 12-бітний АЦП з інтерфейсом SPI. 4 канали в режимі одиночного вимірювання або 2 диференційних. Швидкість вибірки до 100 квибірок/с при 5 В. Нижча роздільна здатність (12 біт проти 16), потребує ліній SPI замість I²C.

PCF8591 (NXP) – 8-бітний АЦП + ЦАП з інтерфейсом I²C. 4 аналогових входи. Роздільна здатність 8 біт є недостатньою для точного вимірювання аналогового виходу GP2Y1010AU0F в діапазоні 0–3,5 В.

До таблиці 3.3 занесемо порівняння характеристик описаних аналогово цифрових конверторів.

Таблиця 3.3 – Порівняння аналогово-цифрових конверторів

Характеристика	ADS1115	MCP3204	PCF8591
Роздільна здатність	16 біт	12 біт	8 біт
Інтерфейс	I²C	SPI	I²C
Кількість каналів	4 (одиночних)	4 (одиночних)	4
Вхідний діапазон	±0,256–±6,144 В (PGA)	0–VDD	0–VDD

Продовження таблиці 3.3

Характеристика	ADS1115	MCP3204	PCF8591
Швидкість вибірки	до 860 вибірок/с	до 100 квибірок/с	до 11,9 квибірок/с
Сумісність з 3,3 В	Так	Так	Так

Обрано ADS1115. Програмований підсилювач дозволяє налаштувати вхідний діапазон на ±4,096 В, що повністю охоплює вихід GP2Y1010AU0F (до 3,5 В) без ризику пошкодження і без додаткового дільника напруги. 16-бітна роздільна здатність забезпечує квант вимірювання ~0,125 мВ при PGA ×1, що значно перевищує мінімально необхідну точність для детектування порогу 50 мкг/м³. Підключення по вже задіяній шині I²C (спільно з SSD1306, описаним у 3.1.6) не вимагає додаткових ліній GPIO (Рис. 3.5).

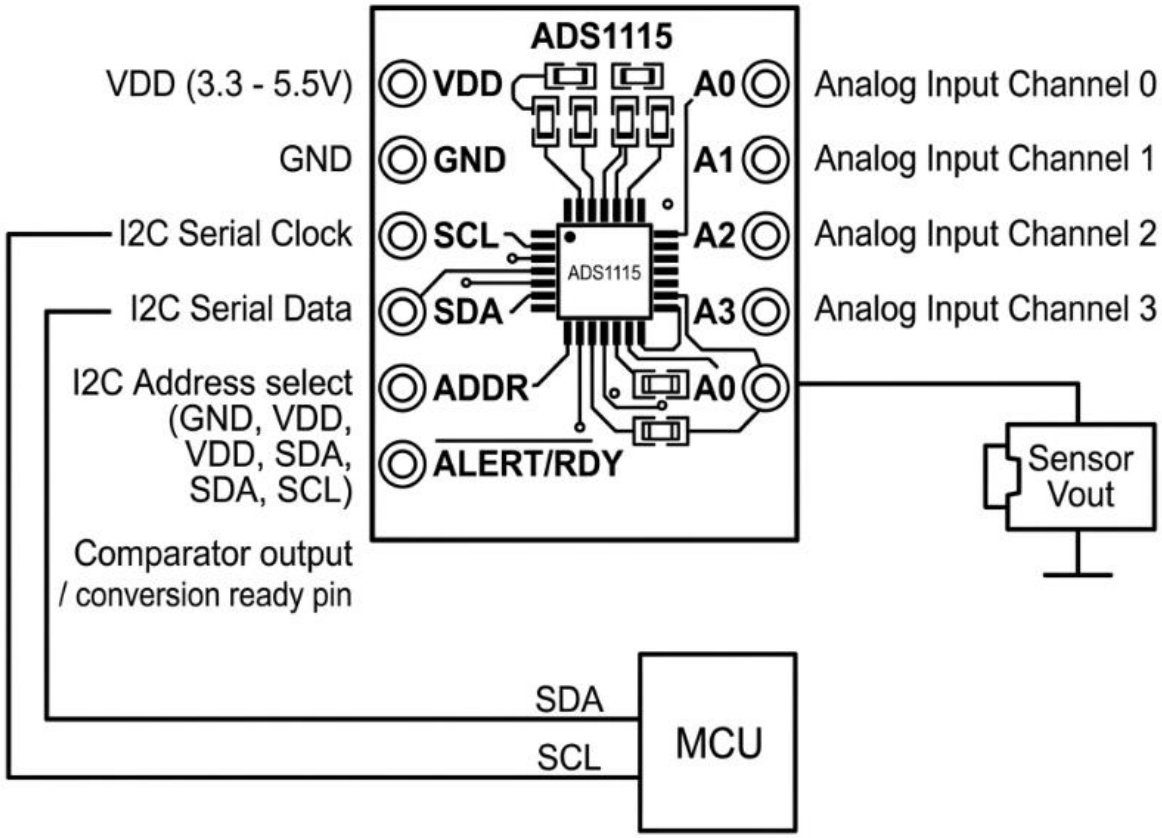


Рисунок 3.5 – Схема розташування виводів аналого цифрового конвертера

3.1.5 Мікроконтролер (граничний шлюз)

Мікроконтролер виконує функцію граничного шлюзу: він циклічно опитує всі датчики, виконує первинну обробку (шумоподавлення), формує пакет телеметрії та передає його на туманний сервер по мережі. Обов'язкові вимоги: наявність вбудованого мережевого інтерфейсу (Wi-Fi), підтримка протоколів 1-Wire (для DS18B20), UART або Single-Wire (для DHT22) та I²C (для ADS1115 і дисплея). Обчислювальна потужність має забезпечувати паралельне виконання задач опитування датчиків і підтримку мережевого стеку. Архітектура мікроконтролера повинна забезпечувати одночасне виконання задач опитування сенсорів, обробки переривань, формування пакетів даних та підтримки TCP/IP-стеку без критичних затримок або втрати стабільності системи.

Фундаментальна відмінність між кандидатами полягає в архітектурному підході до реалізації функціональності. ESP32 є системою на кристалі (System-on-Chip, SoC – всі необхідні периферійні блоки (процесор, пам'ять, радіомодуль, АЦП, крипторушій) інтегровано в одному напівпровідниковому кристалі. Мікроконтролери на базі ATmega328P (Arduino Uno, Arduino Nano) є автономними мікроконтролерами (Standalone MCU) – кристал містить лише CPU, пам'ять та базову периферію; усі додаткові функції (Wi-Fi, Ethernet) реалізуються виключно через зовнішні компоненти, що підключаються окремо.

Ця різниця безпосередньо впливає на складність схемотехніки, надійність та масштабованість системи. У випадку розширення кількості сенсорів або додавання нових функцій ESP32 дозволяє виконати модернізацію переважно програмними засобами, тоді як для ATmega328P часто виникає необхідність у фізичному підключенні додаткових модулів, мультиплексорів або співпроцесорів. Це ускладнює подальший розвиток системи та підвищує ймовірність виникнення апаратних конфліктів.

Приклад різниці структури архітектур на кристалі та автономних мікроконтролерів відобразимо на рисунку 3.6.

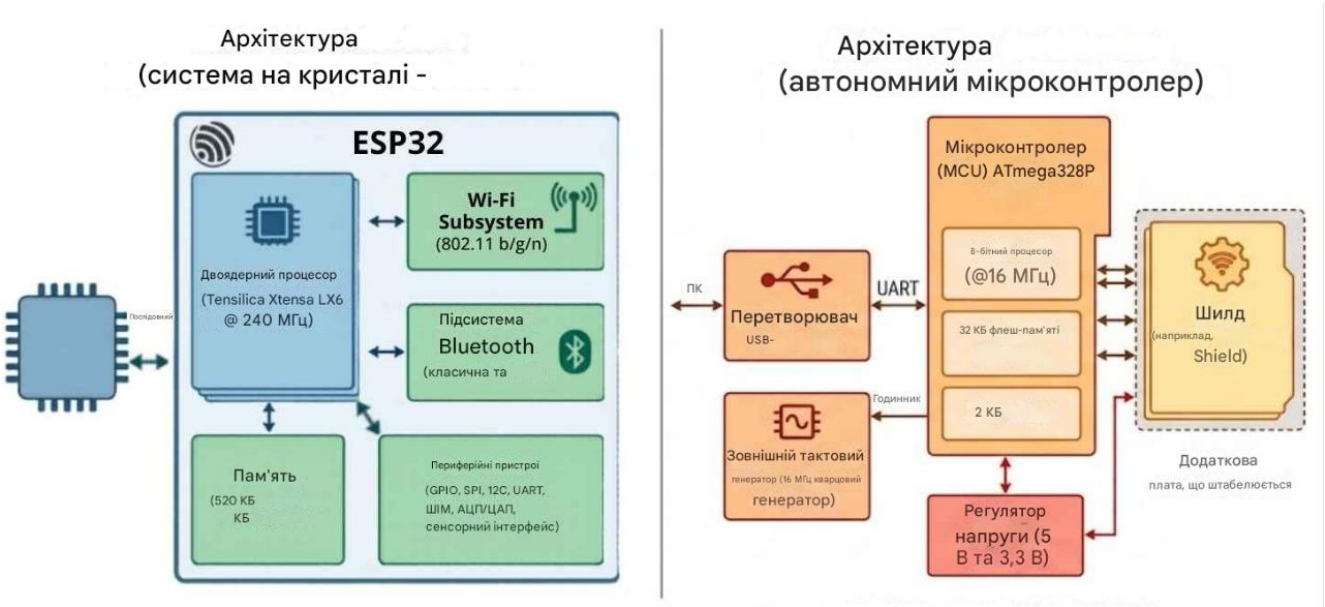


Рисунок 3.6 – Архітектури системи на кристалі та автономного мікроконтролера

ESP32 (Espressif Systems) – SoC на базі двоядерного процесора Xtensa LX6 з тактовою частотою 240 МГц. Вбудований Wi-Fi 802.11 b/g/n та Bluetooth 4.2/BLE реалізовані в тому ж кристалі. Обсяг ОЗП: 520 КБ SRAM, Flash: 4-16 МБ. Периферія: 34 GPIO, 18 каналів 12-бітного SAR-АЦП, два I²C-контролери, три UART, чотири SPI. Підтримує FreeRTOS для виконання задач у реальному часі. Напруга живлення: 3,0-3,6 В. Робочий температурний діапазон: −40°C … +85°C.

ESP32 також підтримує апаратне прискорення криптографічних операцій (AES, SHA, RSA), що дозволяє реалізовувати захищену передачу телеметрії через TLS без значного навантаження на процесор.

Наявність режимів енергозбереження Deep Sleep та Light Sleep дає змогу використовувати платформу не лише у стаціонарних, а й у автономних IoT-вузлах з живленням від акумулятора.

Arduino Uno (ATmega328P, Microchip) – автономний 8-бітний мікроконтролер з архітектурою AVR, тактова частота 16 МГц. ОЗП: 2 КБ SRAM, Flash: 32 КБ. 6 каналів 10-бітного АЦП. Відсутність вбудованого мережевого інтерфейсу вимагає підключення окремого Wi-Fi або Ethernet шилда, що значно

збільшує габарити вузла, кількість компонентів і потенційних точок відмови. Обсяг ОЗП у 2 КБ є критично малим для буферизації телеметрії від 16 GPU.

Arduino Nano (ATmega328P) – компактніший форм-фактор того ж мікроконтролера ATmega328P. Ідентичні апаратні обмеження щодо пам'яті та відсутності мережевого інтерфейсу.

STM32F103C8T6 (STMicroelectronics) – 32-бітний ARM Cortex-M3 з тактовою частотою 72 МГц. ОЗП: 20 КБ, Flash: 64-128 КБ. 16 каналів 12-бітного АЦП.

Попри кращі характеристики порівняно з ATmega328P, також є автономним MCU без вбудованого Wi-Fi – потребує зовнішнього модуля ESP8266 або аналога.

Raspberry Pi Pico W (RP2040) – двоядерний ARM Cortex-M0+ (133 МГц), вбудований Wi-Fi. ОЗП: 264 КБ. Критичне обмеження: лише 3 аналогових входи, що недостатньо для масштабування системи без мультиплексора.

До таблиці 3.4 занесемо порівняння характеристик описаних мікроконтролерів.

Таблиця 3.4 – Порівняння аналогово-цифрових конверторів

Характеристика	ESP32	Arduino Uno/Nano	STM32F103	RPi Pico W
Тип архітектури	SoC	Standalone MCU	Standalone MCU	SoC
Ядра / архітектура	2 × Xtensa LX6 (32-біт)	1 × AVR (8-біт)	1 × Cortex-M3 (32-біт)	2 × Cortex-M0+ (32-біт)
Тактова частота	240 МГц	16 МГц	72 МГц	133 МГц
ОЗП	520 КБ	2 КБ	20 КБ	264 КБ
Вбудований Wi-Fi	Так	Ні	Ні	Так
Кількість АЦП каналів	18 (12-біт)	6 (10-біт)	16 (12-біт)	3 (12-біт)
Інтерфейси	I²C, SPI, UART, 1-Wire*	I²C, SPI, UART, 1-Wire*	I²C, SPI, UART, 1-Wire*	I²C, SPI, UART, 1-Wire*

Продовження таблиці 3.4

Характеристика	ESP32	Arduino Uno/Nano	STM32F103	RPi Pico W
Зовнішній Wi-Fi модуль	Не потрібен	Потрібен	Потрібен	Не потрібен
Температурний діапазон	−40 … +85°C	−40 … +85°C	−40 … +85°C	−20 … +85°C

Обрано ESP32 як SoC-платформу, що найбільш повно відповідає вимогам розділу 2. Двоядерна архітектура дозволяє виконувати задачі опитування датчиків і підтримки мережевого стеку на різних ядрах під керуванням FreeRTOS без взаємного блокування. Інтеграція Wi-Fi у кристал зменшує кількість компонентів і точок відмови порівняно з рішеннями на базі ATmega328P або STM32 із зовнішніми модулями. Обсяг ОЗП 520 КБ є достатнім для буферизації телеметрії від 16 GPU з запасом. Arduino Uno та Nano відхилено через критично малий обсяг пам'яті та відсутність вбудованого Wi-Fi. STM32F103 відхилено з аналогічних причин. Raspberry Pi Pico W відхилено через обмежену кількість аналогових входів.

3.1.6 Дисплей для відображення показань у реальному часі

Дисплей встановлюється на вузлі граничного шлюзу для локального відображення поточних показань датчиків без звернення до веб-інтерфейсу.

SSD1306 (0,96" OLED) – монохромна матриця на органічних світлодіодах з роздільною здатністю 128×64 пікселі. Підключається по шині I²C. Кожен піксель світиться самостійно, тому дисплей не потребує загального підсвічування, що забезпечує низьке енергоспоживання (близько 15–20 мА) та високу контрастність. Живлення та логічні рівні сумісні з 3,3 В.

LCD1602 з модулем I2C – символьний рідкокристалічний дисплей (2 рядки по 16 символів) на базі контролера HD44780 з додатковим I²C-експандером PCF8574. Потребує постійного LED-підсвічування для забезпечення читабельності, що збільшує енергоспоживання. Значно більші габарити. Може відображати лише обмежену кількість символів, не дозволяє будувати графіки чи використовувати нестандартні шрифти.

TFT 1.8" (ST7735) – кольоровий рідкокристалічний дисплей з роздільною здатністю 128×160 пікселів. Інтерфейс підключення – SPI. Забезпечує кольорове зображення, але вимагає значно більшої кількості виводів мікроконтролера (MOSI, SCLK, CS, DC, RST) порівняно з I²C. Високе енергоспоживання через постійно активне LED-підсвічування.

До таблиці 3.5 занесемо порівняння характеристик описаних дисплеїв.

Таблиця 3.5 – Порівняння дисплеїв

Характеристика	SSD1306 OLED (0.96")	LCD1602 (I2C)	TFT ST7735 (1.8")
Технологія	OLED	РК (LCD)	РК (TFT)
Інтерфейс	I²C	I²C	SPI
Кількість задіяних GPIO	2 (SDA, SCL)	2 (SDA, SCL)	5 (SPI + керування)
Роздільна здатність / Ємність	128×64 пікселів	16 символів × 2 рядки	128×160 пікселів
Споживання струму (активне)	~15–20 мА	~40–80 мА	~50–100 мА
Підсвічування	Не потребує	Обов'язкове	Обов'язкове
Габарити	Компактні	Великі	Середні

Обрано OLED-дисплей SSD1306 (0,96", 128×64 пікселі). Підключається по шині I²C (два провідники: SDA та SCL), яка вже задіяна для ADS1115, – таким чином дисплей не займає додаткових GPIO. Споживання струму в активному режимі становить близько 15-20 мА; підтримується програмне вимкнення підсвічування. Монохромна матриця на органічних світлодіодах (OLED) не потребує підсвічування, що знижує споживання у порівнянні з LCD-аналогами. Роздільна здатність 128×64 пікселі дозволяє відображати 4–8 рядків інформації з температурами GPU, запиленістю та вологістю одночасно. Стандартна адреса I²C: 0x3C (або 0x3D при зміні перемички ADDR). Схему розташування виводів дисплею відобразимо на рисунку 3.7.

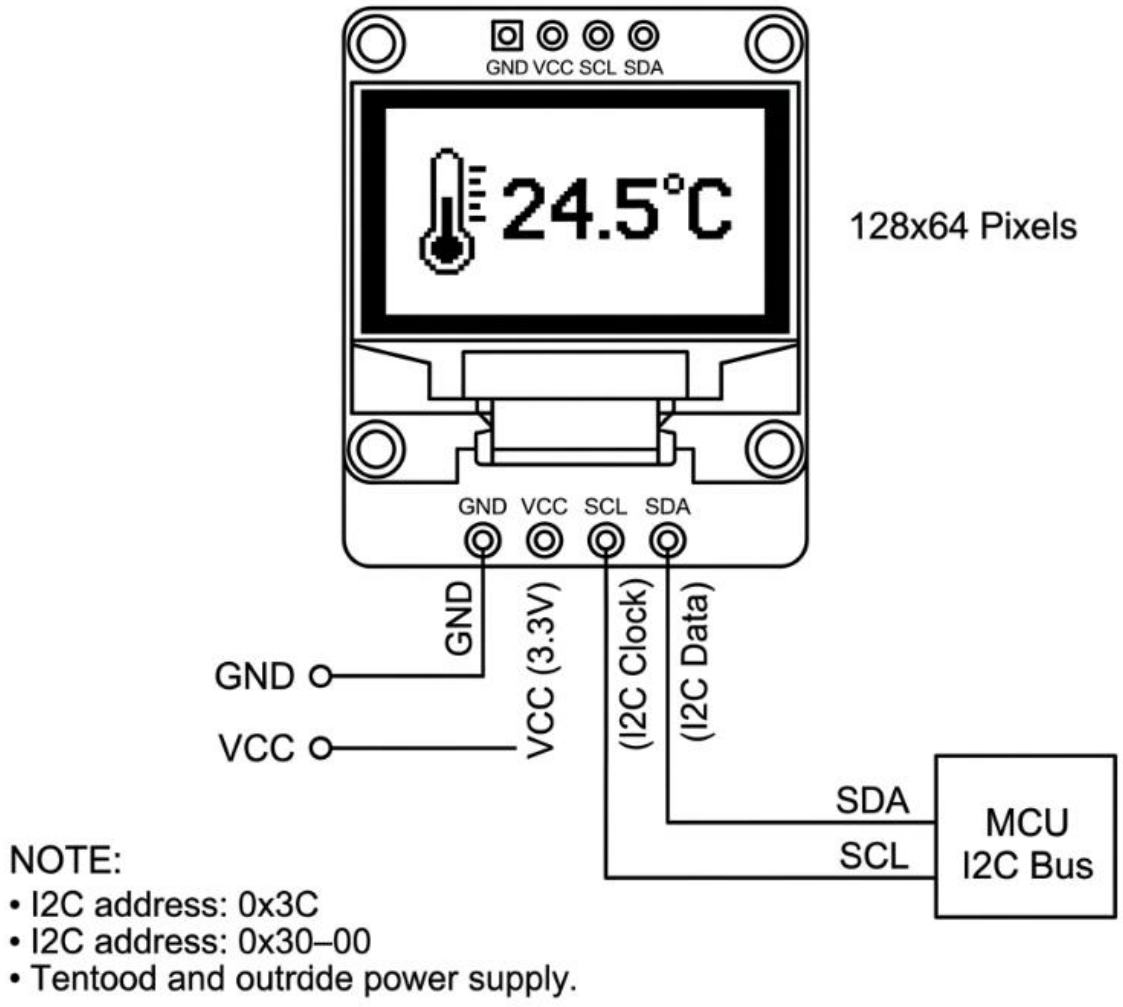


Рисунок 3.7 – Схема розташування виводів дисплею

На рисунку 3.8 наведено узагальнену схему підключення всіх компонентів апаратного рівня до граничного шлюзу та подальшої передачі телеметрії до вузла туманних обчислень.

Температурний сенсор DS18B20 працює через інтерфейс 1-Wire, що дозволяє використовувати мінімальну кількість сигнальних ліній при збереженні стабільної передачі даних. DHT22 використовує однопровідний цифровий інтерфейс для передачі показників температури та вологості.

