

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

Олександр КОВАЛЬ

«_____» _____ 2026р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»**

спеціальності 121 Інженерія програмного забезпечення

**на тему: «Система для аналізу стану обладнання лабораторії
кіберенергетичних систем засобами граничних обчислень»**

Виконала:

студентка IV курсу, групи ТВ-21

Безклинська Валерія Вікторівна

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

асистент Голець Владислав Олександрович

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

професор д.т.н Отрох Сергій Іванович

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студентка _____

(підпис)

Київ – 2026

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Олександр КОВАЛЬ

(підпис)

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

Безклинській Валерії Вікторівні

(прізвище, ім'я, по батькові)

1. Тема роботи

«Система для аналізу стану обладнання лабораторії кіберенергетичних систем засобами граничних обчислень»

керівник роботи: асистент Голець Владислав Олександрович

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “___” _____ 2026 року

№ _____

2. Строк подання студентом роботи “10” червня _____ 2026 р.

3. Вихідні дані до роботи: літературні джерела інформації, параметри та характеристики обладнання лабораторії, онтологічна модель предметної області, пристрій Raspberry Pi Zero V1.3 та допоміжні деталі, програмні засоби розробки.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) проведення аналізу вже створених систем моніторингу даних, проектування цілої архітектури системи EnergyLab, налаштування роботи одноплатного комп'ютери Raspberry Pi, проведення інтеграції з онтологічною моделлю.

5. Перелік ілюстративного матеріалу: презентація.

6. Дата видачі завдання «31» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025- 30.11.2025	
3	Постановка вимог до проектування системи	01.12.2025- 31.12.2025	
4	Дослідження існуючих рішень	01.02.2026 31.01.2026	
5	Розробка програмного продукту	01.02.2026 03.05.2026	
6	Тестування	04.05.2026 10.05.2026	
7	Захист програмного продукту	11.05.2026 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 31.05.2026	
9	Передзахист	01.06.2026 05.06.2026	
10	Захист	15.06.2026 19.06.2026	

Студент

(підпис)

Валерія Безклинська
(ім'я, прізвище)

Керівник роботи

(підпис)

Владислав ГОЛЕЦЬ
(ім'я, прізвище)

РЕФЕРАТ

Обсяг роботи. Дипломна робота налічує 62 сторінки, 13 рисунків, 2 додатки. А також було використано 18 джерел інформації.

Предмет дослідження даної роботи: система створена для детального аналізу стану обладнання кіберфізичного приміщення з використанням методу граничних обчислень.

Мета роботи: аналіз стану обладнання кіберенергетичної лабораторії засобами граничних обчислень.

Методи цього дослідження: глибинний аналіз літературних джерел, а також як будувати архітектури програмних систем, проведення моделювання системи.

Отримані результати роботи: було розроблено таку архітектуру системи для аналізу стану обладнання лабораторії з використанням граничних обчислень. А також було реалізовано корисний програмний продукт для локальної обробки даних з використанням одноплатного комп'ютера Raspberry Pi. Було досліджено різноманітні підходи до збору даних з обладнання (теплового насосу), реалізовано популярні алгоритми аналізу всіляких параметрів роботи обладнання та визначення його стану, розроблено потужний механізм передачі отриманих результатів аналізу користувачу програми.

Ключові слова: моніторинг, кіберенергетична лабораторія, граничні обчислення, обробка та аналіз даних, Raspberry Pi.

ABSTRACT

Scope of work. The thesis consists of 62 pages, 13 figures, 2 appendices. And also 18 sources of information were used.

The subject of the study of this work: a system created for a detailed analysis of the state of the equipment of a cyber-physical room using the method of boundary calculations.

Purpose of the work: analysis of the state of the equipment of the cyber energy laboratory using boundary value calculations.

Methods of this study: in-depth analysis of literary sources, as well as how to build architectures of software systems, conducting system modeling.

Results of the work: the following system architecture was developed for analyzing the state of the laboratory equipment using boundary calculations. And also a useful software product for local data processing using a single-board computer Raspberry Pi was implemented. Various approaches to collecting data from the equipment (heat pump) were investigated, popular algorithms for analyzing various parameters of the equipment operation and determining its state were implemented, a powerful mechanism for transmitting the obtained analysis results to the program user was developed.