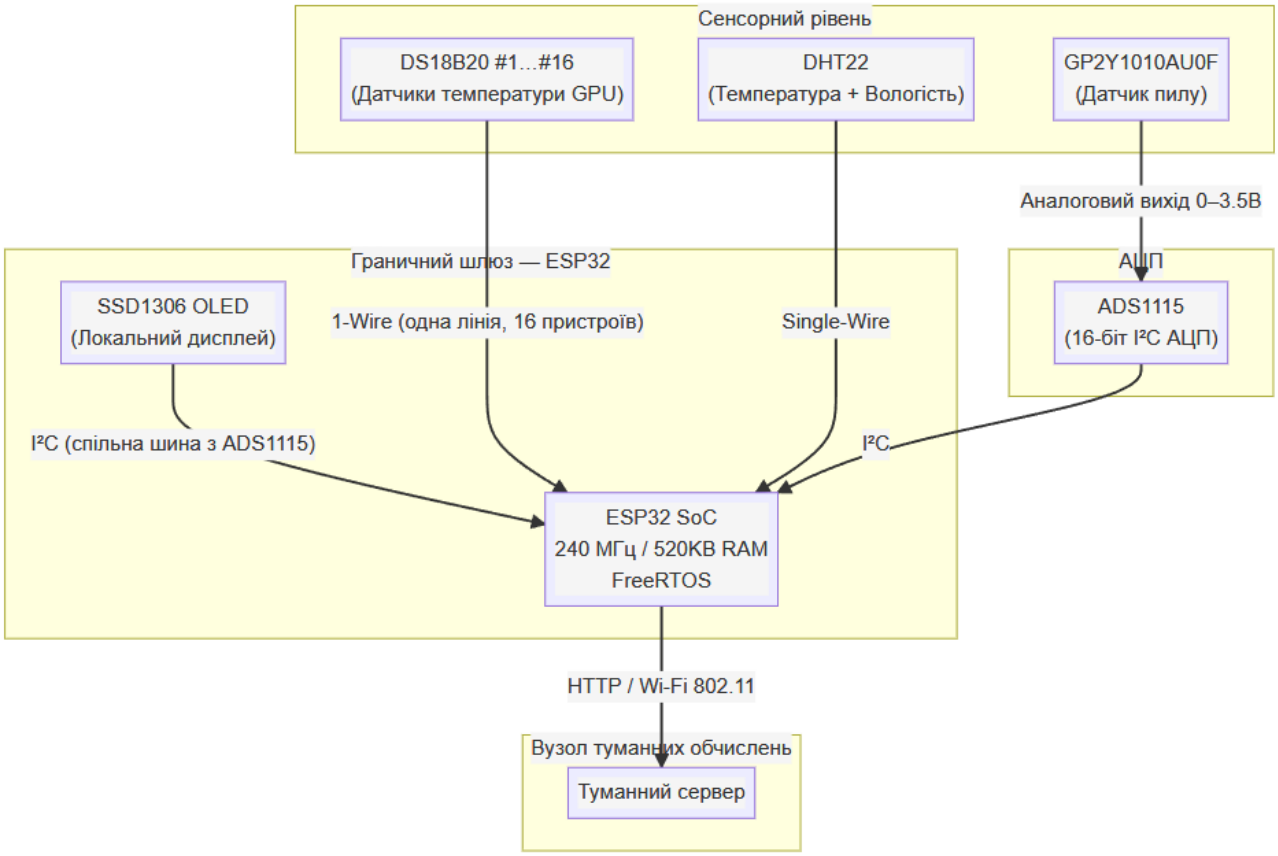


Рисунок 3.8 – Структурна схема підключення обраних компонентів апаратного рівня

3.2 База даних

Кожен граничний шлюз щосекунди генерує серію спостережень: температура кожного з 16 GPU, вологість, запиленість, стан вентиляторів (ШІМ, обороти). Усі ці записи мають таку ключову властивість – вони невід'ємно прив'язані до моменту часу та надходять з постійною частотою. Таку структуру в теорії баз даних прийнято називати часовим рядом (Time Series).

У даній системі реляційний підхід, де центральним поняттям є сутність з унікальним ідентифікатором та зв'язки між сутностями є невиправданим з таких причин. Між записами телеметрії немає реляційних зв'язків, що потребують підтримки через зовнішні ключі – кожен запис є самодостатнім спостереженням.

Натомість кількість операцій запису є екстремально высокой: при 16 GPU з частотою опитування кожні 5 секунд і множиною параметрів система генерує десятки тисяч точок на годину. Реляційна СУБД, оптимізована для транзакційних читань та JOIN-запитів, зазнає деградації продуктивності при такому навантаженні. При цьому більшість реляційних запитів до IoT-телеметрії є однотипними: «надай мені всі значення параметра X за проміжок часу T_1–T_2» – запит, де часова вісь є єдиним виміром для фільтрації.

Вимоги до сховища: оптимізований запис потоку метрик з постійною частотою без деградації при зростанні обсягу; ефективна вибірка за часовим діапазоном; підтримка агрегаційних функцій (середнє, мін, макс) по часових вікнах; автоматичне керування утриманням даних; структурована індексація метаданих (ідентифікатор пристрою, назва метрики).

PostgreSQL (з розширенням TimescaleDB) – реляційна СУБД з відкритим кодом, де розширення TimescaleDB додає підтримку гіпертаблиць (hypertable) для зберігання часових рядів. Дозволяє поєднувати реляційні та часові дані в одній системі. Проте СУБД залишається по суті реляційною: зберігає дані в рядково-орієнтованому форматі, що є менш ефективним для агрегаційних запитів по стовпцях, ніж спеціалізовані TSDB. Для розгортання потребує складнішої конфігурації, ніж нативна часова БД.

MongoDB – документно-орієнтована СУБД. Кожен запис зберігається як JSON-документ без фіксованої схеми. Має вбудовані можливості для роботи з часовими колекціями (Time Series Collections, починаючи з версії 5.0). Проте MongoDB проектувалась для зберігання складних вкладених документів з варіативною структурою, а не для потоків однорідних числових спостережень. Агрегаційний конвеєр (Aggregation Pipeline) значно складніший у синтаксисі порівняно зі спеціалізованими мовами запитів TSDB. Розмір зберігання є вищим через накладні витрати формату BSON.

Redis (з модулем RedisTimeSeries) – in-memory сховище типу «ключ–значення» з опціональним модулем RedisTimeSeries. Забезпечує надзвичайно низьку затримку читання та запису завдяки зберіганню даних у оперативній

пам'яті. Проте основний обсяг даних обмежений обсягом RAM: збереження повної багатомісячної телеметрії від 16 GPU у Redis вимагатиме значних обсягів пам'яті та застосування персистентних механізмів (RDB/AOF), що збільшує складність. Redis оптимально підходить як кеш або брокер черг між компонентами, але не як основне сховище IoT-телеметрії.

InfluxDB також підтримує мову запитів Flux (а також сумісність з InfluxQL у попередніх версіях), що дозволяє виконувати складні аналітичні операції над часовими рядами, включаючи фільтрацію, агрегацію, трансформації та обчислення похідних метрик. Архітектурно система розділяє процеси запису (write path) та читання (read path), що підвищує масштабованість і дозволяє ефективно обробляти високочастотні потоки даних без деградації продуктивності. Дані організовуються у вигляді measurement-ів, тегів (indexed metadata) та полів (field values), що забезпечує оптимізований доступ як до аналітичних, так і до пошукових операцій. Додатково InfluxDB підтримує політики зберігання (Retention Policies) та downsampling через Continuous Queries або Flux tasks, що дозволяє автоматично зменшувати обсяг історичних даних без втрати аналітичної цінності [12].

Структурні концепції InfluxDB, що відрізняють її від реляційних баз даних:

– measurement – аналог таблиці, але без фіксованої схеми стовпців; ідентифікує тип метрики;

– tags – індексовані метадані у форматі «ключ–значення». Tags є індексованими – по них відбувається ефективна фільтрація без повного сканування;

– fields – числові значення спостережень (температура, ШІМ, вологість). Fields не індексуються, але оптимально стискаються;

– timestamp – обов'язкова часова мітка кожної точки з нано-секундною роздільною здатністю. Є первинним ключем впорядкування.

До таблиці 3.6 занесемо порівняння розглянутих баз даних.

Таблиця 3.6 – Порівняння баз даних

Характеристика	InfluxDB 2.x	PostgreSQL + TimescaleDB	MongoDB 7 (TS)	Redis + RedisTimeSeries
Формат зберігання	Стовпцевий (TSM)	Рядковий (heap) + компресія	BSON-документи	In-memory структури
Стиснення числових рядів	до 10–20% від вихідного розміру	30–60% (TimescaleDB compress)	40–70% (BSON overhead)	Відсутнє (в ОЗП)
Максимальна пропускна здатність запису	> 1 млн точок/с (batch)	~100–300 тис. рядків/с	~200–500 тис. документів/с	> 5 млн операцій/с (в ОЗП)
Обмеження обсягу	Дисковий простір	Дисковий простір	Дисковий простір	Обсяг ОЗП
Агрегація по часових вікнах	Вбудована	через TimescaleDB	Aggregation Pipeline	Вбудована
Retention Policy	Вбудована (Tasks + bucket TTL)	pg_partman / cron	TTL-індекс	TTL на ключ
Індексація метаданих пристрою	Інвертований індекс по тегах	B-tree індекс по стовпцях	Індекс по metaField	Ключ рядка
Нативна підтримка JOIN	Відсутня	Повна	Відсутня	Відсутня
Локальне розгортання	Так	Так	Так	Так
Керований хмарний SaaS	InfluxDB Cloud	AWS RDS, Google CloudSQL	MongoDB Atlas	Redis Cloud

Обрано InfluxDB 2.x. Стовпцевий рушій TSM скорочує обсяг зберігання числових рядів до 10–20% від вихідного розміру без втрат. При 16 GPU з інтервалом 5 секунд і 6 числовими полями на кожний 90-денний архів займає орієнтовно 2–4 ГБ після стиснення – проти ~15–25 ГБ у рядковому PostgreSQL без

TimescaleDB-компресії. InfluxDB розгортається як локальний Docker-контейнер на вузлі туманних обчислень без залежності від зовнішньої мережі – весь цикл «запис → запит → агрегація → Retention» виконується локально. При переході до промислового масштабу та необхідності централізованого доступу до архіву той самий клієнтський код без змін підключається до InfluxDB Cloud через зміну URL та токена у конфігурації. Retention Policy видаляє застарілі записи автоматично: налаштування терміну зберігання виконується один раз на рівні bucket і не потребує зовнішніх планувальників. Tasks дозволяють агрегувати хвилинні дані у годинні середні за заданим розкладом – стандартна практика downsampiling для IoT-архівів. Відносно Redis: при 16 GPU, 6 полях і інтервалі 5 секунд обсяг даних за 90 діб становить близько 49 мільйонів точок. Зберігання цього обсягу в ОЗП Redis вимагає не менше 8–12 ГБ RAM лише під числові ряди. Redis доцільний як буфер черги між граничним шлюзом і InfluxDB, але не як архівне сховище.

3.3 Платформа вузла туманних обчислень

Вузол туманних обчислень є центральним компонентом системи: він отримує телеметрію від граничного шлюзу, виконує алгоритми адаптивного керування, зберігає дані в InfluxDB та надає API для веб-інтерфейсу. Вимоги до платформи: можливість реалізації довільних алгоритмів керування (включно з прогностичними моделями та індексами деградації); підтримка асинхронної обробки запитів для мінімізації затримки; прозора інтеграція з InfluxDB; підтримка контейнеризації; мінімальне споживання ресурсів на апаратному вузлі туману; легкість модифікації алгоритмів без перекомпіляції.

EdgeX Foundry – відкрита мікросервісна платформа для туманних обчислень під управлінням Linux Foundation [16]. Архітектурно складається з набору незалежних мікросервісів (Device Services, Core Services, Support Services, Application Services), що взаємодіють через шину повідомлень (Redis Pub/Sub або

MQTT). Підтримує широкий реєстр протоколів підключення пристроїв: OPC-UA, Modbus, BACnet, REST. Однак EdgeX Foundry є надважкою платформою для даного проекту: стандартне розгортання містить понад 12 Docker-контейнерів, що вимагає значних апаратних ресурсів (RAM > 2 ГБ лише для ядра платформи). Алгоритми керування реалізуються через Application Services на Go або Python, але вбудована бізнес-логіка жорстко обмежена архітектурою платформи. Налаштування EdgeX для нестандартного алгоритму керування (такого як каскадний алгоритм охолодження з гістерезисом та індексами деградації) вимагає глибокої інтеграції з внутрішніми механізмами платформи.

AWS IoT Greengrass – керована платформа Amazon для розгортання хмарної логіки на периферійних пристроях (Edge/Fog). Дозволяє розгортати Lambda-функції або компоненти Greengrass безпосередньо на локальному вузлі з синхронізацією з хмарою AWS. Висока зрілість та інтеграція з екосистемою AWS (S3, CloudWatch, ML-моделі через SageMaker). Проте система повністю залежна від хмари AWS: без активного підключення до Інтернету та акаунта AWS платформа не функціонує повноцінно. Це суперечить концепції туманних обчислень, де вузол повинен функціонувати автономно навіть при відсутності зовнішньої мережі. Ліцензування є комерційним, вартість масштабується зі зростанням навантаження.

Azure IoT Edge – аналогічна платформа Microsoft для розгортання модулів (контейнерів) на периферії з управлінням через Azure IoT Hub. Підтримує розгортання ML-моделей через Azure ML. Так само, як і AWS Greengrass, залежна від хмарної інфраструктури Azure для оркестрування та моніторингу. Для автономного локального розгортання потребує складного налаштування офлайн-режиму. Привносить надмірну складність для однорівневої системи з одним вузлом туману.

Замість застосування готових IoT-платформ загального призначення обрано підхід розробки власного туманного сервера на мові Python з використанням фреймворку FastAPI. Ключова перевага власної реалізації – алгоритм охолодження, алгоритм контролю мікроклімату, розрахунок індексів корозії та зносу вентиляторів повністю визначаються розробником.

У платформах типу EdgeX зміна логіки керування потребує адаптації до внутрішньої шини та архітектури Application Services. У власному сервері будь-який алгоритм переписується безпосередньо в Python-функції без перекомпіляції та без знання внутрішніх механізмів сторонньої платформи.

Власна кодова база проектується з явним розподілом відповідальності між модулями: маршрутизація запитів, алгоритм мікроклімату, алгоритм терморегуляції, індекси деградації. Кожен модуль може бути замінений або розширений незалежно. Подібний рівень модульності в EdgeX вимагає розробки окремого мікросервісу з усіма накладними витратами на комунікацію між сервісами. На відміну від AWS Greengrass та Azure IoT Edge, власний сервер функціонує повністю локально без будь-якої залежності від зовнішніх сервісів. Це відповідає ключовому принципу туманних обчислень: вузол туману має бути автономним і продовжувати роботу при відсутності з'єднання з Інтернетом.

Python обраний як основна мова реалізації туманного сервера. Python має найбагатшу екосистему бібліотек для задач, характерних для системи: numpy та scipy для реалізації математичних моделей теплової динаміки та розрахунку індексів за формулою Арреніуса; scikit-learn та torch для майбутньої інтеграції алгоритмів машинного навчання (наприклад, предиктивне моделювання часу до відмови вентилятора на основі тренду FWI); influxdb-client – офіційний клієнт InfluxDB з нативною підтримкою асинхронного запису.

FastAPI обраний серед Python-фреймворків за такими критеріями: нативна підтримка асинхронного виконання (async/await) на базі ASGI-сервера Uvicorn, що критично для IoT з високою частотою запитів; автоматична генерація OpenAPI-документації (Swagger UI) без додаткового коду; строга типізація через Pydantic, що забезпечує валідацію телеметрії на вхідному шарі.

Вибір Python відкриває можливість безшовної інтеграції ML-моделей у майбутньому: передбачення температурного піку до фактичного перевищення порогу, класифікація аномальних режимів роботи, адаптивне калібрування порогів гістерезису на основі статистики реальних навантажень.

Жодна з розглянутих IoT-платформ не надає такого рівня інтеграції з Python ML-стеком без зовнішніх адаптерів. До таблиці 3.7 занесемо порівняння розглянутих рішень для серверу туманних обчислень.

Таблиця 3.7 – Порівняння платформ туманного сервера

Характеристика	Власний сервер (FastAPI/Python)	EdgeX Foundry	AWS Greengrass v2	Azure IoT Edge
Залежність від хмари	Відсутня	Відсутня	Обов'язкова	Обов'язкова
Кількість сервісів при розгортанні	1 (+ InfluxDB)	12+	3+ (+ хмарні агенти)	3+ (+ хмарні агенти)
Споживання ОЗП	< 100 МБ	> 2 ГБ	> 512 МБ	> 512 МБ
Свобода зміни алгоритмів	Повна	Обмежена архітектурою	Обмежена Lambda/компонентами	Обмежена модулями
Підтримка ML-бібліотек Python	Нативна	Через адаптери	Через SageMaker	Через Azure ML
Контейнеризація	Docker / Docker Compose	Docker (обов'язково)	Greengrass Core	IoT Edge Runtime
Ліцензія	Відкрита (MIT)	Apache 2.0	Комерційна (AWS)	Комерційна (Azure)
Підтримка офлайн-роботи	Повна	Повна	Обмежена	Обмежена

3.4 Стек розробки веб-системи

Веб-система виконує функцію рівня представлення: вона відображає актуальний стан кластера, надає засоби ручного керування виконавчими механізмами та дозволяє переглядати архівну телеметрію у вигляді часових графіків. З аналізу функціональних вимог розділу 2 виокремлено такі групи задач.

Перша група – відображення поточного стану у реальному часі. Дашборд з інтервалом 5 секунд опитує REST API туманного сервера і оновлює показники температур 16 GPU, стан вентиляторів з поточними ШІМ та оборотами, параметри мікроклімату і активні сповіщення. Відображення є двосекційним: карткова сітка GPU і карткова сітка вентиляторів з індикаторами стану.

Друга група – відображення часових рядів з InfluxDB. Сторінка історії відображає до 8 незалежних часових рядів: температури GPU, ШІМ і оберти вентиляторів, вологість, запиленість, індекс корозії CI, індекс зносу FWI та кореляційний оверлей усіх метрик на одному полотні. Глибина запиту – 1, 3, 6 або 24 години. Режим відображення перемикається між повноекранним видом одного ряду і сіткою мініатюр усіх рядів одночасно.

Третя група – керування системою. Сторінка ручного керування дозволяє перемикати режим між автоматичним і ручним та задавати ШІМ для кожного вентилятора індивідуально. Адміністративна панель перемикає профіль середовища між п'ятьма сценаріями і скидає акумульовані індекси зносу після технічного обслуговування.

Четверта група – сторінка трендів, яка відображає поточні значення CI і FWI, рівень ризику та інтерпретацію значень, розраховані модифікатори ефективності охолодження.

З цих вимог випливають технічні обмеження на стек: клієнт повинен виконувати HTTP-запити асинхронно без перезавантаження сторінки; бібліотека для побудови графіків зобов'язана рендерити до 16 ліній на одному графіку при

частоті оновлення 5 секунд; вся програма розгортається у Docker-контейнері разом із туманним сервером.

Vue 3 + Vite + Chart.js. Vue 3 є реактивним SPA-фреймворком із Composition API. Chart.js – бібліотека на Canvas з широким набором типів діаграм. Canvas-рендеринг є продуктивним для великих масивів, проте Chart.js не надає декларативного React-подібного API для побудови складних мультисерійних графіків. Конфігурація Chart.js для кореляційного overlay-графіку з двома осями Y потребує значного обсягу налаштувань через imperative API. Vite забезпечує швидку збірку, проте Vue 3 не має SSR-рівня Next.js, що обмежує можливість серверного попереднього рендерингу.

Angular 17 + ng2-charts. Angular – повноцінний MVC-фреймворк з вбудованою залежністю ін'єкцій, роутингом і TypeScript за замовчуванням. ng2-charts є обгорткою над Chart.js. Angular є надмірно важким для задачі: стандартний bundle перевищує 300 КБ gzip навіть при агресивному tree-shaking; вартість налаштування і онбордингу висока порівняно із задачами, які реалізує дана система.

Vanilla JS + Fetch API + Chart.js без фреймворку. Найлегший варіант за розміром bundle, але без реактивності стан UI необхідно оновлювати вручну через DOM-маніпуляції. При 16 GPU-картках і 8 графіках, кожен з яких оновлюється незалежно, відсутність компонентної моделі призводить до значного ускладнення коду підтримки стану.

Next.js 16 + React 19 + Recharts + Tailwind CSS 4. Next.js побудований поверх React і надає App Router з підтримкою серверних і клієнтських компонентів, автоматичне розділення коду по маршрутах, вбудований сервер розробки і оптимізований production-build. React 19 включає компілятор із вбудованою мемоізацією без ручного використання додаткових хуків. Recharts – декларативна бібліотека на SVG, побудована поверх React і D3, яка рендерить кожен елемент графіку як React-компонент. Tailwind CSS 4 генерує лише ті CSS-класи, що присутні у вихідному коді.

Порівняння характеристик розглянутих стеків технологій для розробки веб-системи занесемо до таблиці 3.8.

Таблиця 3.8 – Порівняння технологій для веб-системи

Характеристика	Next.js + React 19 + Recharts	Vue 3 + Vite + Chart.js	Angular 17 + ng2-charts	Vanilla JS + Chart.js
JS bundle gzip, мінімальний	~90 КБ	~70 КБ	~300 КБ	~60 КБ
Реактивна компонентна модель	React 19 Compiler	Composition API	Angular DI	Відсутня
SSR і гібридний рендеринг	App Router вбудований	Nuxt, окремо	Angular Universal, окремо	Відсутній
API побудови графіків	Recharts, декларативний JSX	Chart.js, imperative config	Chart.js, imperative config	Chart.js, imperative config
TypeScript	Нативно	Опціонально	Нативно	Відсутня
Routing	File-system routing	Vue Router	Angular Router	Ручний
Production CSS bundle	< 15 КБ після purge	Залежить від підходу	~50 КБ	Довільний
Docker multi-stage build	node:alpine, < 200 МБ образ	node:alpine	node:alpine	nginx + static

Обрано Next.js 16 із React 19, Recharts 3, Tailwind CSS 4 і TypeScript 5. Recharts рендерить кожну серію і кожну точку як SVG-елемент у дереві React. Оновлення одного значення – наприклад, нова температура GPU-3 за поточну секунду – ініціює перерендеринг лише тих SVG-вузлів, що змінились, без перемальовування усього canvas. Для графіку з 16 лініями і частотою опитування 5 секунд це критично: при canvas-рендерингу Chart.js кожне оновлення даних

повністю перемальовує полотно. Recharts також надає нативний компонент ComposedChart, в якому кореляційний overlay-графік – накладення середньої температури, середнього ШІМ, вологості і запиленості на одному полотні з двома осями Y – описується декларативно через JSX без додаткової конфігурації.

Вибір Next.js з App Router зумовлений поділом сторінок на серверні і клієнтські компоненти. Статичні частини UI – розмітка, навігація, заголовки – рендеруються на сервері і надходять до браузера як готовий HTML. Клієнтська частина – опитування API, управління станом, графіки – виконується в браузері і позначається директивою use client. Такий поділ зменшує час до першого рендерингу і скорочує обсяг JavaScript, який браузер виконує при першому відкритті сторінки.

TypeScript у поєднанні з Pydantic-схемами туманного сервера утворює наскрізну типізацію: кожен інтерфейс у api.ts відповідає полям JSON-відповіді, яку FastAPI генерує з Pydantic-моделей. Помилка у назві поля або типі даних виявляється на етапі компіляції TypeScript без потреби у ручному тестуванні. Це скорочує цикл налагодження при змінах API. Tailwind CSS 4 компілює лише класи, наявні у TSX-файлах. Production CSS bundle складає менше 15 КБ після gzip. Адаптивна сітка реалізується через breakpoint-класи, що задаються безпосередньо в JSX без окремих CSS-файлів і без розбіжності між розміткою і стилями.

Next.js запускається у Docker через офіційний multi-stage Dockerfile: перший stage збирає production bundle, другий копіює тільки необхідні артефакти в мінімальний node:alpine-образ. Розмір фінального Docker-образу – менше 200 МБ. Адреса туманного сервера задається через змінну середовища NEXT_PUBLIC_API_URL, що дозволяє перемикати цільове середовище через один параметр у Docker Compose без перебудови образу. Сторінки веб-системи структуровані через File System Router Next.js. Додавання нової сторінки не потребує модифікації конфігурації роутера – достатньо створити файл page.tsx у відповідній директорії. Такий підхід відповідає принципу модульності, закладеному в архітектурі туманного сервера, і дозволяє розширювати веб-систему незалежно від інших компонентів.

4 РОЗРОБКА СИСТЕМИ

4.1 Розробка сервера туманних обчислень

Сервер туманних обчислень реалізовано на Python 3.11. Усі зовнішні залежності зафіксовано у файлі requirements кореневої директорії модуля fog-server, вміст якого відображено у лістингу 4.1. Для розгортання у ізольованому середовищі використовується Docker; для локальної розробки – стандартне Python-оточення venv

Лістинг 4.1 – Вміст файлу залежностей fog-сервера:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.9.2
pydantic-core==2.23.4
pydantic-settings==2.1.0
influxdb-client==1.38.0
python-dotenv==1.0.0
aiohttp==3.9.1
numpy>=2.0.0
scipy>=1.14.0
requests==2.31.0
pyyaml==6.0.1
colorama==0.4.6
```

FastAPI версії 0.104.1 разом з ASGI-сервером Uvicorn утворюють асинхронний HTTP-шар. Pydantic 2.x виконує строгу валідацію вхідних JSON-пакетів телеметрії безпосередньо на рівні десеріалізації – некоректно сформований пакет відхиляється до того, як потрапить в будь-який алгоритм. Бібліотека influxdb-client є офіційним Python-драйвером для запису і запиту InfluxDB. numpy і scipy використовуються в модулі AdvancedTrendCalculator для розрахунку накопичувальних індексів деградації за формулами, визначеними у підрозділі 2.5.

python-dotenv зчитує конфігураційні параметри – URL та токен InfluxDB, кількість GPU, порогові температури – з файлу .env що дозволяє змінювати налаштування без перебудови Docker-образу.

До рисунку 4.1 занесемо архітектуру модулю сервера туманних обчислен.

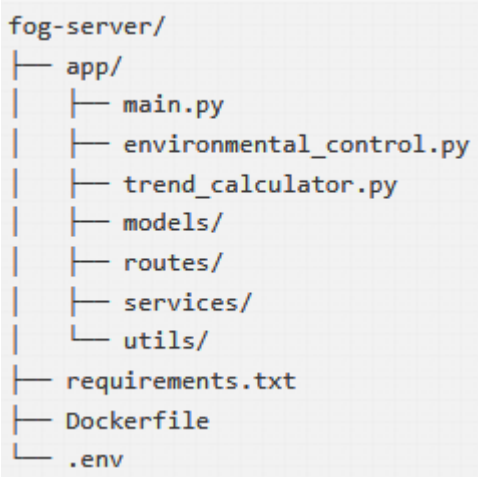


Рисунок 4.1 – Файлова архітектура fog-сервера

main.py виконує роль оркестратора: визначає всі HTTP-маршрути, ініціалізує з'єднання з InfluxDB через клас InfluxDBManager, і, при старті застосунку через менеджер контексту lifespan, підключається до бази даних і створює екземпляр AdvancedTrendCalculator. environmental_control.py і trend_calculator.py – ізольовані модулі з алгоритмами; вони не знають про HTTP-шар і залежать лише від стандартної бібліотеки та numpy. Це дозволяє тестувати алгоритмічні класи незалежно від сервера.

Розробка main.py ведеться за принципом поступового нарощування: спочатку визначаються моделі даних, потім клас доступу до бази даних, потім алгоритмічні класи, і в кінці – FastAPI-застосунок із маршрутизацією. Зовнішні алгоритмічні модулі (environmental_control.py, trend_calculator.py) підключаються імпортом і використовуються як незалежні компоненти з чітко визначеним інтерфейсом.

Структура асоціацій між класами main.py наведена на рисунку 4.2.

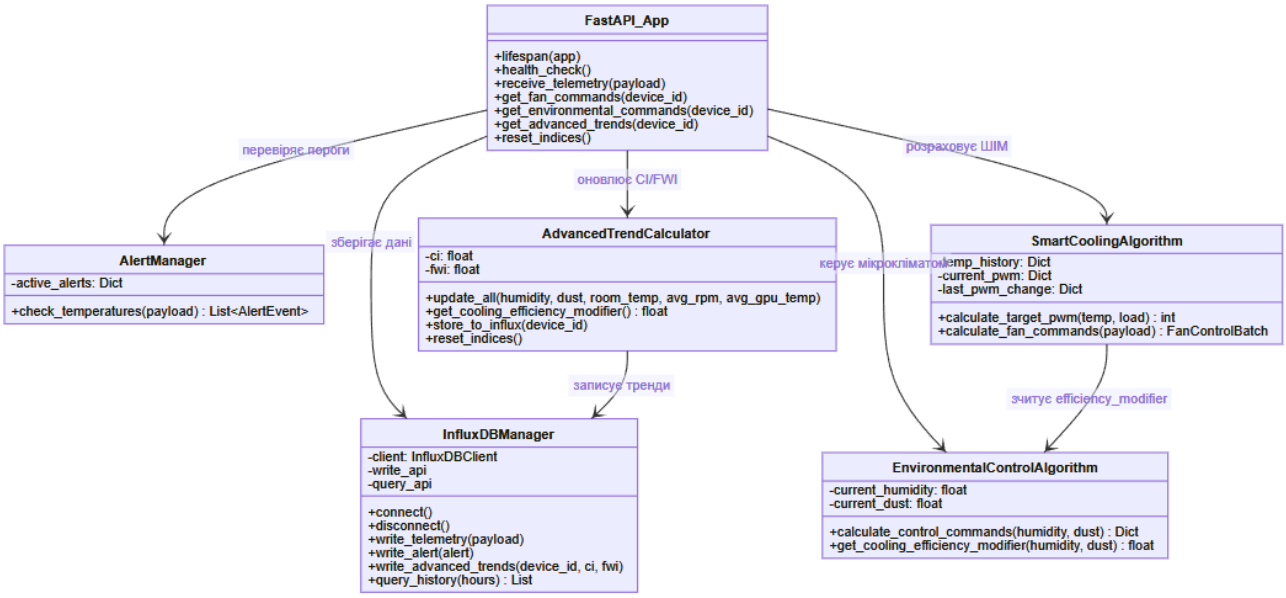


Рисунок 4.2 – Схема асоціацій між класами головного файлу fog-сервера

Файли environmental_control.py та trend_calculator.py – ізольовані модулі без залежності від HTTP-шару. main.py імпортує їхні singleton-екземпляри (environmental_control_algo, advanced_trend_calc) і викликає методи їх публічного інтерфейсу. SmartCoolingAlgorithm зчитує efficiency_modifier безпосередньо з екземпляра environmental_control_algo – це єдина міжмодульна залежність, що не проходить через FastAPI.

Усі параметри середовища зчитуються через клас Config. При запуску сервера менеджер контексту lifespan встановлює з'єднання з InfluxDB. На відміну від традиційних реляційних баз даних, InfluxDB оптимізована для часових рядів і вимагає специфічного підключення через InfluxDBClient.

Ініціалізація клієнта відбувається в класі InfluxDBManager. Підключення створюється із вказанням url, token та org. Для забезпечення високої пропускної здатності без блокування основного потоку FastAPI, об'єкт write_api конфігурується у пакетному (batching) режимі із використанням оптимізованих пулів запису. Фрагмент ініціалізації БД та lifespan наведено у лістингу 4.2.

Лістинг 4.2 – Конфігурація та ініціалізація БД InfluxDB (app/main.py)

```
class Config:
    INFLUXDB_URL   = os.getenv("INFLUXDB_URL",
"http://localhost:8086")
    INFLUXDB_TOKEN = os.getenv("INFLUXDB_TOKEN")
    INFLUXDB_ORG   = os.getenv("INFLUXDB_ORG",   "gpu-monitoring")
    INFLUXDB_BUCKET= os.getenv("INFLUXDB_BUCKET", "gpu-metrics")
    GPU_COUNT      = int(os.getenv("GPU_COUNT", 8))

class InfluxDBManager:
    def connect(self):
        self.client = InfluxDBClient(url=Config.INFLUXDB_URL,
token=Config.INFLUXDB_TOKEN, org=Config.INFLUXDB_ORG)
        self.write_api =
self.client.write_api(write_options=SYNCHRONOUS)
        self.query_api = self.client.query_api()

@asynccontextmanager
async def lifespan(app: FastAPI):
    influx_manager.connect()
    global advanced_trend_calc
    advanced_trend_calc = AdvancedTrendCalculator(influx_manager)
    yield
    influx_manager.disconnect()
```

Усі точки даних упаковуються в об'єкти Point і записуються однією операцією self.write_api.write(bucket=Config.INFLUXDB_BUCKET, record=points), що мінімізує мережеві витрати на встановлення з'єднання. Усі вхідні та вихідні структури даних визначено через Pydantic-класи. Це відповідає принципу наскрізної типізації: JSON-поля, визначені тут, відображаються у TypeScript-інтерфейсах веб-клієнта. Ключові моделі наведено у лістингу 4.3. TelemetryPayload є головним вхідним контрактом ендпоінту прийому телеметрії, а FanControlBatch – вихідним контрактом, що повертається граничному шлюзу або веб-клієнту.

Лістинг 4.3 – Pydantic-моделі телеметрії

```
class GPUTemperature(BaseModel):
    gpu_id:      int   = Field(..., ge=1, le=16)
    temperature: float
    load:        float = Field(0.0, ge=0.0, le=100.0)

class TelemetryPayload(BaseModel):
    device_id: str
    timestamp: str
    sensors:   SensorData
```

```
    fans:       FanData

class FanControlBatch(BaseModel):
    device_id: str
    commands:  List[FanControlCommand]
```

Для забезпечення повноцінної взаємодії між граничним шлюзом, веб-інтерфейсом та алгоритмічним ядром, у main.py зареєстровано повну мапу маршрутизації (API endpoints). Усі ендпоінти працюють асинхронно. Ця структура маршрутизації повністю ізолює апаратний рівень від веб-фронтенду, забезпечуючи виконання концепції туманних обчислень:

– POST /api/v1/telemetry – основний ендпоінт для прийому телеметрії від апаратного стенду. Виконує повний цикл: серіалізацію TelemetryPayload, запис в InfluxDB, перевірку критичних температур, виклик SmartCoolingAlgorithm (у режимі AUTO) та оновлення кумулятивних індексів CI/FWI;

– GET /api/v1/fan-control/{device_id} – ендпоінт для отримання апаратним шлюзом розрахованих команд ШІМ. Сервер повертає масив команд з буфера pending_commands, який був заповнений на попередньому кроці обробки телеметрії. Якщо команд немає, повертається HTTP 204 No Content;

– GET /api/v1/environmental-control/{device_id} – аналогічний ендпоінт для отримання команд керування кліматичним обладнанням (вмикання/вимикання зволожувача та осушувача);

– GET /api/v1/trends/advanced/{device_id} – аналітичний ендпоінт для веб-клієнта. Повертає поточні значення індексів CI та FWI, а також штрафний коефіцієнт efficiency_modifier, що застосовується до вентиляторів;

– POST /api/v1/admin/reset-indices та POST /api/v1/admin/change-profile – адміністративні маршрути, які ініціюються інженером через веб-інтерфейс для обнулення індексів деградації (наприклад, после технічного обслуговування) або зміни профілю зовнішнього середовища у системі.

Алгоритм SmartCoolingAlgorithm реалізує вимоги до адаптивного терморегулювання, визначені у підрозділі 2.5.1: він ураховує теплову інерцію системи через гістерезис та попереджувальне збільшення ШІМ при зростанні

навантаження. Клас зберігає HISTORY_SIZE = 3 останніх температурних відліків для кожного GPU, на основі яких визначає теплове середовище (HEATING, COOLING, STEADY).

Розрахунок цільового ШІМ за температурою та навантаженням реалізовано у методі calculate_target_pwm за кусково-лінійною кривою охолодження з компенсацією зниженої ефективності радіатора. Фрагмент методу наведено у лістингу 4.4.

Лістинг 4.4 – Розрахунок цільового ШІМ

```
def calculate_target_pwm(self, temp: float, load: float) -> int:
    if   temp < 30: target = 20
    elif temp < 50: target = 20 + (temp - 30) * 1.0    # 20→40%
    elif temp < 70: target = 40 + (temp - 50) * 1.5    # 40→70%
    elif temp < 85: target = 70 + (temp - 70) * 2.0    # 70→100%
    else:           target = 100

    if load > 80: target = max(target, 50)   # feed-forward
    elif load > 50: target = max(target, 40)

    efficiency =
environmental_control_algo.get_cooling_efficiency_modifier(...)
    if 0.1 < efficiency < 0.99:
        target = target / efficiency          # компенсація
деградації

    return int(max(config.MIN_FAN_PWM, min(target,
config.MAX_FAN_PWM)))
```

Feed-forward компонент за навантаженням реалізує принцип випередження: ще до того, як температура перевищила поріг, алгоритм збільшує мінімальний ШІМ на підставі знання про поточне обчислювальне навантаження GPU. Компенсаційний множник target / efficiency відображає зворотний зв'язок між накопиченим CI і розрахованим ШІМ: при деградації ефективності охолодження через корозійний наліт алгоритм компенсаційно збільшує оберти вентиляторів.

Логіка застосування гістерезису реалізована у методі calculate_fan_commands. При зростанні цільового ШІМ вище поточного – реакція миттєва. При зниженні – ШІМ утримується не менше MIN_PWM_HOLD_TIME =

60 секунд і знижується кроками не більше 10% за цикл. Це запобігає інтенсивному зносу двигунів вентиляторів від частих перемикань, що описано у підрозділі 2.5.

Клас AdvancedTrendCalculator у модулі trend_calculator.py реалізує математичні моделі CI і FWI, визначені у підрозділі 2.5.4. Обидва індекси є кумулятивними: вони накопичуються протягом усього часу роботи системи і скидаються лише при зміні профілю або ручному скиданні через адміністративний ендпоінт.

Розрахунок приросту CI за формулою Арреніуса(див. формулу 2.4) та його запис наведено у лістингу 4.5.

Лістинг 4.5 – Розрахунок приросту індексу корозії CI

```
def update_ci(self, humidity: float, dust: float, temperature:
float):
    dt = (time.time() - self.last_update) / 3600.0  # у годинах

    dust_factor = 1 + 0.1 * dust
    # Прихований фактор: різкі зміни вологості пришвидшують осідання
пилу
    if abs(humidity - self.last_humidity) / dt > 1.0:
        dust_factor *= 1.2

    temp_factor = math.exp((temperature - 25) / 10)   # рівняння
Арреніуса
    delta_ci    = (humidity / 50) * dust_factor * temp_factor * dt

    self.ci += delta_ci
    # Пороги: CI < 1.0 — низький ризик; 1.0–2.0 — середній; > 2.0 —
високий
```

Розрахунок приросту FWI наведено у лістингу 4.6.

Лістинг 4.6 – Розрахунок приросту індексу зносу вентиляторів FWI

```
def update_fwi(self, avg_rpm: float, dust: float, avg_temp: float):
    dt = (time.time() - self.last_update) / 3600.0

    rpm_factor  = avg_rpm / self.MAX_RPM              # MAX_RPM =
5000
    dust_factor = 1 + 0.2 * (dust / 50)
    if (dust - self.last_dust) / dt > 2.0:            # різке
зростання пилу
        dust_factor *= 1.15
```

```
    temp_factor = 1.5 if avg_temp > 70 else 1.0        # деградація
мастила

    delta_fwi = rpm_factor * dust_factor * temp_factor * dt
    self.fwi += delta_fwi
    # Пороги: FWI < 100 — норма; 100-200 — підвищений; > 200 —
критичний
```

Метод get_cooling_efficiency_modifier повертає мультиплікатор ефективності охолодження залежно від накопиченого CI. При CI <= 1.5 модифікатор дорівнює 1.0 (деградація відсутня). Починаючи з CI = 1.5, ефективність лінійно знижується до максимального штрафу 4% при CI = 3.0 – що відповідає функції зворотного зв'язку(див. формулу 2.4). Клас EnvironmentalControlAlgorithm у модулі environmental_control.py реалізує гістерезисне керування осушувачем і зволожувачем відповідно до математичної моделі динаміки вологості (див. формулу 2.3). Нормативний діапазон вологості – 40–60%. При виході за межі діапазону алгоритм активує відповідний актюатор на рівні потужності 75%. Ключовою особливістю є інерційна затримка: між послідовними активаціями одного актюатора витримується мінімум MIN_HOLD_TIME = 600 секунд (10 хвилин), що запобігає швидкому зносу реле при коливаннях вологості біля граничного значення. При концентрації пилу понад DUST_ALERT_THRESHOLD = 50 μg/m³ алгоритм формує сповіщення критичного рівня – єдиний сценарій, коли система не може вирішити проблему автоматично і вимагає фізичного втручання інженера.

Сервер упакований у Docker-контейнер. Dockerfile наведено у лістингу 4.7.

Лістинг 4.7 – Dockerfile серверу туманних обчислень