Keywords: monitoring, cyber energy laboratory, edge computing, data processing and analysis, Raspberry Pi..

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ	12
1.1 Загальна характеристика кіберенергетичних систем	12
1.1.1 Що таке кіберенергетичні системи	12
1.1.2 Роль кіберенергетичних систем в енергетиці.....	13
1.1.3 Структура кіберенергетичної системи.....	14
1.2 Особливості лабораторій кіберенергетичних систем.....	15
1.3 Тепловий насос як об'єкт дослідження	16
1.3.1 Принцип роботи теплового насоса.....	17
1.3.2 Основні параметри та режими роботи	18
1.4 Аналіз методів моніторингу стану обладнання	19
1.4.1 Класичні методи моніторингу	19
1.4.2 Централізовані системи моніторингу	20
1.4.3 Недоліки централізованих підходів	21
1.5 Технологія граничних обчислень	22
1.6 Огляд існуючих програмних рішень.....	24
1.6.1 SCADA-системи	24
1.6.2 IoT-платформи.....	25
1.6.3 Системи збору телеметрії.....	26
1.7 Аналіз недоліків існуючих рішень та постановка задачі	27
Висновки до розділу 1	28
2 ЗАСОБИ РОЗРОБКИ СИСТЕМИ	30
2.1 Апаратні засоби системи	30
2.2 Програмні засоби розробки.....	32
2.2.1 Операційна система та налаштування мережі	32
2.2.2 Засоби віддаленого доступу	33
2.2.3 Мова програмування.....	33
2.2.4 Система контролю версій.....	34
2.3 Інтеграція апаратної та програмної частин системи	34
Висновки до розділу 2	35
3 ОПИС АРХІТЕКТУРИ СИСТЕМИ.....	36
3.1 Загальна архітектура	36
3.1 Діаграма прецедентів.....	38
3.2 Діаграма послідовностей.....	39

Висновки до розділу 3	40
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	42
4.1 Реалізація головного модуля аналізу стану обладнання.....	42
4.1.1 Реалізація ML-модуля.....	43
4.1.2 Використані бібліотеки та інструменти.....	44
4.1.3 Модуль діагностики та онтологічного аналізу	45
4.1.4 Комбінований аналіз: ML та rule-based підхід.....	46
4.2 Сервер тривоги та моніторингу системи	47
Висновки до розділу 4	48
5 ІНТЕРФЕЙС ПРОГРАМНОГО ПРОДУКТУ	50
5.1 Головна сторінка дашборду EnergyLab	50
5.2 Сторінка зібраних метрик	55
5.3 Сторінка з онтологією	56
5.4 Сторінка завантаження паспортів	57
Висновки до розділу 5	59
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CPES Cyber-Physical Energy Systems (кіберфізичні енергетичні системи)

IoT - Internet of Things (Інтернет речей)

CPES - Cyber-Physical Energy Systems (кіберфізичні енергетичні системи)

Edge Computing - граничні обчислення

MQTT - Message Queuing Telemetry Transport (протокол передачі телеметрії)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON - JavaScript Object Notation (формат обміну даними)

SSH - Secure Shell (протокол захищеного віддаленого доступу)

ВСТУП

Останніми роками можна спостерігати досить активний розвиток гетерогенних цифрових технологій в такій галузі, як - енергетика, а особливо якщо говорити про кіберенергетичні системи. Дані системи створені для поєднання фізичних та енергетичних процесів з комп'ютерними технологіями.

Кіберенергетичні системи зараз активно впроваджуються у дуже багатьох науково-дослідних лабораторіях, університетах та на підприємствах пов'язаних з енергетикою. Загалом, зростання складності лабораторного приладдя, величезний вибір використаних пристроїв, а також потреба обробки величезних обсягів даних у реальному часі висувають підвищені вимоги до сьогоденних систем моніторингу та аналізу стану.