```
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8001
CMD ["python", "app/main.py"]
```

Для запуску усього стеку – серверу туманних обчислень, InfluxDB і веб-інтерфейсу – використовується Docker Compose, визначений у кореневій директорії проекту. Запуск стеку виконується командами docker-compose up --build. Після запуску сервер приймає HTTP-запити на порту 8001.

Процес обробки вхідного пакету телеметрії є головним циклом туманного сервера, його схему занесемо до додатку А.1.

Алгоритм починається з отримання HTTP POST-запиту. Першим кроком відбувається десеріалізація JSON у об'єкт TelemetryPayload із суворою валідацією типів через Pydantic. Якщо валідація не проходить, сервер негайно повертає помилку 422. У разі успіху викликається метод запису всіх отриманих даних (температури, оберти, мікроклімат) до InfluxDB. Далі алгоритм перевіряє температури GPU: якщо показник перевищує критичний поріг (120°C) або поріг попередження (90°C), формується та зберігається відповідне сповіщення.

Після перевірки порогів алгоритм визначає поточний режим роботи системи. У ручному режимі (MANUAL) сервер бере команди, заздалегідь встановлені оператором через веб-інтерфейс. В автоматичному режимі (AUTO) запускається SmartCoolingAlgorithm, який розраховує нові значення ШІМ для вентиляторів на основі температурної історії, навантаження та деградації радіаторів, застосовуючи гістерезис для плавного зниження обертів. Розраховані команди зберігаються у буфер pending_commands для подальшої відправки на апаратний шлюз. На останньому етапі розраховуються середні значення температур і обертів по всьому кластеру, які передаються у AdvancedTrendCalculator для обчислення накопичувальних індексів зносу (CI та FWI). Оновлені індекси також записуються до InfluxDB, після чого алгоритм завершується поверненням HTTP статусу 200.

Алгоритм починає ітерацію по кожному графічному процесору, отриманому в пакеті телеметрії. Для кожного GPU спочатку оновлюється локальний буфер історії температур (зберігаються останні 3 відліки). Порівнюючи останній і передостанній відліки, алгоритм визначає теплове середовище: якщо температура зросла більше ніж на 0.5°C – фіксується стан нагріву (HEATING); якщо впала – стан охолодження (COOLING); інакше – стан стабільності (STEADY).

Далі викликається метод розрахунку цільового ШІМ за кусково-лінійною кривою, де значення ШІМ зростає від 20% при температурі нижче 30°C до 100% при перевищенні 85°C. На цей розрахунок накладається механізм feed-forward: якщо навантаження (Load) перевищує 80%, мінімальний ШІМ примусово встановлюється на рівні 50%, що дозволяє розкрутити вентилятори ще до фактичного перегріву кристала. Отримане цільове значення додатково коригується (збільшується) на основі модифікатора ефективності охолодження, якщо на радіаторах присутній корозійний наліт (CI > 1.5).

Заключний етап алгоритму – застосування гістерезису. Якщо розрахований цільовий ШІМ більший за поточний (реакція на нагрів), нове значення застосовується миттєво для забезпечення безпеки. Якщо ж цільовий ШІМ менший (реакція на охолодження), алгоритм перевіряє час, що минув з останньої зміни ШІМ. Якщо минуло менше 60 секунд, поточні оберти утримуються незмінними. Якщо минуло понад 60 секунд, ШІМ знижується плавно, з максимальним кроком не більше 10% за цикл. Це нівелює ефект «теплових гойдалок» і захищає підшипники вентиляторів від зносу через часті перемикання. Розраховане значення зберігається та додається до пакету керуючих команд.

Наведемо в додатку А.2 деталізовану схему алгоритму, що реалізує адаптивне терморегулювання. Діаграма розміщення для серверу туманних обчислень наведемо у додатку Б.1.

4.2 Розробка тестового стенду

Тестовий стенд апаратного рівня виконує функцію цифрового двійника для фізичної частини IoT-системи. Він дозволяє симулювати роботу датчиків мікроклімату, температурні процеси нагріву кристалів GPU під навантаженням та виконання команд широтно-імпульсної модуляції для вентиляторів без ризику пошкодження реального обладнання.

Проектована гібридна IoT-система призначена для експлуатації в неспеціалізованих умовах (офісні або житлові приміщення), де забезпечення суворого апаратного контролю мікроклімату є неможливим. На відміну від промислових ЦОД, параметри середовища у таких приміщеннях зазнають суттєвих сезонних та добових коливань (наприклад, критичне зниження вологості взимку через опалення, або підвищення температури влітку через недостатню вентиляцію або зміну погоди). Окрім того, локальний обчислювальний кластер малого бізнесу є мобільним об'єктом інфраструктури і може бути переміщений у приміщення з принципово іншими термодинамічними характеристиками.

Враховуючи високу вартість GPU, проведення повноцінних випробувань керуючих алгоритмів та тестування граничних навантажень безпосередньо на реальному апаратному забезпеченні є фінансово неприйнятним. Будь-яка алгоритмічна помилка або збій на етапі налагодження може призвести до термічного пошкодження високовартісних обчислювальних компонентів. З метою нівелювання цих експлуатаційних та фінансових ризиків, у межах проєкту прийнято рішення перенести процес тестування у симуляційне середовище шляхом розробки тестового стенда за концепцією цифрового двійника (Digital Twin) та апаратно-програмного моделювання (Hardware-in-the-Loop).

Архітектура розробленого тестового стенда дозволяє проводити випробування у кількох режимах:

– повна програмна симуляція: абсолютно всі показники (температура, вологість, пил, завантаження GPU) та реакції виконавчих механізмів генеруються виключно математичними моделями;

– гібридне тестування (HIL): до системи підключається реальний Edge-шлюз із фізичними мікроконтролерами та сенсорами, проте зміна параметрів середовища симулюється програмно; або навпаки – використовується реальний GPU-кластер, але сенсорні показники підміняються для штучної імітації аварійних кліматичних ситуацій.

Такий підхід дозволяє перенести весь процес налагодження алгоритмів туманного сервера у безпечне та контрольоване середовище. Для забезпечення

адекватності такого тестування та функціонування аналітичного ядра необхідне точне математичне моделювання фізичних процесів. Цей підрозділ формалізує базові математичні моделі теплової динаміки, накопичення забруднюючих речовин (пилу), зміни вологості та процесів апаратної деградації компонентів (корозія та знос вентиляторів), які закладені в основу розробленого тестового стенда.

Головним компонентом стенду є файл emulator.py, який оркеструє всіма симуляторами. Для взаємодії з Fog-сервером розроблено модуль api_client.py, який реалізує HTTP-клієнт, що повністю імітує мережеву поведінку мікроконтролера ESP32. Моделі даних, якими оперує емулятор, описані у models.py і повністю синхронізовані з Pydantic-контрактами на стороні сервера. Наведена схема асоціацій компонентів тестового стенду на рисунку 4.3.

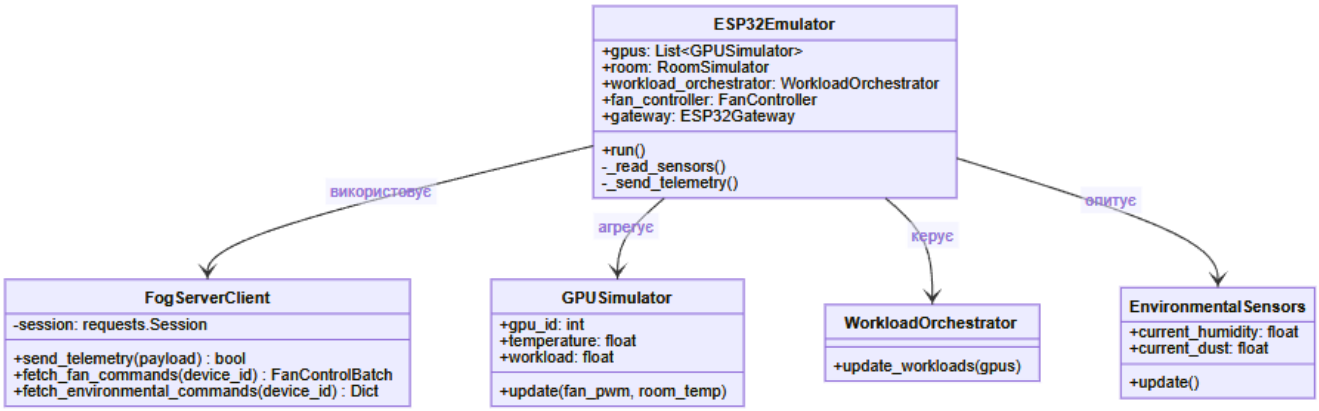


Рисунок 4.3 – Схема асоціацій компонентів тестового стенду

Замість фізичного мікроконтролера ESP32, який зчитує аналогові та цифрові сигнали по шинах I2C або UART, розроблено клас ESP32Gateway. Він збирає дані з програмних симуляторів сенсорів, які реалізовані з використанням динамічних профілів середовища. Це дозволяє симулювати плавне підвищення вологості чи накопичення пилу, спираючись на математичні моделі, описані у розділі 2.5.

Для генерації реалістичних умов нагріву використовується GPUSimulator у зв'язці з WorkloadOrchestrator. Замість сталого навантаження стенд підтримує симуляцію дата-центру. Оркестратор імітує запуск важких завдань, різко

підвищуючи навантаження до 95-100% на кілька хвилин, після чого скидає його до базового рівня (idle). GPUSimulator перераховує температуру кристала за законом Ньютона, враховуючи поточний ШІМ вентилятора та температуру навколишнього середовища, створюючи реалістичну інерцію нагріву та охолодження.

Додатково система підтримує моделювання деградації охолодження. При накопиченні пилу віртуальний коефіцієнт тепловіддачі поступово зменшується, що призводить до повільнішого відведення тепла та підвищення робочих температур навіть при незмінному навантаженні. Це дозволяє тестувати алгоритми раннього виявлення забруднення системи охолодження, механізми автоматичного підвищення швидкості вентиляторів та логіку генерації попереджень про необхідність технічного обслуговування.

Алгоритм симуляції параметрів навколишніх умов та роботи компонентів апаратного рівня занесемо до додатку А.3.

Комунікація з туманним сервером реалізована за протоколом HTTP. Модуль api_client.py ініціює з'єднання, використовуючи пул з'єднань для оптимізації TCP-хендшейків та зменшення накладних витрат на встановлення мережевого підключення. Діаграма розміщення для тестового стенду наведена у додатку Б.2.

Взаємодія імітує мережеву поведінку реального мікроконтролера ESP32. Оскільки шлюз фізично не може приймати вхідні HTTP-з'єднання через обмеження NAT локальної мережі, емулятор реалізує шаблон Polling (циклічне опитування). Стенд виступає виключно як HTTP-клієнт і періодично звертається до двох основних ендпоінтів сервера:

– POST /api/v1/telemetry – відправка зібраного об'єкта TelemetryPayload із даними сенсорів і GPU. У разі помилки валідації на боці сервера (наприклад, відсутність обов'язкового поля), клієнт перехоплює помилку 422 Unprocessable Entity та записує її у локальний лог;

– GET /api/v1/fan-control/{device_id} – запит команд керування. Якщо сервер розрахував нові команди (HTTP 200 OK), клієнт десеріалізує отриманий FanControlBatch і застосовує нові значення ШІМ до віртуального FanController.

Якщо нових команд немає, сервер повертає HTTP 204 No Content, і клієнт утримує поточні значення ШІМ, заощаджуючи обчислювальні ресурси.

4.3 Розробка веб-системи та інтерфейсу користувача

Для розробки вебсистеми обрано багаторівневу клієнт-серверну архітектуру з розділенням відповідальності між компонентами. Архітектура веб-застосунку організована за принципом розділення відповідальності. Кореневий рівень містить конфігураційні файли Next.js, TypeScript, Tailwind CSS та package.json з переліком залежностей. Папка app містить сторінки застосунку з використанням нової структури файлів: page.tsx для головної сторінки, page.tsx у папці history для графіків, page.tsx у папці control для ручого управління. Кожна сторінка є компонентом реакт, що оброблюється на сервері для швидкого початкового завантаження.

Серверна частина реалізована у вигляді fog-сервера на базі FastAPI. Він виступає центральною точкою обробки даних і виконує роль API для всіх зовнішніх клієнтів. Fog-сервер відповідає за прийом даних, та збереження, виконання алгоритмів автоматичного керування охолодженням, управління режимами роботи системи та формування повідомлень. Також сервер надає API для веб-інтерфейсу, забезпечуючи доступ до поточного стану та історії даних.

Папка components містить переиспользовані компоненти інтерфейсу: GPUCard.tsx для відображення стану одного GPU, FanCard.tsx для відображення стану вентилятора, TempChart.tsx для графіка температур, FanControlSlider.tsx для слайдера управління ШІМ тощо. Компоненти побудовані за принципом максимальної переиспользовуємості.

Папка lib містить модулі та бізнес-логіку. Файл api.ts інкапсулює всі функції для взаємодії з fog-сервером: getCurrentState(), getFanStatistics(), setSystemMode(), setManualFanControl(). Кожна функція повертає типізовані дані через TypeScript

інтерфейси, що забезпечує типобезпеку. Файл utils.ts містить допоміжні функції для форматування дат, конвертації одиниць вимірювання, обчислення статистики тощо.

Клас APIClient виступає центральним для всієї HTTP-комунікації з fog-сервером. Він інкапсулює деталі формування запитів, обробки відповідей та помилок, надаючи методи з типізованими результатами. Метод getCurrentState повертає результат, що передається у об'єкт CurrentState з масивом температур GPU та активних повідомлень. Метод getFanStatistics запитує статистику роботи вентиляторів за останню годину включаючи середні значення ШІМ та час на максимальних обертах. Методи setSystemMode та setManualFanControl виконують POST-запити для зміни конфігурації системи з валідацією на серверній стороні.

Моделі даних CurrentState, GPUTemp, Alert, FanStatistics, SystemMode, ManualCommand представляють TypeScript-інтерфейси що відповідають JSON-схемам fog-сервера. Використання інтерфейсів замість класів обумовлено тим що TypeScript-інтерфейси компілюються у нічого у runtime і додані для перевірки типів. Композиційні зв'язки показані через відношення "contains": CurrentState містить масив GPUTemp та масив Alert, що відображає вкладеність JSON-структури.

React-компоненти DashboardPage, ControlPage, HistoryPage є page-компонентами що відповідають маршрутам Next.js та містять логіку завантаження даних через функцію useEffect, управління локальним станом через функцію useState, обробку подій користувача. DashboardPage використовує APIClient для періодичного опитування поточного стану кожні 5 секунд та використовує дочірні компоненти GPUCard і FanCard для кожного GPU та вентилятора. ControlPage обробляє перемикання режимів та застосування ручних команд з перевіркою небезпечних налаштувань. HistoryPage завантажує дані історії та відображає графіки через бібліотеку Recharts.

Реалізація вебсистеми моніторингу організована як набір незалежних компонентів, що взаємодіють через чітко визначені інтерфейси. Кожен компонент відповідає за конкретну частину функціональності та може розроблятися,

тестуватися та модифікуватися незалежно від інших. Такий модульний підхід спрощує розробку, підвищує переиспользовуваність коду та полегшує підтримку проекту.

Структура головної сторінки організована вертикально з чіткою ієрархією інформації. На початку сторінки розташований заголовок з назвою системи та коротким описом. Нижче розташований стан системи, поточний режим роботи (автоматичний або ручний) та час останнього оновлення даних, зобразимо макет головної сторінки на рисунку 4.4.

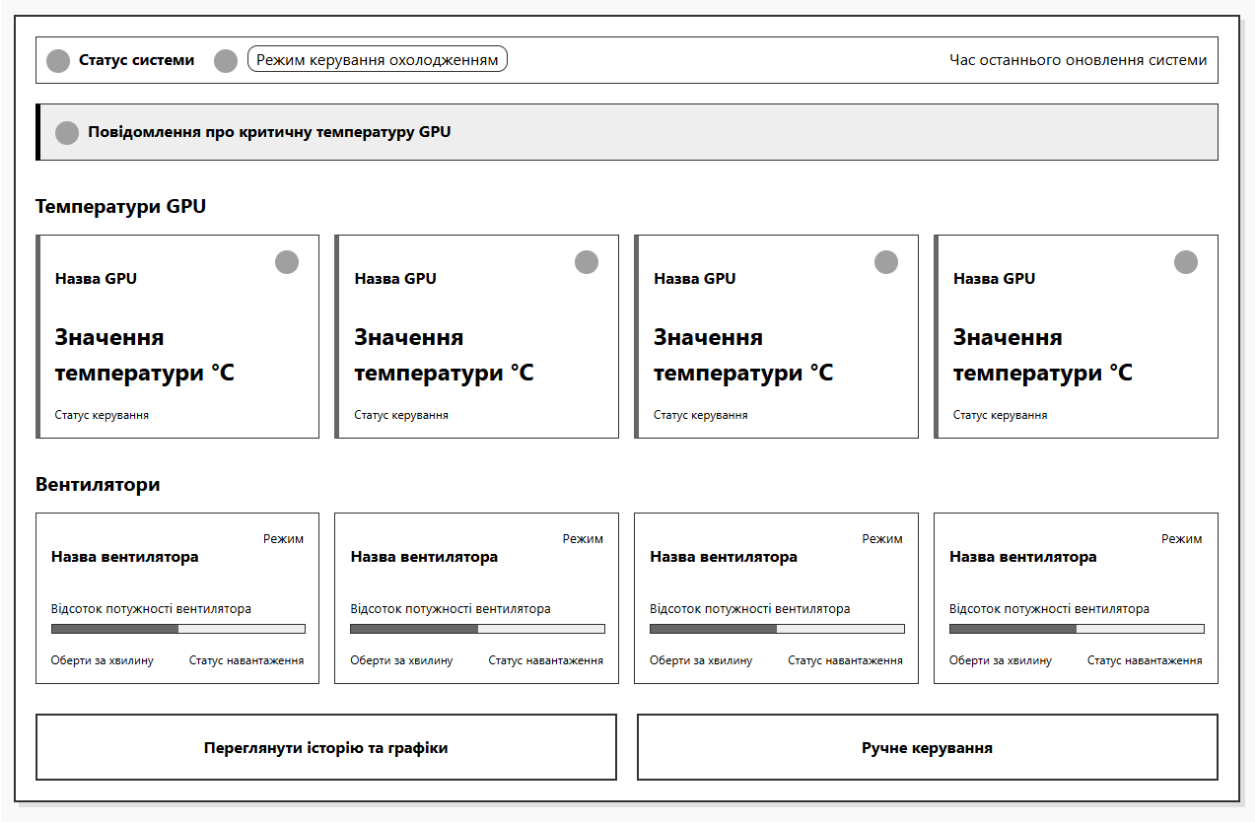


Рисунок 4.4 – Концептуальна модель головної сторінки

Після цього блоку розташована система повідомлень про критичний стан систем. Повідомлення реалізовані як блочні кольорові елементи з (червона для критичних, жовта для попереджень). Кожен критичне повідомлення містить інформацію про номер GPU та температуру GPU.

Температури усіх GPU відображаються у вигляді рядів інформаційних блоків, де кожен блок представляє інформацію про один GPU. Блоки організовані в адаптивну сітку, що автоматично підлаштовується під розмір екрану. Кожна картка містить ідентифікатор GPU, поточну температуру великим шрифтом та інформацію про режим керування вентилятором.

Останнім блоком головної сторінки будуть кнопки навігації до сторінок графіків температур GPU та ручного керування вентиляторами.

Реалізація інформаційних блоків GPU здійснюється через передачу масиву інформації температур з стилізацією на основі значень температур.

На рисунку 4.5 ми відобразимо ескіз для сторінки з графіками температур та потужності вентиляторів, на сторінці знаходяться кнопки-перемикачі між вибором інтервалів для відображення даних для графіків, графік температур, де кожна окрема GPU відображена іншим кольором, графік потужності вентиляторів з двома режимами відображення: у обертах на хвилину та у процентах від максимальної потужності.

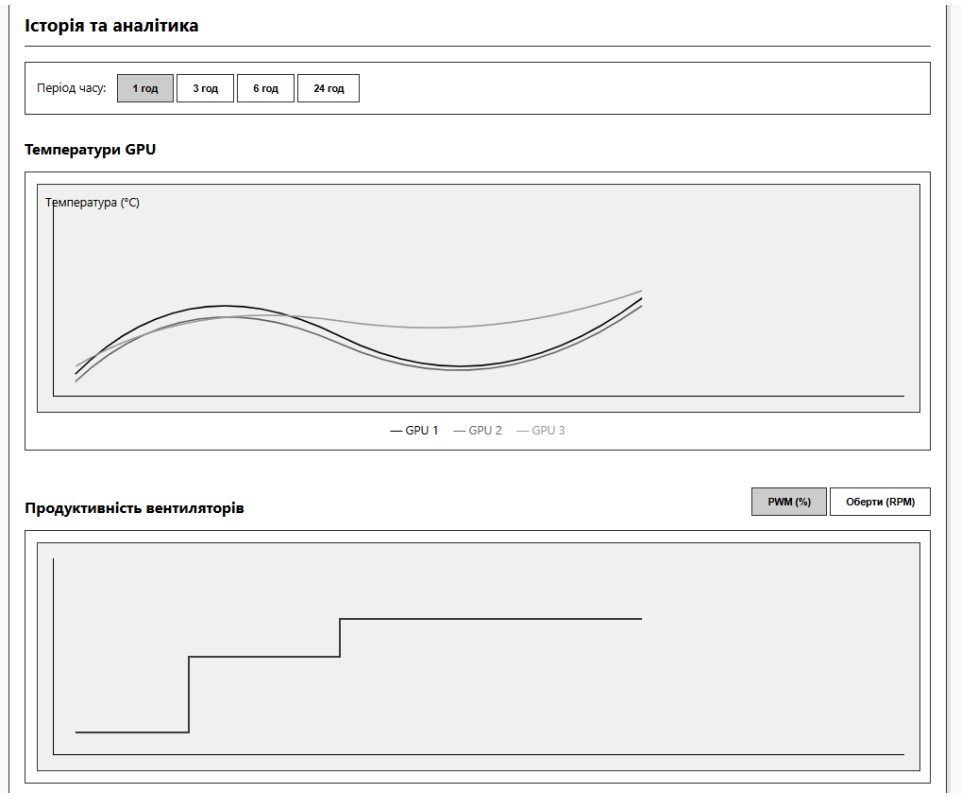


Рисунок 4.5 – Концептуальна модель сторінки графіків

Для візуалізації даних у вигляді графіків розглядалися бібліотеки Chart.js, D3.js та Recharts. Chart.js є простою у використанні бібліотекою, що підтримує основні типи графіків: лінійні, bar, pie тощо. Однак Chart.js працює через Canvas API, що ускладнює інтеграцію з React та кастомізацію елементів графіка. D3.js є найпотужнішою бібліотекою для створення складних інтерактивних візуалізацій, але вимагає глибокого розуміння SVG, масштабування, осей та інших низькорівневих деталей, що значно збільшує час розробки навіть простих графіків.

Recharts обрано як компроміс між простотою та гнучкістю. Recharts побудована специфічно для React та використовує декларативний підхід: графік описується через компоненти. Recharts автоматично обробляє більшість рутинних завдань: масштабування осей, відображення підказок при наведенні, легенд, але водночас дозволяє кастомізувати кожен елемент через пропси. Для проекту моніторингу, де потрібні лінійні графіки температур та ступінчасті графіки.

Дані для графіків отримуються з API fog-сервера у вигляді масиву точок даних з бази даних InfluxDB, де кожна точка містить час, тип вимірювання (температура GPU, температура приміщення, оберти вентилятора), ідентифікатор GPU або вентилятора та значення. Для відображення на графіках ці дані перетворюються у структуру, сумісну з форматом Recharts.

Перетворення даних здійснюється через функцію групування, що перетворює масив точок даних у об'єкт, де ключем є час (відформатований у формат "година:хвилина"), а значенням об'єкт з температурами всіх GPU та температурою приміщення. Recharts створює графік, де кожна точка на осі X відповідає моменту часу, а на осі Y значенням температур для різних GPU.

Групування даних виконується через метод reduce масиву JavaScript, щоб ефективно обробити великі обсяги даних. Для кожного GPU створюється окрема лінія на графіку з унікальним кольором. Температура приміщення відображається пунктирною чорною лінією.

Графік температур GPU реалізований як лінійний графік з використанням компонента LineChart з бібліотеки Recharts. Графік відображає температури всіх восьми GPU та температуру приміщення на одному графіку.

Графік включає інтерактивні елементи: сітку, підказки при наведенні курсора на точки, легенду для ідентифікації ліній.

Графік роботи вентиляторів реалізує перемикання між двома режимами відображення: PWM (потужність у відсотках) та RPM (оберти на хвилину). Перемикання здійснюється через кнопки в інтерфейсі, що змінюють стан компонента та відображають відповідний графік.

Графік PWM використовує ступінчастий тип лінії type="stepAfter"`, це відображає дискретні зміни керуючих команд. Графік RPM також використовує ступінчасту лінію, що відображає фактичні оберти вентиляторів у відповідь на зміни PWM.

Діапазон осі Y для PWM становить від 0% до 100%, для RPM – від 500 до 5500 обертів на хвилину.

Сторінка історії реалізує вибір періоду відображення даних через кнопки: 1 година, 3 години, 6 годин, 24 години. При зміні періоду автоматично виконується новий запит до API з оновленням даних та перебудовою графіків.

Реалізація вибору періоду здійснюється через React state, що зберігає поточне значення періоду. При зміні значення спрацьовує useEffect функція, що виконує новий запит до API з оновленим параметром hours. Отримані дані обробляються та передаються для відображення на графіках.

Стан вентиляторів відображається через інтеграцію компонента FanCard , що інкапсулює логіку відображення детальної інформації про кожен вентилятор.

Компонент FanCard відповідає за візуалізацію стану вентилятора, включаючи поточні значення обертів за хвилину та RPM, кольорове відображення потужності вентилятора, статистику за останню годину та інформацію про температуру GPU за який він відповідає.

При перемиканні на автоматичний режим система відразу застосовує зміну через API запит, що змінює глобальний режим роботи на fog-сервері. Після

успішного перемикання відображається інформаційне повідомлення з підтвердженням зміни режиму. При перемиканні на ручний режим відображається попередження у модальному вікні.

На ескізі сторінки ручного керування вентиляторами який ми занесемо до рисунку 4.6, ми розмістимо 3 режими керування як шаблонні: тихий, сбалансований та турбо. Нижче будуть розміщуватись ідентифікатори кожного вентилятора та повзунки які регулюють його потужність.

Після того як користувач налаштує потужність, він має можливість підтвердити свої дії та передати команду на сервер або скасувати усі налаштування через кнопки керування.

Усі дії користувача, такі як переходи між режимами керувань, шаблонами потужності вентиляторів та їх ручного керування будуть записані до журналу дій.

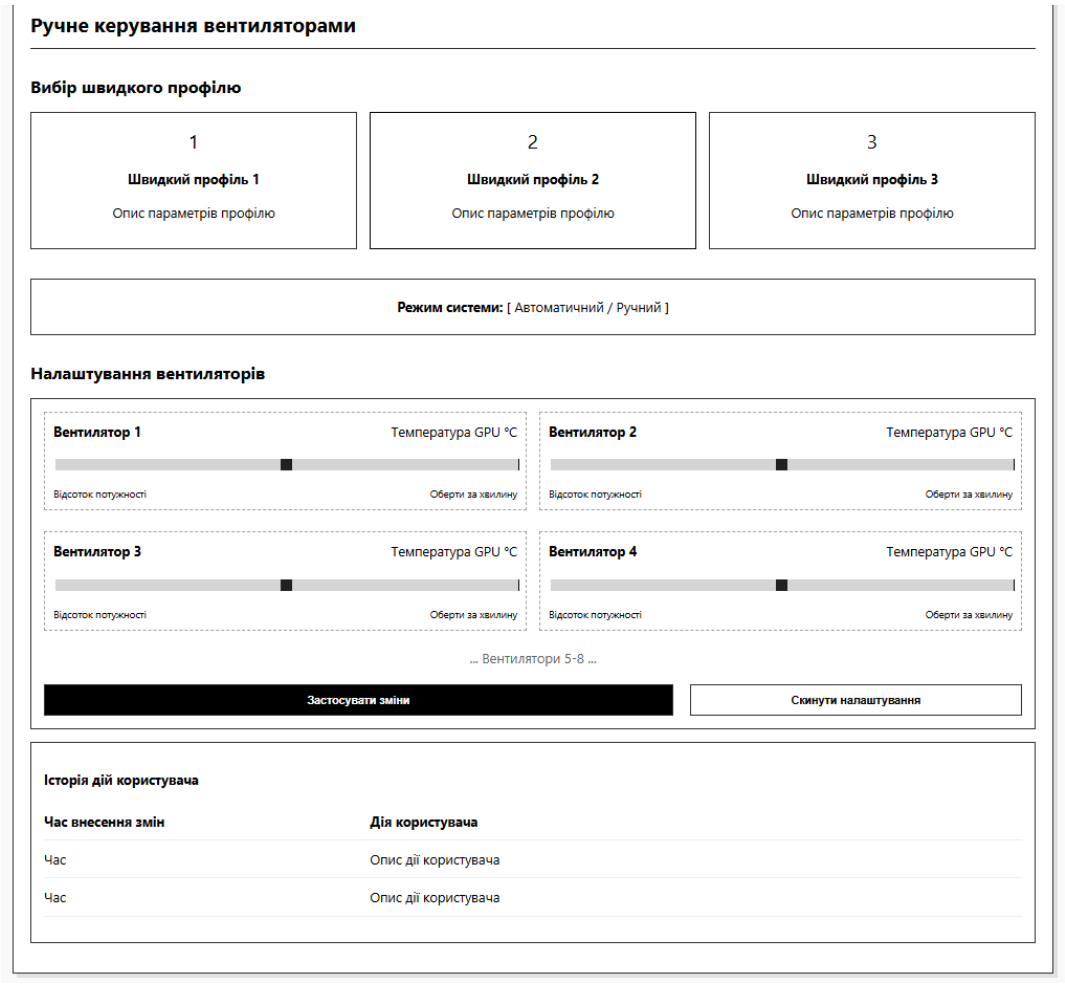


Рисунок 4.6 – Концептуальна модель сторінки ручного керування вентиляторами

Реалізація перемикання режимів здійснюється через асинхронну функцію handleModeSwitch, що виконує POST-запит до API /api/v1/system-mode з новим режимом. Після успішного запиту локальний стан компонента оновлюється, що призводить до перерендеру інтерфейсу з оновленою індикацією режиму. У випадку помилки користувачу показується повідомлення про помилку з описом проблеми.

Для налаштування кожного вентилятора окремо створена система слайдерів, для встановлення PWM у діапазоні від 0% до 100%. Кожен слайдер розташований у власному блоці з інформацією про вентилятор, поточну температуру відповідного GPU, поточне значення PWM та значення RPM.

5 РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ

5.1 Тестування моніторингу та обробки даних телеметрії

Першим етапом тестування стане запуск серверу туманних обчислень, який під'єднається до бази даних та помодульно запустить усі алгоритми керування для передачі команд на пристрою керування охолодженням для збору телеметрії від сенсорів та окремі алгоритми для розрахунку трендів. Консольний вивід повідомлень запуску серверу занесемо до рисунку 5.1.

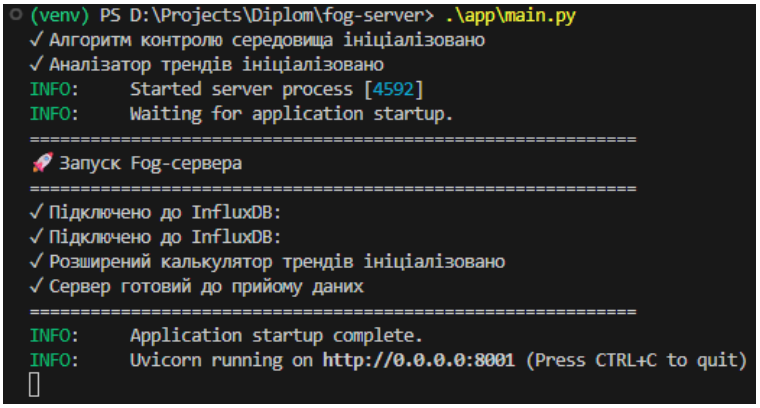


Рисунок 5.1 – Запуск Fog-сервера

Другим кроком ми запустимо тестовий стенд, який буде симулювати як апаратну частину у вигляді сенсорів, які начебто збирають інформацію, та мікроконтролера у вигляді граничного шлюзу вираженому обмеженнями та особливостями esp32, так і параметри навколишнього середовища у вигляді концентрації дисперсних частинок, вологості та температури як приміщення, так і кожної окремої відеокарти у кластері. Консольний вивід з ініціалізацією симуляції середовища та GPU-кластеру занесемо до рисунку 5.2.

```
(venv) PS D:\Projects\Diplom\hardware-emulator> .\main.py
[2026-05-18 00:06:34] INFO: Ініціалізація WorkloadOrchestrator...
[2026-05-18 00:06:34] INFO:   ✓ ML профілі активовано (режим датацентру)
[2026-05-18 00:06:34] INFO: Ініціалізація 16 вентиляторів...
[2026-05-18 00:06:34] INFO: Ініціалізація ESP32 Edge Gateway...
[2026-05-18 00:06:34] INFO:   Fog-сервер:
[2026-05-18 00:06:34] INFO: Ініціалізація сенсорів середовища...
[2026-05-18 00:06:34] INFO:   ✓ Сенсор вологості: 50.0%
[2026-05-18 00:06:34] INFO:   ✓ Сенсор пилу: 35.0 µg/m³
[2026-05-18 00:06:34] INFO: Ініціалізація контролера середовища...
[2026-05-18 00:06:34] INFO: Контролер середовища ініціалізовано
[2026-05-18 00:06:34] INFO:   ✓ Приводи середовища готові
[2026-05-18 00:06:34] INFO: ✓ Ініціалізацію завершено
[2026-05-18 00:06:34] INFO: ============================================================
[2026-05-18 00:06:34] INFO: 🚀 Запуск емулятора...
[2026-05-18 00:06:34] INFO:    Інтервал читання датчиків: 5 сек
[2026-05-18 00:06:34] INFO:    Інтервал відправки даних: 30 сек
[2026-05-18 00:06:34] INFO:    Натисніть Ctrl+C для зупинки
[2026-05-18 00:06:34] INFO: ============================================================
```

Рисунок 5.2 – Запуск середовища для тестування

Сервер туманних обчислень отримує дані від тестового стенду, показує зчитані значення температур та навантаженості відеокарт. На основі цих даних прогнозує зміну температури, повідомляючи, буде це нагрів чи охолодження. Консольний вивід який демонструє отримання сервером повідомлення від тестового стенду занесемо до рисунку 5.3.

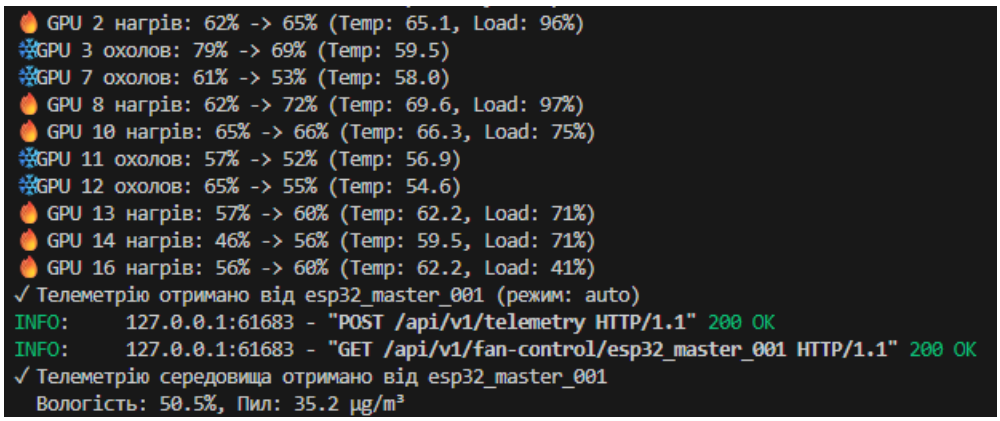


Рисунок 5.3 – Fog-сервер отримує телеметрію від аппаратного рівня

Також fog-сервер передає команди на керуючи пристрої, такі як осушувач та зволожувач, і в залежності від спрогнозованих раніше значень температури відеокарт передає команди на вентилятори, для зменшення деградації компонента на необчислювальні дешевші пристрої.

У свою чергу, оператор або інженер має можливість віддаленого моніторингу за допомогою вебсистеми, перегляду як поточних значень, так і історії даних за останню годину, три, шість або 24 години. Також він може аналізувати тренди – зокрема ризик корозії та запиленості приміщення, що є головним параметром, який вказуватиме на необхідність фізичного втручання, обслуговування та прибирання.

Завдяки ручному керуванню та можливості відстежувати ретроспективні дані – такі як температура відеокарт, навантаженість і швидкість вентиляторів, запиленість, вологість, а також ризики корозії та запиленості приміщень – інженер зможе виявляти аномалії в роботі будь-яких компонентів та оцінювати їхню відмовостійкість.

Консольний вивід з повідомленнями від серверу про передачу команд на керуючи пристрої занесемо до рисунку 5.4.

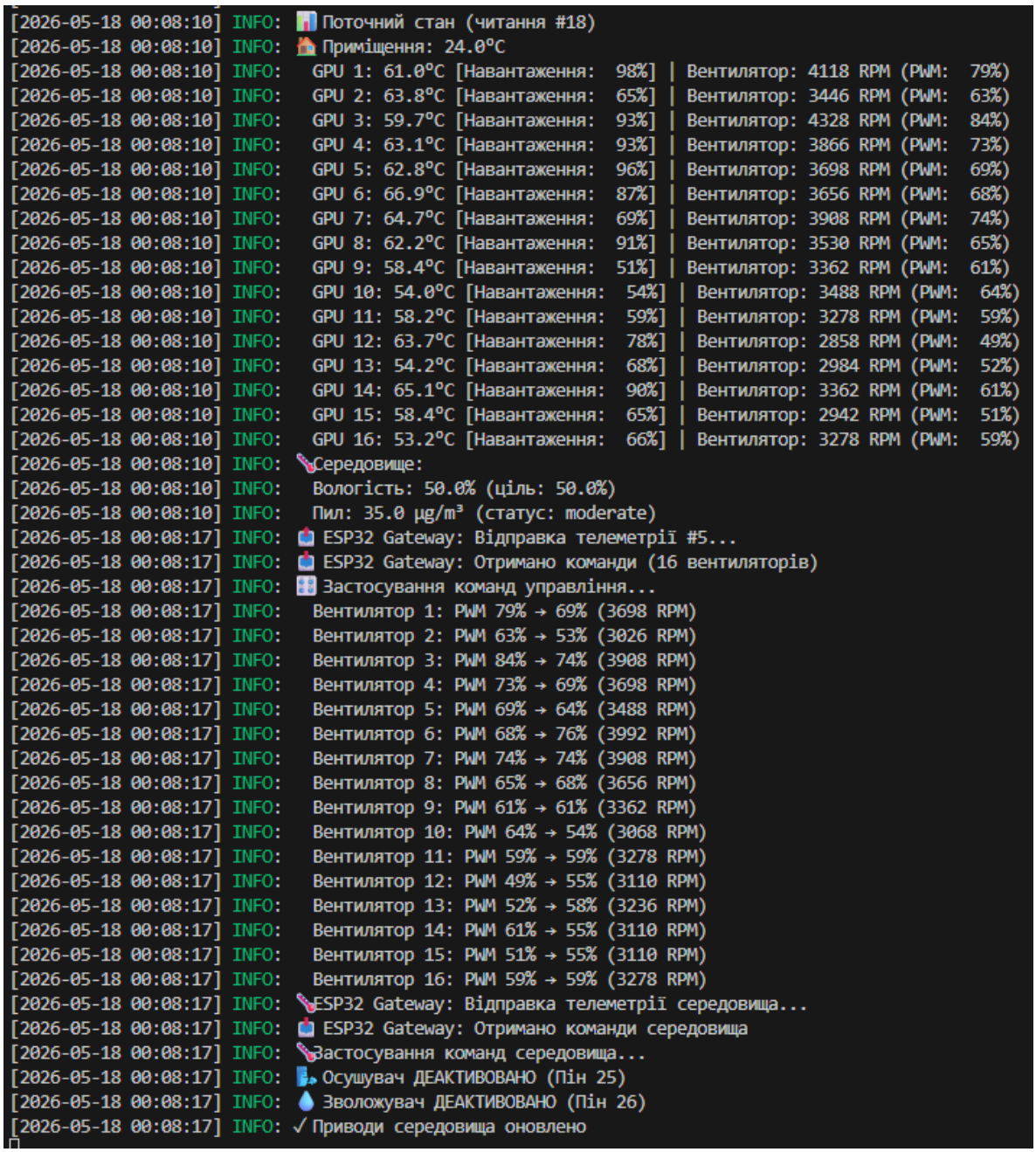


Рисунок 5.4 – Fog-сервер передає команди на керуючі пристрої керування охолодженням

За допомогою адмінпанелі, у якій можна симулювати параметри зовнішнього середовища (запиленість та вологість), можна відстежувати індекс деградації компонентів. Це необхідно для прогнозування часу реагування, коли приміщенню з GPU-кластером знадобиться фізичне обслуговування. Розроблену головну сторінку веб-системи занесемо до рисунку 5.5

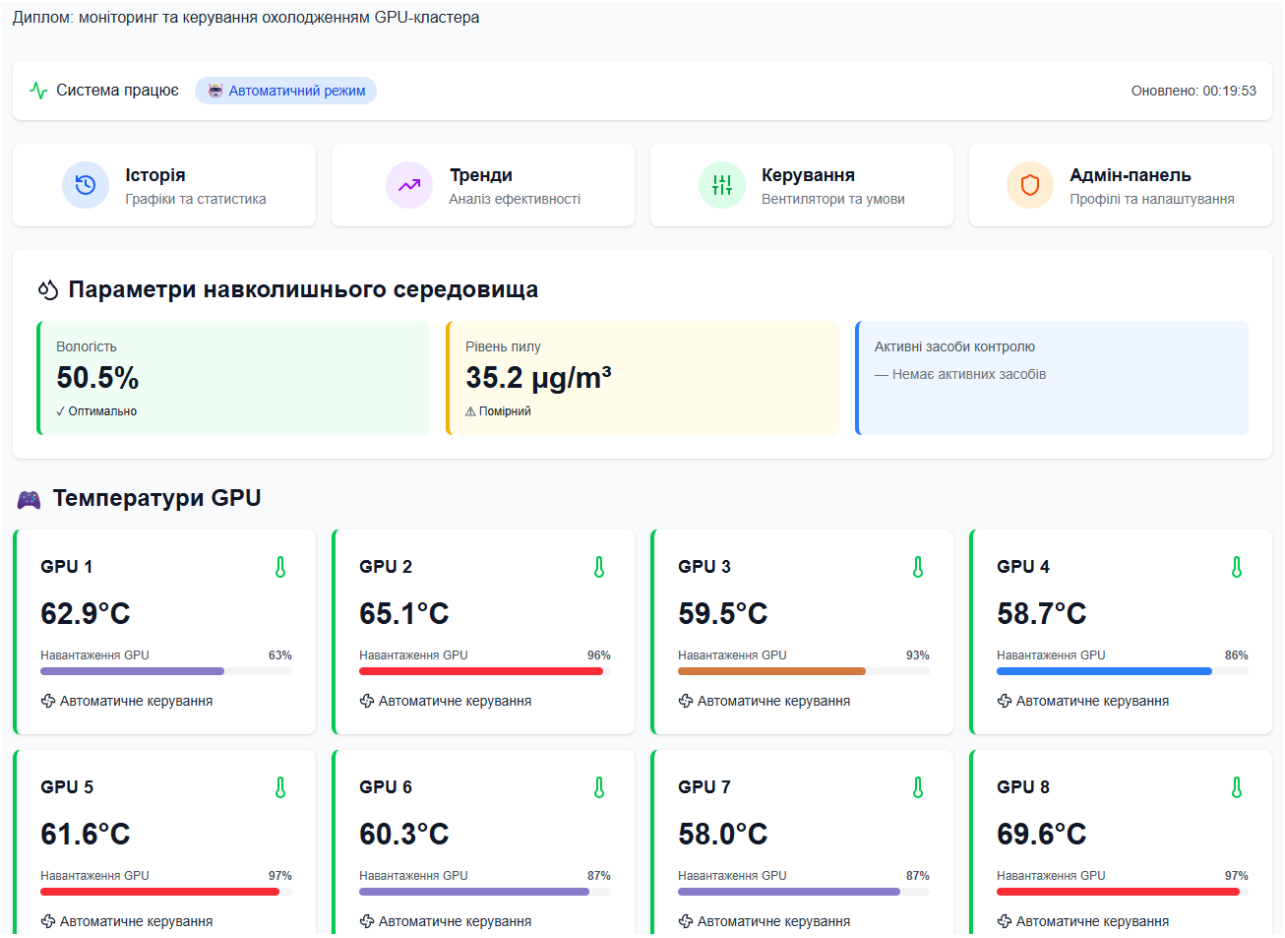


Рисунок 5.5 – Головна сторінка веб-системи

5.2 Можливості подальшого розвитку