Традиційні раніше підходи до збору та обробки величезних обсягів даних, які в більшості випадках базуються на застарілих централізованих обчисленнях, часто не можуть забезпечити такої необхідної швидкості реагування в умовах, коли занадто динамічно змінюються параметри потрібного обладнання. Особливо це є критичним для лабораторій таких кіберенергетичних систем, де занадто важлива точність обчислень, оперативність і наявність постійного контролю. Тому все більшої популярності набуває концепція граничних обчислень. Суть - швидка обробка великої кількості даних безпосередньо поблизу джерела їх прямої генерації.

Використання методу граничних обчислень (Edge Computing) дозволяє суттєво зменшити затримки передачі даних, а також підвищити продуктивність та надійність системи, при цьому знижуючи навантаження на головні сервери. Проте є мінус - існуючі програмні рішення не повною мірою здатні покрити та врахувати специфіку таких лабораторій кіберенергетичних систем, а саме необхідність інтеграції лабораторного обладнання, надання масштабованості та роботи в режимі онлайн.

Для вирішення описаних проблем було розроблено програмний продукт EnergyLab для проведення аналізу стану обладнання лабораторії

кіберенергетичних систем, де використовуються граничні обчислення. Його розробка є максимально актуальним завданням сьогодення. Така система дозволить значно підвищити швидкість моніторингу приладів, також забезпечити своєчасне виявлення помилок та збоїв у роботі обладнання та максимально оптимізувати процеси його обслуговування та починки. Обчислення здійснюються на маленькому одноплатовому комп'ютері Raspberry Pi, для написання програми використано широковідому мову програмування Python.

Дана розроблена система EnergyLab здатна забезпечити такий важливий локальний аналіз гетерогенних параметрів лабораторного обладнання, а також може формувати висновки щодо його стану та швидко передавати результати напряму кінцевому користувачу програмного продукту. Особливістю цього підходу в даній дипломній роботі є інтеграція з онтологічною моделлю, що дозволяє програмі інтерпретувати дані моніторингу на основі зібраних структурованих знань та суттєво підвищувати точність оцінки стану приладів лабораторії.

Метою роботи є аналіз стану обладнання лабораторії кіберенергетичних систем із використанням відомих засобів граничних обчислень. Для того, щоб досягнути поставленої вище мети виконано детальний аналіз багатьох сучасних підходів до моніторингу систем, досліджено різноманітні можливості, де успішно застосовано edge computing, а також розроблено архітектуру великої системи спільно з трьома іншими виконавцями, реалізовано програмний код на базі комп'ютера Raspberry Pi з використанням відомої мови Python, а також широко досліджено процес оброблення та опрацювання даних.

Об'єктом дослідження в даній роботі є лабораторія кіберенергетичних систем, а предметом дослідження виступає - система для аналізу стану обладнання з використанням методів граничної обробки. Під час написання роботи проаналізовано різні літературні джерела, проведено моделювання системи.

Практичне значення від отриманих результатів заключається у наданні можливості використовувати дану розроблену систему не тільки у навчальних та

науково-дослідних лабораторіях, а також на робочих підприємствах енергетичної галузі для підвищення загальної швидкості моніторингу приладів та своєчасного виявлення багатьох помилок у роботі заводського обладнання. Запропоноване у даній роботі рішення дозволяє якомога раніше виявити аномалії та цим самим зменшити непотрібні затримки обробки великих даних, підвищити автономність системи загалом та оптимізувати надання технічного обслуговування.

Наукова новизна дипломної роботи полягає у тому що, поєднуються два важливі підходи граничних обчислень із використанням онтологічної моделі розробленої за допомогою документації обладнання, ця інтеграція здатна забезпечити більш глибоку інтерпретацію даних, підвищити швидкість діагностики та створити фундамент для майбутньої побудови інтелектуальних систем збирання даних у кіберенергетичних середовищах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

У першому розділі описано теоретичні особливості даної роботи для розуміння термінології та сутності дослідження. У розділі проаналізовано важливі поняття кіберенергетичних систем, будову їх структуру та опис функціонування. Також наведено головні підходи до організації подібних систем. Також надано детальний опис поняттю граничних обчислень та як їх використовують в житті.

1.1 Загальна характеристика кіберенергетичних систем

У підрозділі розглянуто основні важливі характеристики кіберенергетичних систем (CPES), їх елементи та опис взаємодії між фізичними процесами і айти технологіями.