Розроблена гібридна туманно-хмарна система моніторингу та управління мікрокліматом GPU-кластерів володіє низкою суттєвих архітектурних переваг та має широкий потенціал для подальшого масштабування. Основні досягнення та вектори розвитку можна розділити за архітектурними рівнями системи. На фізичному рівні було обрано економічно доступні та поширені компоненти. Використання шини 1-Wire для температурних датчиків дозволяє суттєво зекономити флеш-пам'ять та зменшити загальну кількість необхідних

мікроконтролерів, оскільки безліч пристроїв можуть паралельно працювати на одній лінії передачі даних. Окремою перевагою є розробка програмного тестового стенда (емулятора). Він дозволяє симулювати як поведінку апаратного забезпечення, так і складні фізичні параметри навколишнього середовища. Це критично економить фінансові ресурси на етапі прототипування, усуваючи необхідність закупівлі дорогого обладнання до фінального тестування алгоритмів. Крім того, емулятор забезпечує високу гнучкість масштабування: інженер може штучно збільшувати або зменшувати обчислювальне навантаження, змінювати кількість та потужність віртуальних вентиляторів для перевірки стресостійкості системи у будь-яких сценаріях.

Сервер туманних обчислень побудований за принципом розділеної відповідальності на базі стека Python. Такий підхід робить його надзвичайно легковаговим, що суттєво економить обчислювальні ресурси дороговартісного серверного обладнання. Завдяки цій оптимізації, у ролі Fog-вузла може виступати навіть звичайний ноутбук або компактний комп'ютер, де будь-який зекономлений об'єм оперативної пам'яті та процесорного часу є критично важливим. Архітектура сервера забезпечує високу відмовостійкість та має потенціал до нарощування функціоналу, оскільки обчислювальні алгоритми ізольовані і можуть запускатися та оновлюватися окремо. На рівні туманного вузла закладено модульну структуру, що дозволяє у майбутньому легко реалізувати підтримку багатьох додаткових протоколів обміну даними.

У контексті подальшого розвитку планується розширення функціоналу веб-системи таким чином, щоб конфігурацію самого Fog-сервера можна було відлагоджувати та змінювати повністю віддалено. Це мінімізує необхідність фізичного втручання в інфраструктуру туманного рівня та дозволить керувати розподіленими кластерами з єдиної хмарної панелі адміністратора.

ВИСНОВКИ

За результатами виконання дипломної роботи було успішно спроєктовано та реалізовано гібридну IoT-систему моніторингу та адаптивного керування охолодженням GPU-кластеру. Розроблений програмно-апаратний комплекс забезпечує безперервний збір телеметрії, оцінку мікроклімату приміщення та автоматичне керування температурними і екологічними параметрами локального дата-центру.

У ході роботи було проведено детальний аналіз предметної області, обґрунтовано вибір гібридної хмарно-туманної архітектури та відповідного технологічного стеку. Було успішно підібрано апаратні компоненти, розроблено математичні моделі кумулятивного зносу обладнання, а також створено туманне серверне ядро з алгоритмами випереджального керування і сучасний веб-інтерфейс для візуалізації телеметрії. Усі функціональні та нефункціональні вимоги, встановлені на етапі розробки специфікації, були виконані в повному обсязі.

Впровадження даної системи дозволяє інженерному персоналу значно економити свій робочий час завдяки автоматизації рутинного моніторингу, а бізнесу суттєво оптимізувати фінансові витрати. Головним здобутком є реалізація концепції делегування зносу: дороговартісні обчислювальні компоненти (графічні процесори) переносять своє амортизаційне навантаження на значно дешевші допоміжні механізми (вентилятори, системи осушення та зволоження).

Після успішного тестування встановлено, що система стабільно функціонує, демонструє швидку реакцію на термодинамічні зміни та зберігає автономність базових контурів стабілізації навіть при втраті з'єднання з хмарним рівнем.

Таким чином, поставлену мету щодо запобігання прискореній деградації обладнання та забезпечення його стабільної роботи було досягнуто повністю. Розроблена смартсистема характеризується високою гнучкістю та має великий потенціал до подальшого масштабування відповідно до майбутніх потреб та зростання потужностей GPU-кластеру.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Transforming Agriculture with High Performance Computing: Sustainable Smart Farming – An Experimental Study. URL: https://premierscience.com/pjs-25-1214/ (дата звернення 04.04.2026).

2. Що таке Інтернет речей? URL: https://aws.amazon.com/what-is/iot/ (дата звернення 04.04.2026).

3. Safety-first AI for autonomous data centre cooling and industrial control. URL: https://deepmind.google/blog/safety-first-ai-for-autonomous-data-centre-cooling-and-industrial-control/ (дата звернення 05.04.2026).

4. Deploying mission-critical edge applications with AWS IoT Greengrass in disconnected environments. URL: https://aws.amazon.com/ru/blogs/publicsector/deploying-mission-critical-edge-applications-with-aws-iot-greengrass-in-disconnected-environments/ (дата звернення 16.04.2026)

5. Проектування мережі 1-WIRE. URL: https://ist.kpi.ua/wp-content/uploads/2017/05/Навчальний-посібник-1-Wire.pdf (дата звернення 20.04.2026).

6. Чому AI переходить від Cloud до Fog-обчислень. URL: https://dou.ua/lenta/articles/cloud-fog-computing/ (дата звернення 03.04.2026).

7. Рівняння Арреніуса. URL: https://uk.wikipedia.org/wiki/Рівняння_Арреніуса (дата звернення 11.04.2026).

8. JavaScript MDN. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 10.05.2026).

9. React: The library for web and native user interfaces. URL: https://react.dev/ (дата звернення 05.05.2026).

10. Next.js by Vercel – The React Framework. URL: https://nextjs.org/docs (дата звернення 14.05.2026).

11. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення 08.05.2026).

12. InfluxDB OSS v2 Documentation. URL: https://docs.influxdata.com/influxdb/v2/ (дата звернення 04.04.2026).

13. Docker Compose Overview URL: https://docs.docker.com/compose/ (дата звернення 28.04.2026).

14. I2C Communication Protocol. URL: https://learn.sparkfun.com/tutorials/i2c/all (дата звернення 15.04.2026).

15. AI-driven cooling technologies for high-performance data centres: state-of-the-art review and future directions. URL: https://www.sciencedirect.com/science/article/pii/S221313882500342X (дата звернення 23.04.2026).

16. The New AI Architecture: Combining Edge, Fog, and Cloud Computing. URL: https://www.seco.com/news/details/the-new-ai-architecture-combining-edge-fog-and-cloud-computing (дата звернення 01.05.2026).

17. Застосування хмарних технологій в Інтернеті речей. URL: https://ela.kpi.ua/server/api/core/bitstreams/3a1de33d-ee23-4c8b-acee-2f919304d0c9/content (дата звернення 08.05.2026).

18. Cloud vs Edge vs Fog Computing in IoT: How to Choose the Right Architecture. URL: https://metadeskglobal.com/cloud-vs-edge-vs-fog-computing-in-iot/ (дата звернення 28.04.2026).