1.1.1 Що таке кіберенергетичні системи

Кіберенергетичні системи є популярним сучасним видом складних інженерних систем. У собі вони поєднують мультимодальні фізичні процеси звичайного виробництва, простої передачі та споживання енергії з так званими інформаційно-комунікаційними технологіями. Їх основна особливість - це дуже тісна інтеграція апаратних елементів таких як: енергетичне обладнання, велика кількість датчиків, виконавчі пристрої - з програмними засобами збору інформації (наприклад скрипт на обраній мові програмування), її обробки та аналізу.

У CPES системах постійна відбувається взаємодія фізичного рівня з цифровим. Саме це дозволяє швидко проводити моніторинг гетерогенних параметрів приладів у режимі реального часу, а також у майбутньому прогнозування стану обладнання (нормальний стан чи критичний стан) та автоматичне прийняття висновків. У своїй книзі Дж. Парашар наголошує, що кіберфізичні системи поєднують елементи реального та віртуального світів для досягнення єдиної мети [1]. Це все означає, що керування важливими

енергетичними процесами у приладах переходить від класичних добре вивчених централізованих моделей до нових інтелектуальних розподілених систем.

Отже, у висновку кіберенергетичні системи є важливими для того, щоб забезпечити підняття рівня автоматизації і одночасного зменшення впливу людського фактора на роботу. Вони підвищують рівень загальної надійності важливих енергетичних процесів.

1.1.2 Роль кіберенергетичних систем в енергетиці

У енергетичній галузі CPES відіграють дуже важливу, можна навіть сказати ключову роль у трансформації звичних нам енергетичних мереж у більш розумні та легко адаптивні інфраструктури. Вони створюються для забезпечення інтеграції розподілених джерел енергії, систем накопичення у загальну легко керовану мережу.

Основними важливими функціями CPES є:

- постійний безперервний моніторинг над станом енергетичного обладнання;
- отримання та обробка, інтерпретація величезних обсягів даних у режимі реального часу;
- забезпечення рівного розподілу потоків енергії між всіма компонентами наданої системи;
- своєчасне та автоматичне виявлення різноманітних збоїв у роботі обладнання та швидке виправлення аварійних ситуацій;
- створення прогнозу можливих несправностей у майбутньому.

Застосування CPES дозволяє явно підвищити використання потрібних енергоресурсів. Також знижує витрати енергії при передачі та дозволяє забезпечити максимально гнучке управління навантаженням системи.

У книзі Гуана Х. зазначається, що CPES (кіберфізична система) надає змогу швидко керувати фізичними об'єктами в реальному часі та з високою продуктивністю. Зараз формується нове покоління кіберенергетичної мережі з

самосвідомістю, саморегуляцією та автономним виконанням дій [2]. Це все підкреслює повний перехід до автономних енергетичних систем, які здатні самостійно та швидко адаптуватися до постійних змін умов роботи без постійного втручання людини. Відсутність впливу людського фактору забезпечує зниження потенційних помилок у роботі системи.

1.1.3 Структура кіберенергетичної системи

CPES загалом доцільно розглядати як велику багаторівневу структуру. Система містить два головних взаємопов'язаних компоненти:

- фізична складова;
- кібернетична (інформаційна) складова.

Перша фізична підсистема складається з матеріальних об'єктів енергетичної інфраструктури, наприклад: генератори, теплові насоси, трансформатори, різні вимірювальні пристрої, важливі виконавчі механізми, а також технологічні процеси промисловості і розподілу енергії. Характеризується в основному енергетичними та матеріальними станами.

Друга частина, а саме кібернетична підсистема, на противагу першій, є відповідальною за збір інформації, її передачу та опрацювання. Ця підсистема надає інформаційні зв'язки між всіма елементами та формує головну логіку прийняття рішень.

Невлюдов І.Ш. у своїй книзі пише, що на відміну від фізичного аспекту, в кібернетичному приймаються до уваги тільки інформаційні зв'язки між різними елементами (підсистемами). Єдність забезпечується на основі інформації та управління [3]. Це підкреслює, що саме інформаційний обмін між елементами є головною ознакою функціонування всієї системи загалом.

Додатково варто відзначити, що швидкість роботи кіберенергетичної системи великою мірою залежить від якості взаємодії між різними рівнями: фізичним та кібернетичним. Будь-які порушення їх взаємної роботи скоріше всього

призведуть до зниження точності керування системою, а також сильних затримок у прийнятті рішень системою та великому зменшенні надійності в цілому.

1.2 Особливості лабораторій кіберенергетичних систем

Більшість сучасних лабораторій кіберенергетичних систем світу використовують дуже складне та різноманітне обладнання. Це повністю забезпечує дослідження процесів створення, передачі, перерозподілу та витрат енергії. Також занадто важливу роль відіграють інформаційні технології.

В основному виділяють таке лабораторне обладнання: системи автоматизованого керування (SCADA), теплові насоси, різноманітні датчики параметрів - температури, витрати, тиску, енергоспоживання. Щодо інформаційних технологій виділяють - контролери та мікрокомп'ютери (Raspberry Pi), які є відповідальними за збір, аналіз і передачу даних. Все обладнання кіберенергетичної лабораторії працює в одній зв'язці, де усі фізичні процеси дуже тісно пов'язані з цифровими системами управління. Саме це і забезпечує максимально підвищені вимоги до точності, надійності та постійної обробки великої кількості інформації.

Усі експерименти, які представлені у даній роботі, були проведені на базі навчальної кіберенергетичної лабораторії кафедри. Дослідження проводились в умовах, які є максимально наближеними до реального життя. Навчальні лабораторії виконують надважливу роль у формуванні нових фахівців, вони не тільки забезпечують можливість ознайомлення на практиці з новітніми кіберенергетичними системами, а й надають змогу відпрацювати навички роботи з приладами та протестувати програмні рішення у середовищі, яке контролюється. При цьому такі лабораторії мають змогу поєднати як навчальну та і дослідницьку функції, і це все дозволяє детально вивчати вже існуючі технології та ще розробляти додатково нові підходи для моніторингу даних та їх аналізу.

Для тривалого функціонування таких лабораторій кіберенергетичних систем є потреба в організації безперервного моніторингу параметрів роботи обладнання.

До цільових вимог моніторингу даних належать оперативність отримання поточкових даних з обладнання у реальному часі, а також надвисока точність усіх вимірювань, стійкість до можливих збоїв та надання масштабування системи при збільшенні або зменшенні кількості підключених пристроїв. Дуже важливою також є здатність такої системи дуже швидко реагувати на відхилення у роботі лабораторного обладнання та забезпечувати цільового користувача тільки актуальною інформацією для прийняття майбутніх рішень.

Також не рідко у процесі експлуатації описаних кіберенергетичних систем виникає ряд проблем, які здатні суттєво впливати на якість моніторингу. До таких випадків належать збої в роботі лабораторного обладнання або мережевої інфраструктури, також часте перевантаження каналів передачі гетерогенних даних при надвеликій кількості пристроїв, а також часто зашумленість здійснювальних вимірювань, що ускладнює майбутній аналіз. У додатку можуть виникати постійні затримки у передачі великого обсягу даних, що є дуже критичним для тих систем, які потребують дуже швидкого реагування. Сукупність зазначених факторів обумовлює постійну необхідність розробки гарних рішень для локальної обробки телеметрії та майбутнє підвищення надійності різних систем моніторингу.

1.3 Тепловий насос як об'єкт дослідження

В цій роботі головним об'єктом дослідження було обрано лабораторний тепловий насос. А саме модель Mitsubishi electric cylinder unit ehst20. Ця модель знаходиться в навчальній кіберенергетичній лабораторії.

Тепловий насос є дуже важливим елементом усіх сучасних енергетичних систем. Даний об'єкт переносить теплову енергію з одного середовища в інше. Всередині такого насоса є теплообмінник, він нагріває воду (іншими словами теплоносієм). Ця вода потім йде в систему опалення. У більшості лабораторій кіберенергетичних систем, зокрема в кафедральній лабораторії - теплові насоси

виступають як об'єкти дослідження, це дозволяє простіше аналізувати процеси енергоспоживання, теплообміну обладнання.