19. How Data Centers Actually Work: From Cooling Systems to GPU Clusters. URL: https://pub.towardsai.net/how-data-centers-actually-work-from-cooling-systems-to-gpu-clusters-a23918104762 (дата звернення 03.05.2026).

20. Cooling Control Strategies in Data Centers for Energy Efficiency and Heat Recovery. URL: https://www.diva-portal.org/smash/get/diva2:1349556/FULLTEXT01.pdf (дата звернення 08.05.2026).

ДОДАТОК А
СХЕМИ АЛГОРИТМІВ

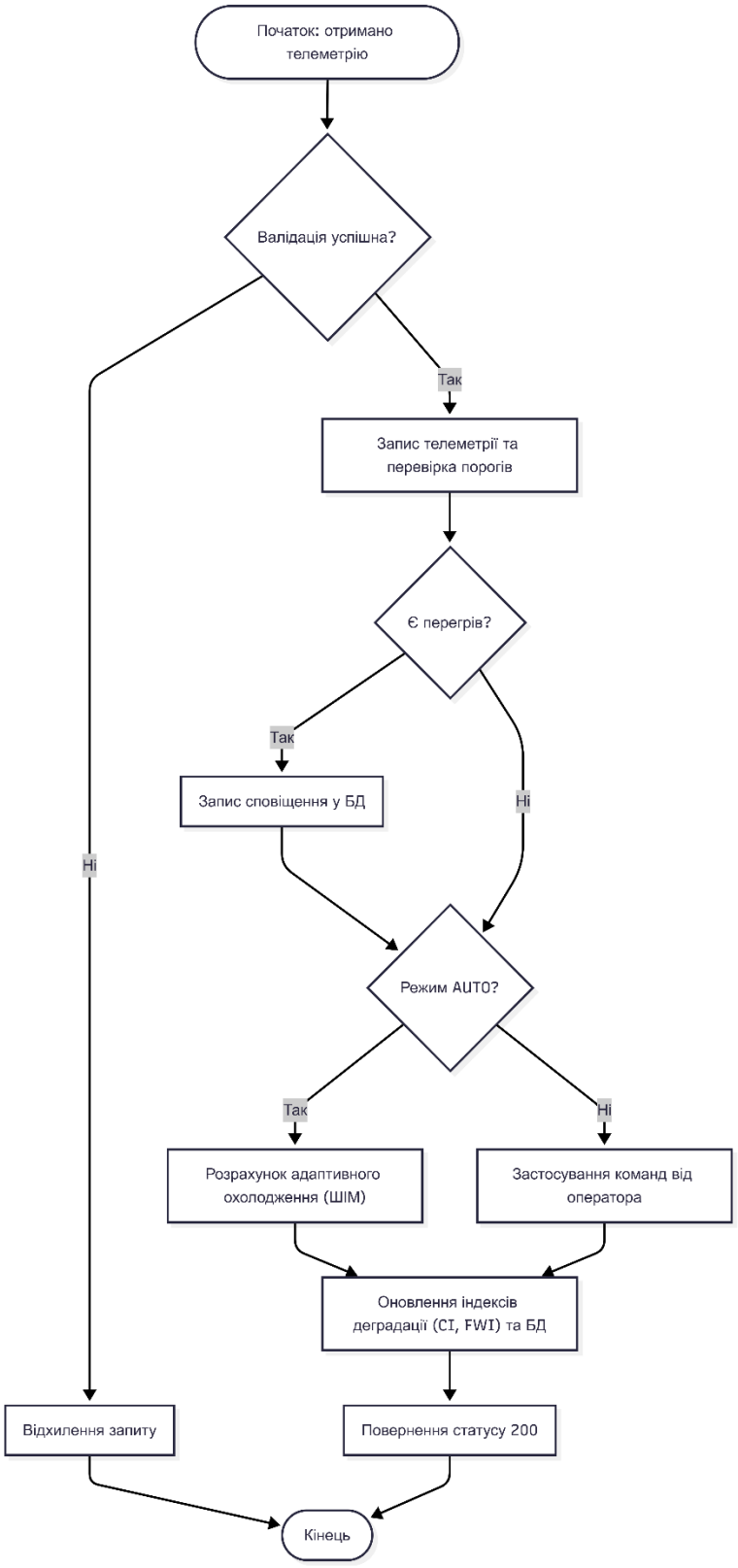


Рисунок А.1 – Алгоритм обробки телеметрії

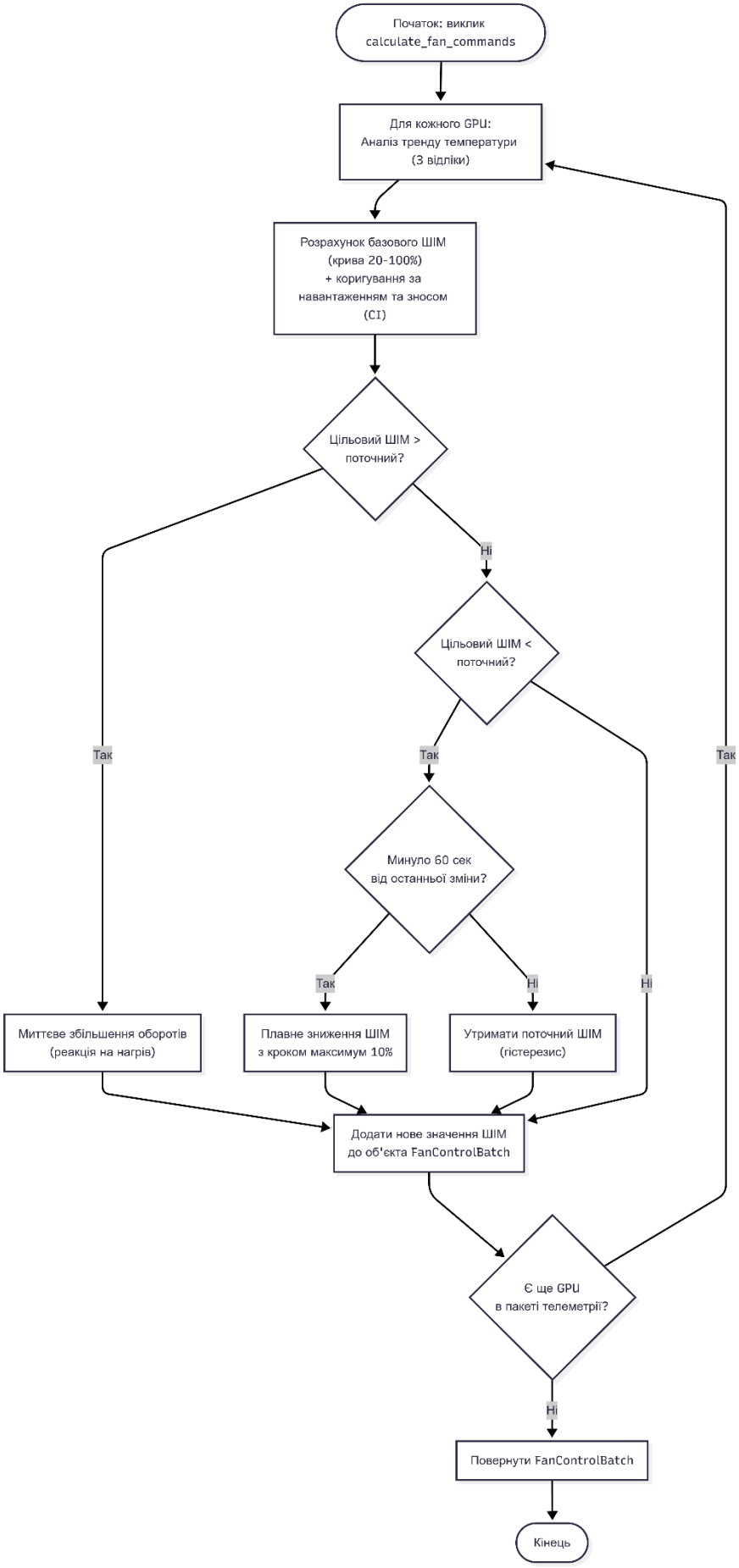


Рисунок А.2 – Алгоритм адаптивного керування вентиляторів

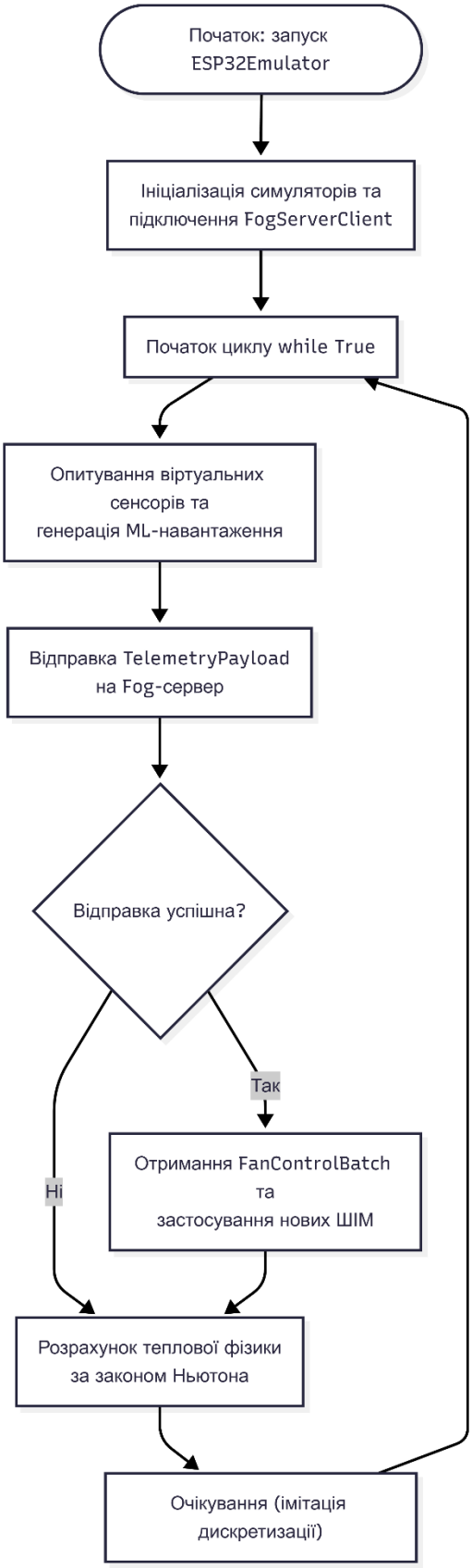


Рисунок А.3 – Алгоритм генерації параметрів середовища і телеметрії тестового стенду

ДОДАТОК Б
ДІАГРАМИ РОЗМІЩЕННЯ

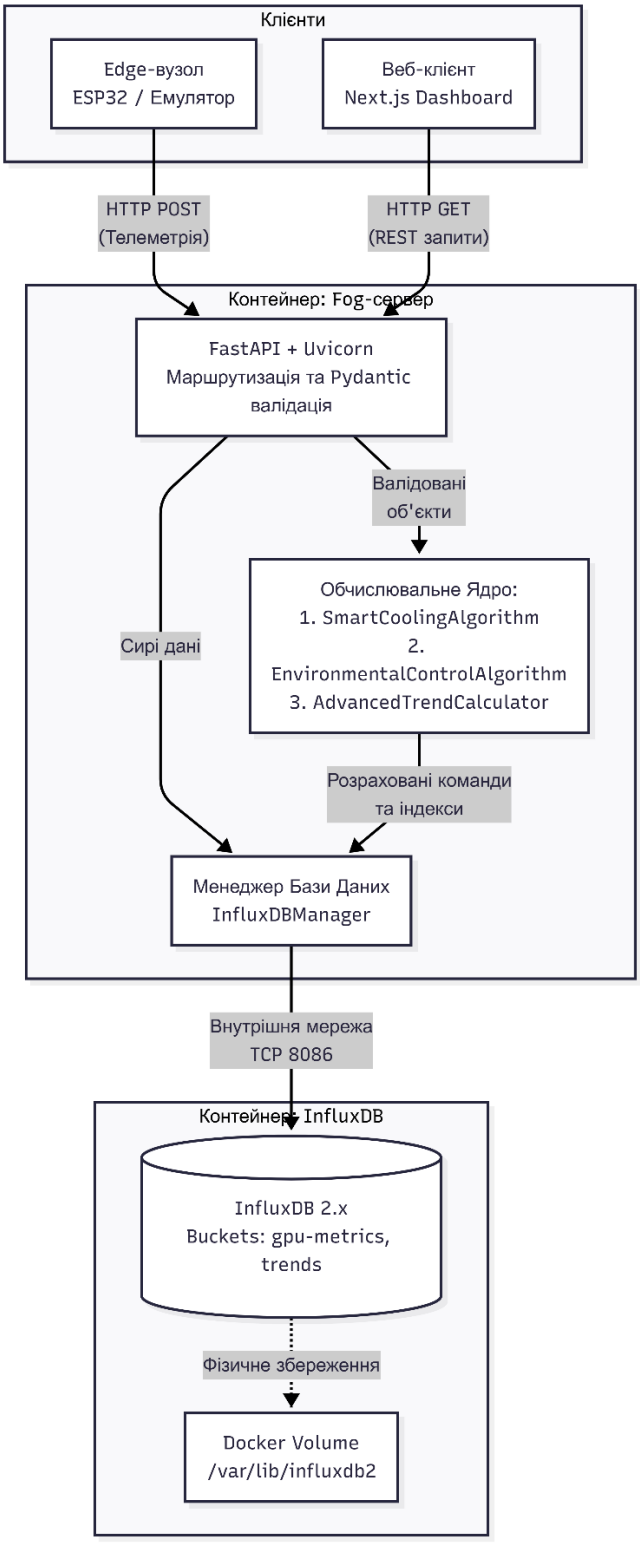


Рисунок Б.1 – Діаграма розміщення сервера туманних обчислень

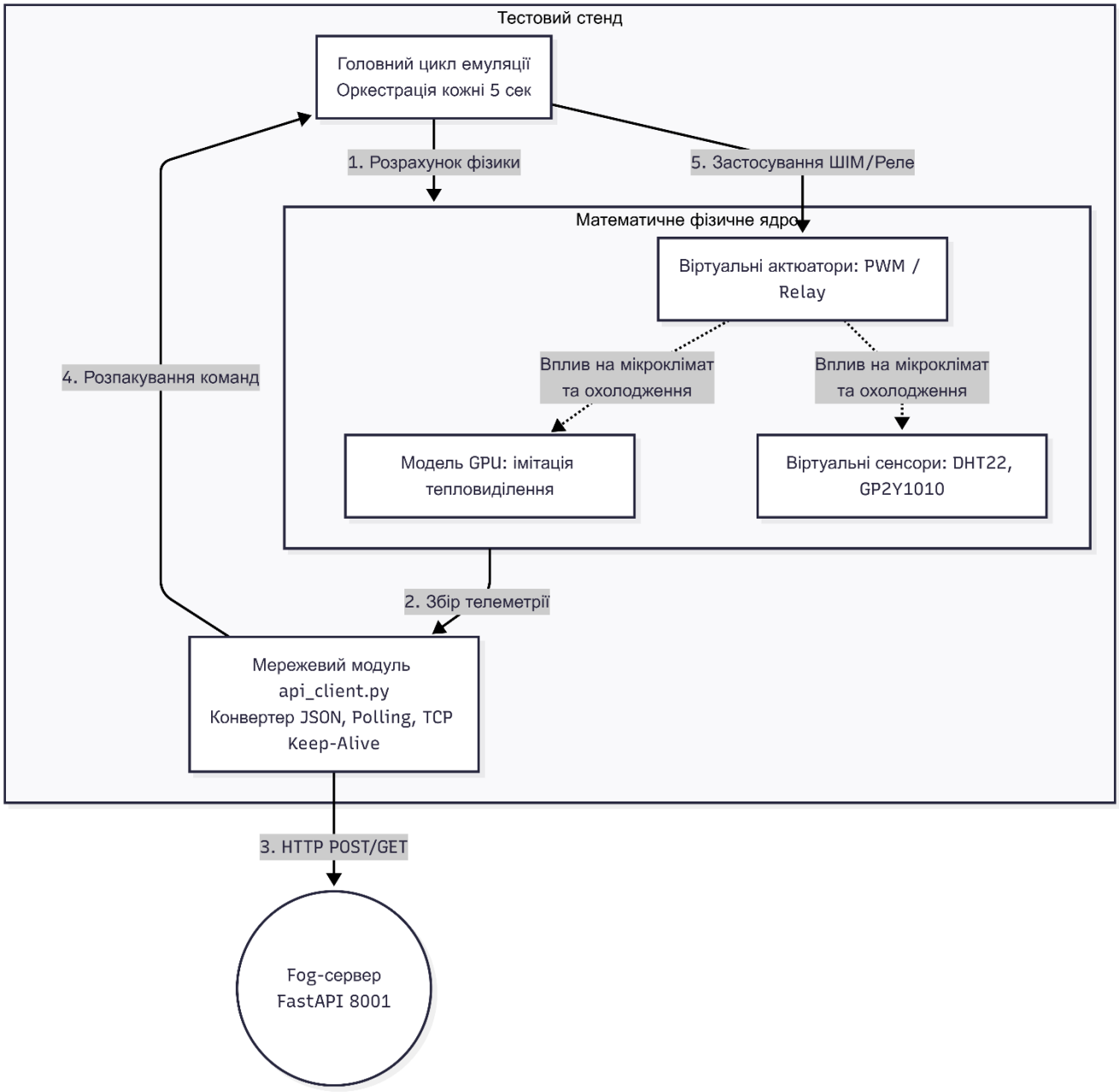


Рисунок Б.2 – Діаграма розміщення тестового стенду

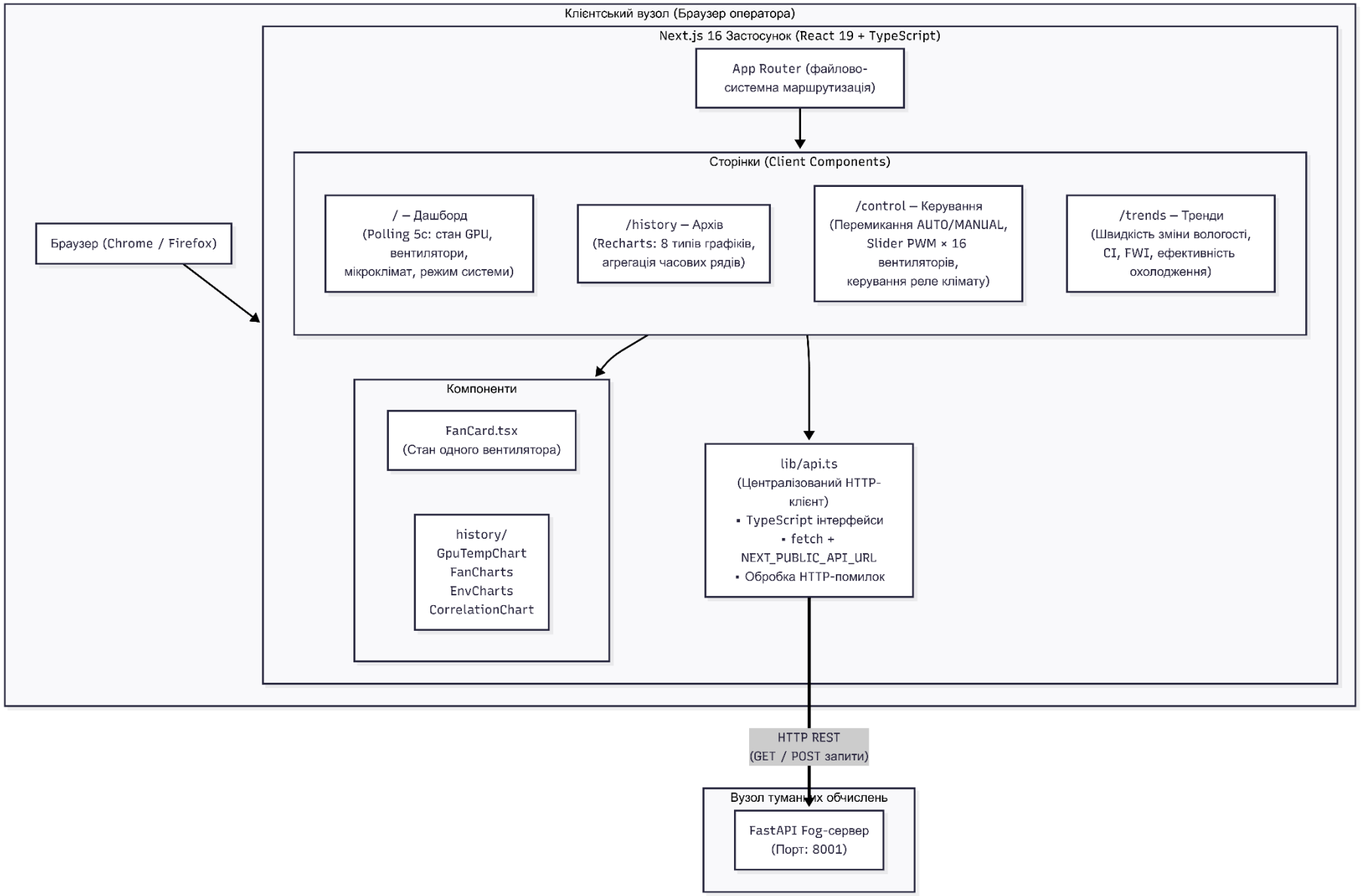


Рисунок Б.3 – Діаграма розміщення веб-системи

ДОДАТОК В
ДІАГРАМИ КЛАСІВ

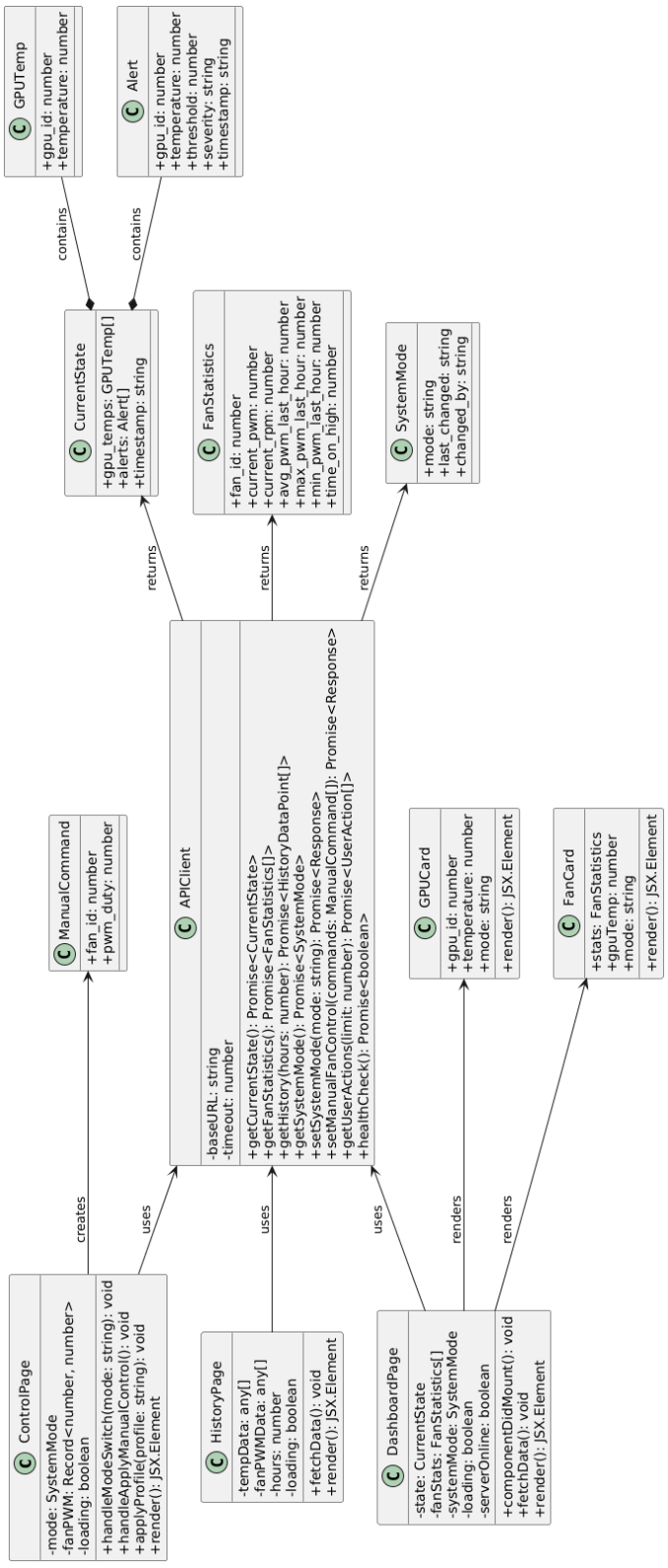


Рисунок В.1 – Діаграма класів веб-системи

ДОДАТОК Г
СЛАЙДИ ПРЕЗЕНТАЦІЇ

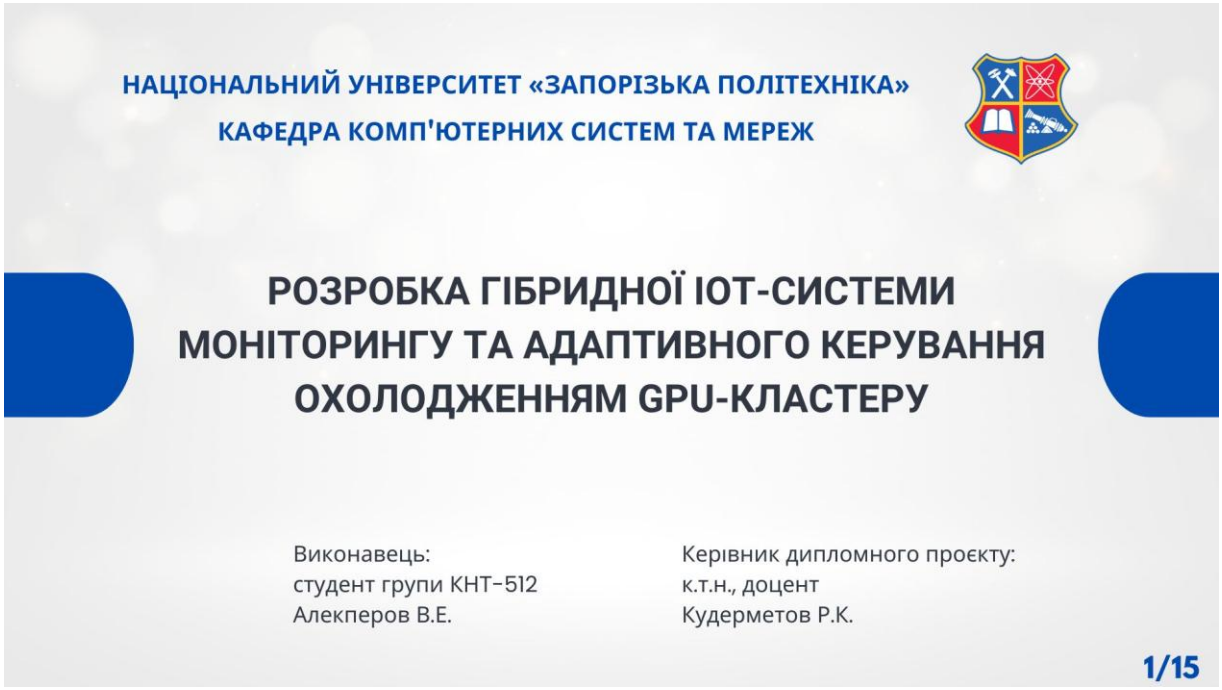


Рисунок Г.1 – Слайд 1

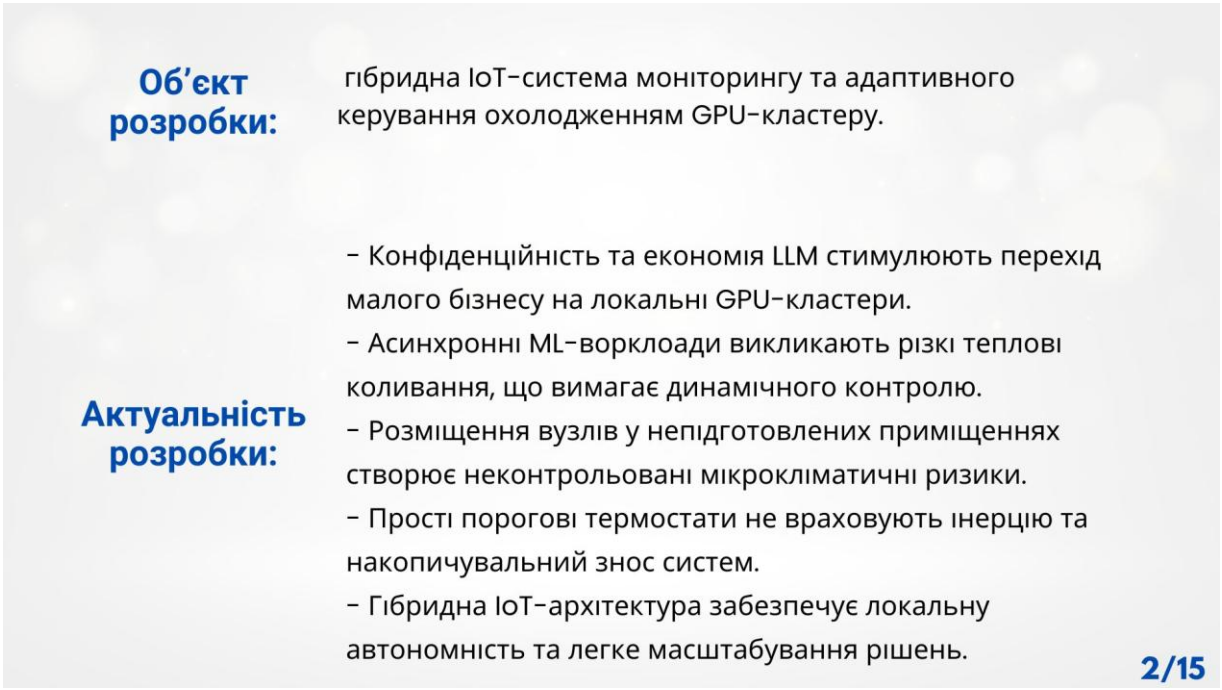


Рисунок Г.2 – Слайд 2

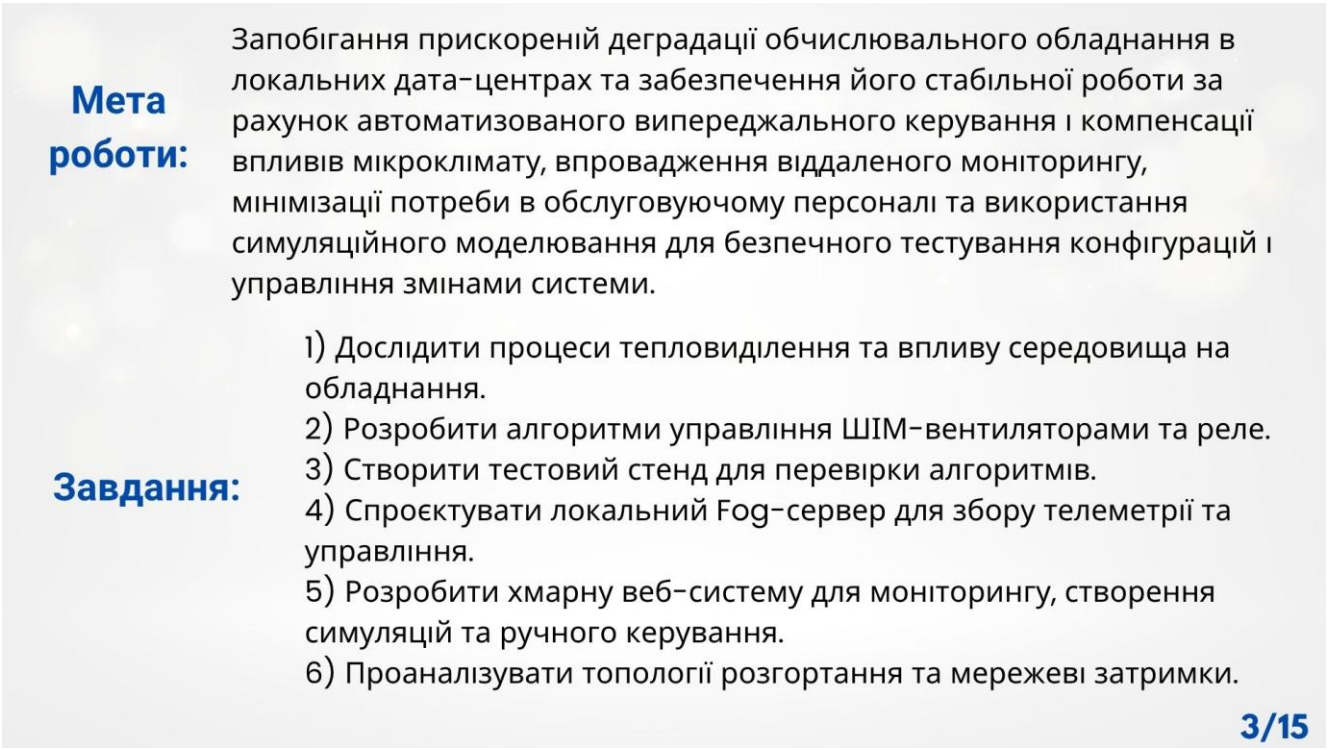


Рисунок Г.3 – Слайд 3

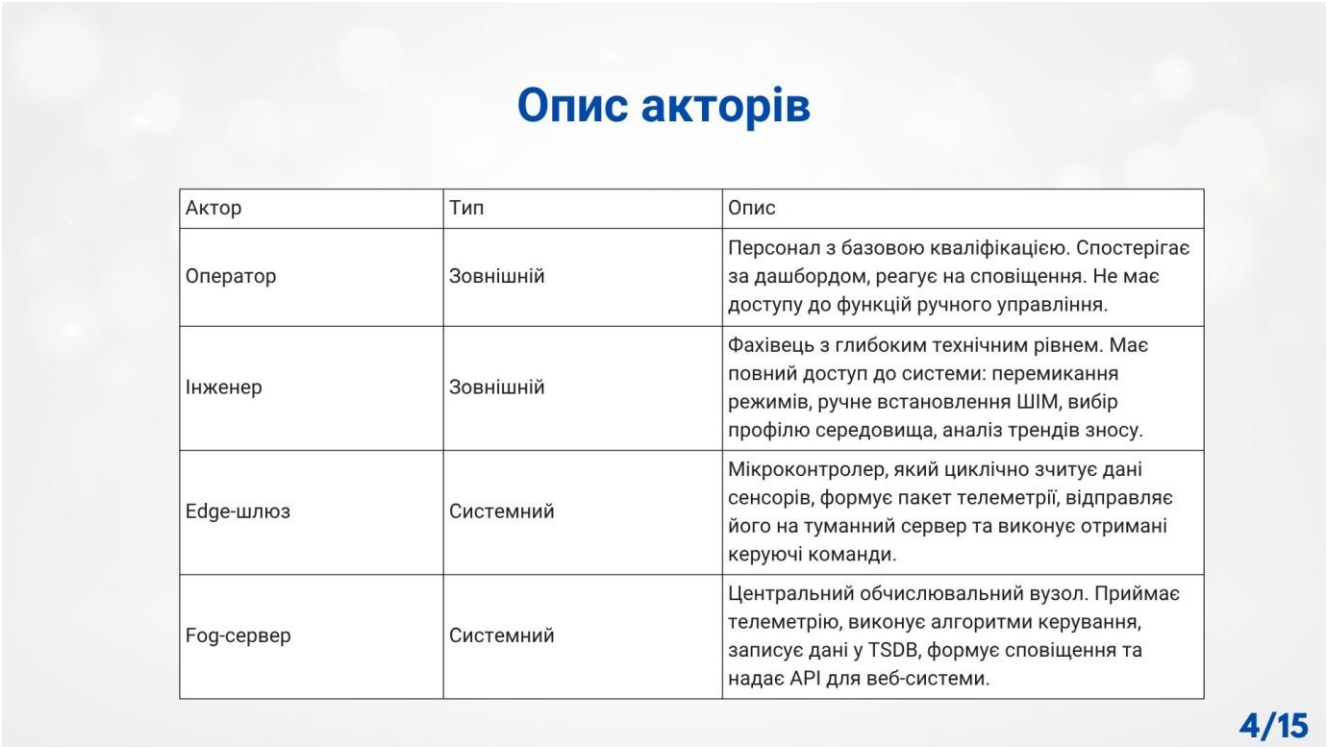


Рисунок Г.4 – Слайд 4

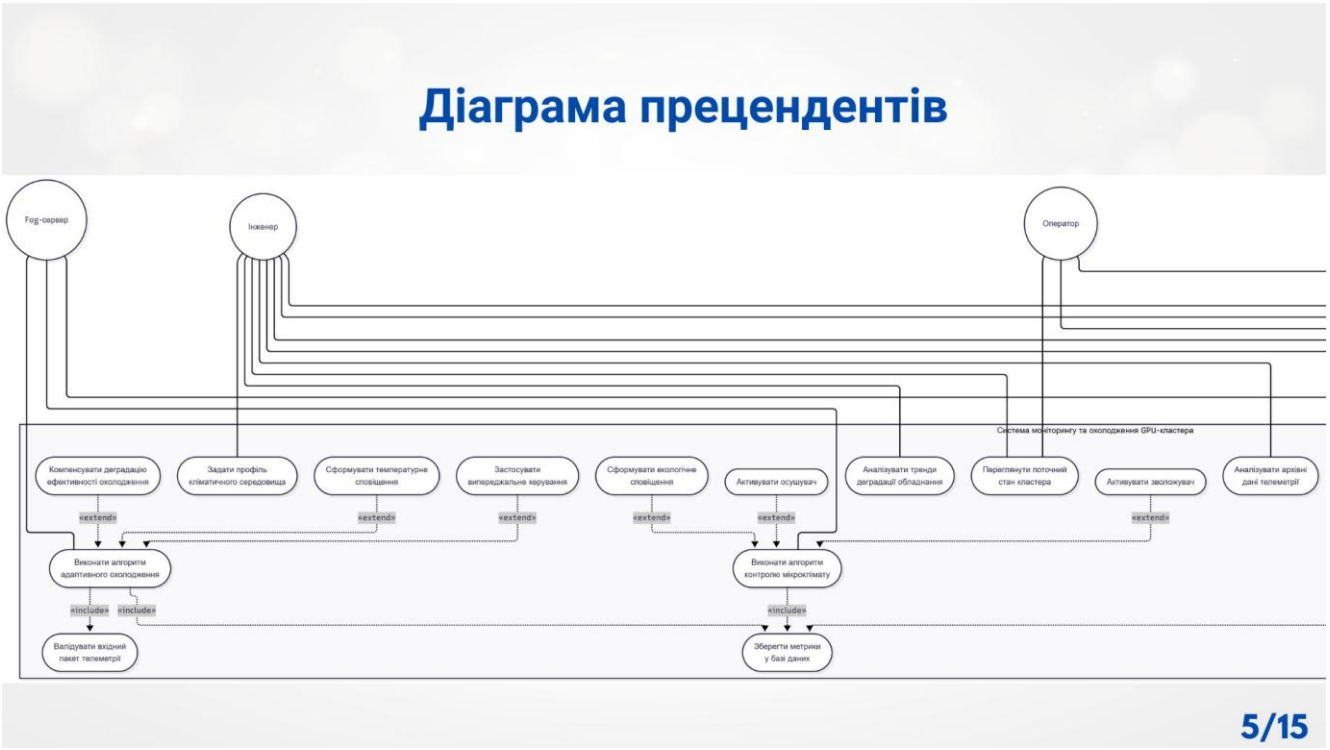


Рисунок Г.5 – Слайд 5

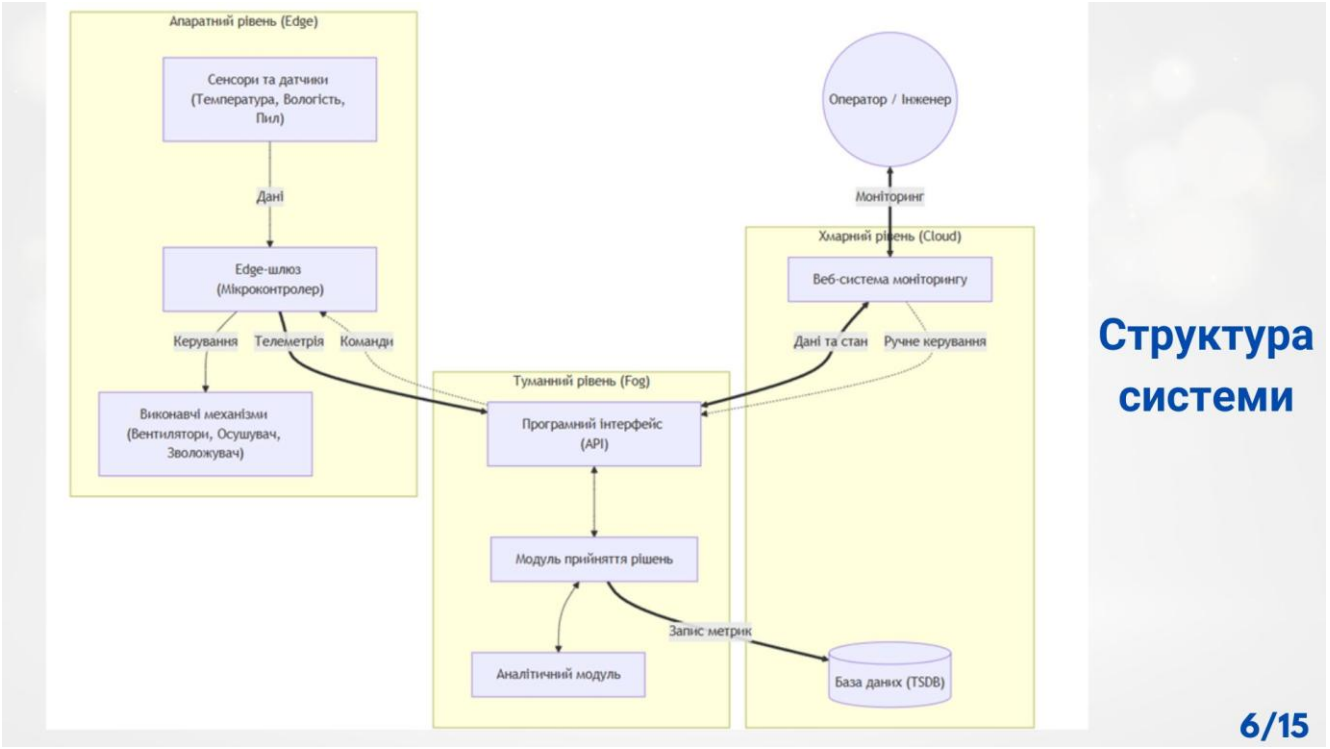


Рисунок Г.6 – Слайд 6

Рисунок Г.7 – Слайд 7

Рисунок Г.8 – Слайд 8

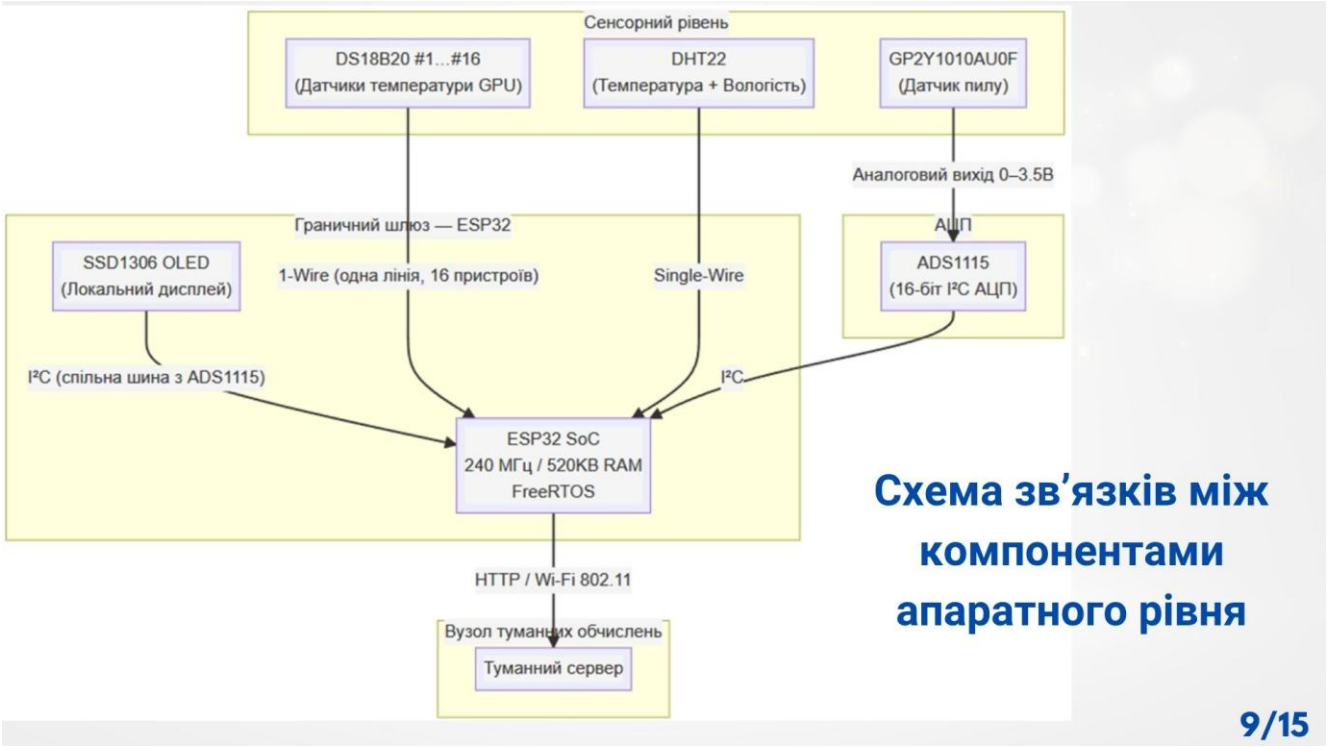
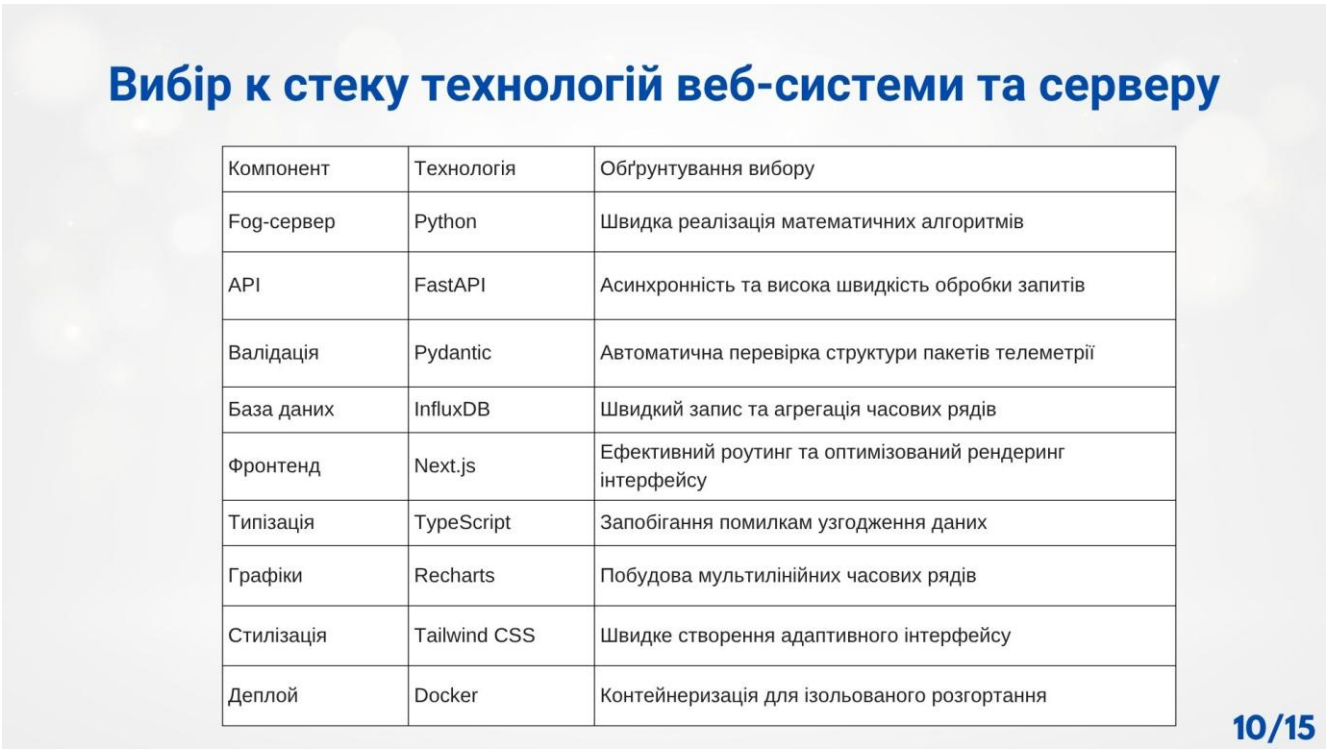


Рисунок Г.9 – Слайд 9



Компонент	Технологія	Обґрунтування вибору
Fog-сервер	Python	Швидка реалізація математичних алгоритмів
API	FastAPI	Асинхронність та висока швидкість обробки запитів
Валідація	Pydantic	Автоматична перевірка структури пакетів телеметрії
База даних	InfluxDB	Швидкий запис та агрегація часових рядів
Фронтенд	Next.js	Ефективний роутинг та оптимізований рендеринг інтерфейсу
Типізація	TypeScript	Запобігання помилкам узгодження даних
Графіки	Recharts	Побудова мультилінійних часових рядів
Стилізація	Tailwind CSS	Швидке створення адаптивного інтерфейсу
Деплой	Docker	Контейнеризація для ізольованого розгортання

Рисунок Г.10 – Слайд 10

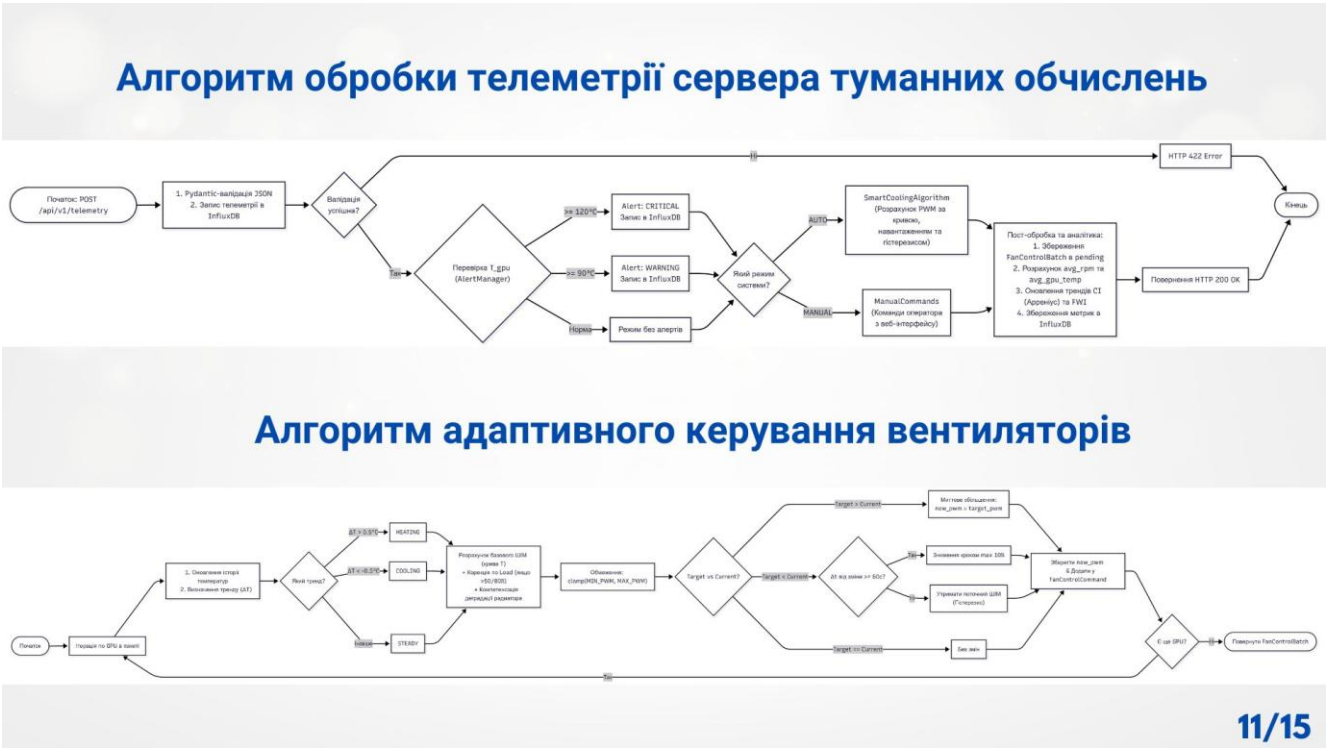


Рисунок Г.11 – Слайд 11

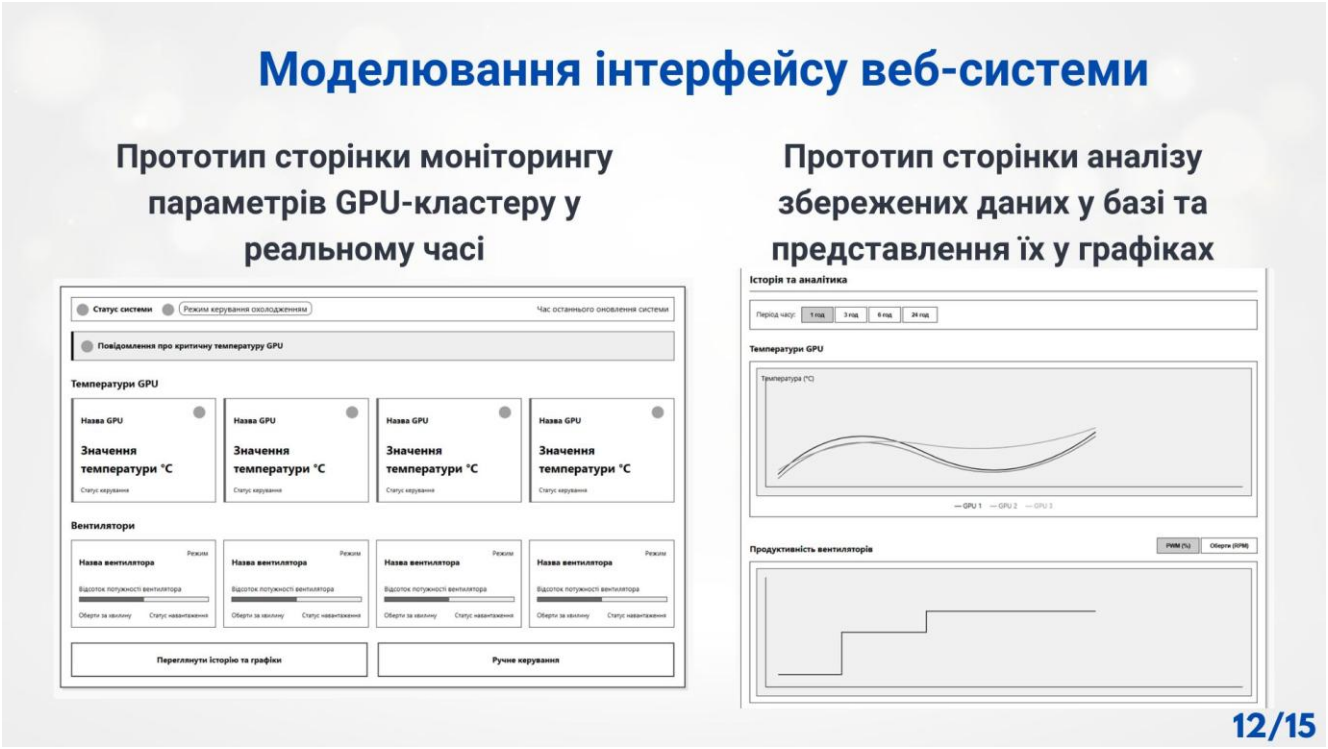


Рисунок Г.12 – Слайд 12

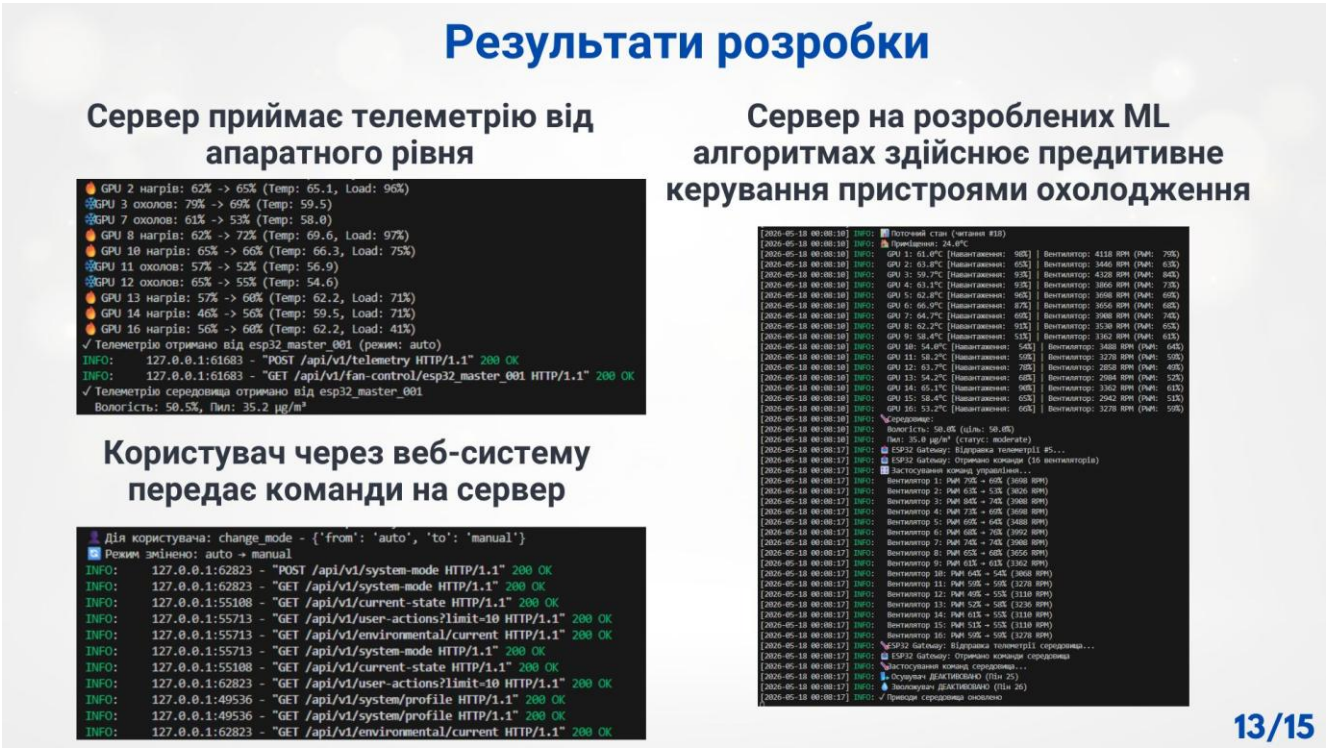


Рисунок Г.13 – Слайд 13

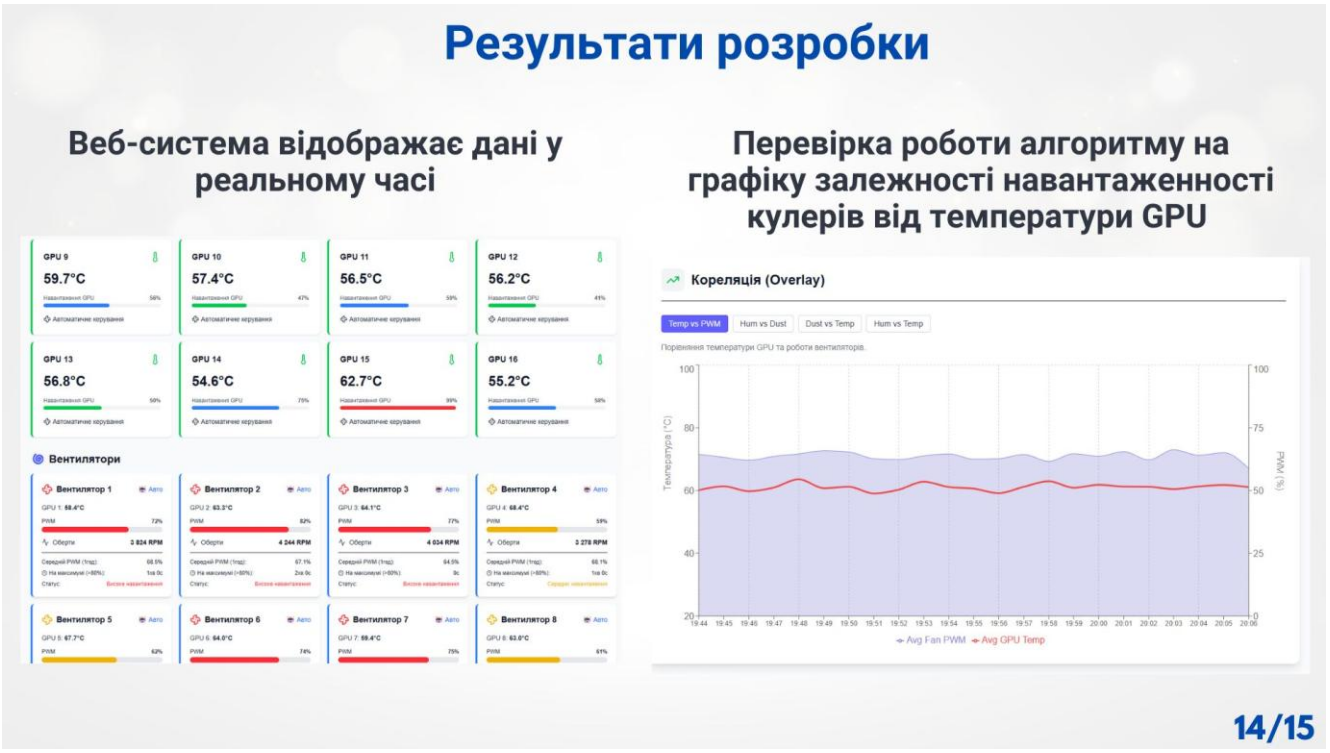


Рисунок Г.14 – Слайд 14

Висновки

В дипломному проєкті було спроєктовано та розроблено гібридну IoT-систему моніторингу та адаптивного керування охолодженням GPU-кластеру

- Туманний сервер виконує збір телеметрії та предитивне управління, а веб-система забезпечує глобальний моніторинг та аналітику
- Розрахунок індексів зносу вентиляторів та корозії дозволяє прогнозувати аварійні ситуації до їх виникнення
- Програмний емулятор забезпечив повне тестування алгоритмів під навантаженням без фінансових витрат на апаратні компоненти
- Можливість локального розгортання бази даних та веб-інтерфейсу повністю ізолює керуючі ендпоінти від зовнішніх кібератак.

15/15

Рисунок Г.15 – Слайд 15