Важливою особливістю такого теплового насоса є наявність багатьох параметрів, що характеризують його роботу. До них належать температура, витрата теплоносія (води) та споживання електроенергії. Контроль всіх цих параметрів дозволяє якісно оцінювати швидкість функціонування всієї системи та миттєво виявляти можливі відхилення від норми роботи.

1.3.1 Принцип роботи теплового насоса

У цьому підпункті наданий більш детальний опис принципу роботи такого теплового насоса. Подібний тепловий насос здатний переносити теплову енергію від вихідного джерела з більш низькою температурою до середовища з вищою температурою і все це здійснюється за рахунок витрати зовнішньої енергії. Основа роботи полягає в термодинамічному циклі. Такий цикл включає важливі процеси, а саме: випаровування, стиснення, конденсації та розширення холодоагенту (робочого тіла). Детальніше зображено на рисунку 1.1.

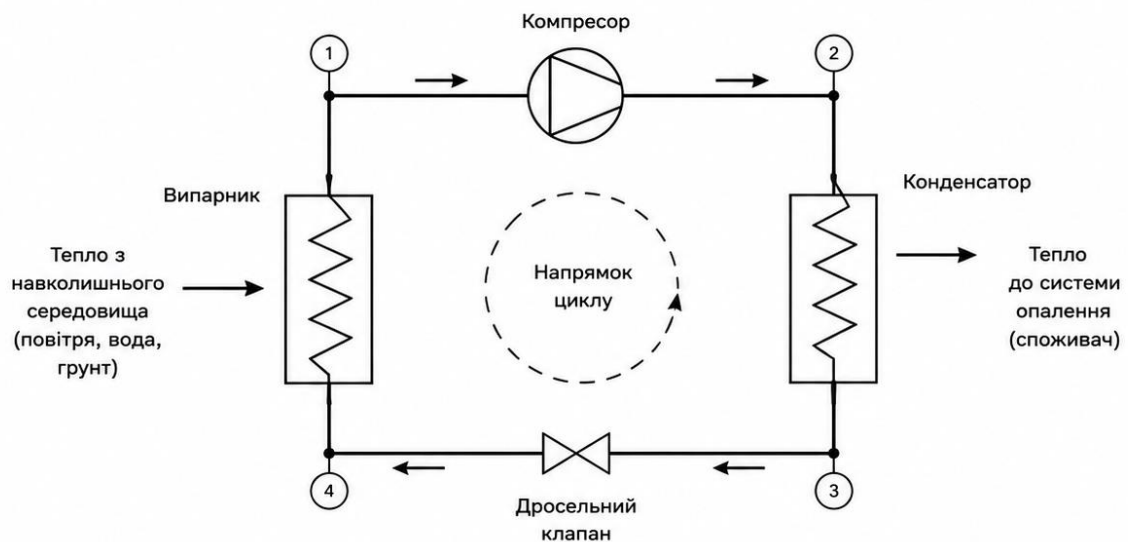


Рисунок 1.1 – Схема роботи теплового насоса

Перший етап полягає в тому, що робоче тіло у випарнику вбирає тепло з навколишнього середовища, а саме з повітря, води або ґрунту. Потім переходить у газоподібний стан. Далі компресор стискає газ, внаслідок цього підвищуються його температура і тиск. Далі у елементі конденсаторі відбувається вже передача тепла до системи опалення. Наступним етапом після цього холодоагент (робоче тіло) конденсується. В завершенні відбувається розширення холодоагенту через дросельний клапан, в подальшому це призводить до зниження температури та тиску холодоагенту, після чого весь цей процес повторюється знову.

Таким чином, тепловий насос забезпечує мінімальне використання енергії навколишнього середовища та дозволяє отримувати більшу кількість теплової енергії у порівнянні з витраченою електричною. Арсен'єв В. М у своєму посібнику зазначає, що одним із простих способів економії паливних ресурсів, а також захисту навколишнього середовища є широке впровадження теплонасосних установок, в яких низькопотенціальні теплові потоки перетворюються в потоки із вищим температурним рівнем [4]. Це робить його перспективним об'єктом для дослідження у системах моніторингу та аналізу енергетичних процесів.

1.3.2 Основні параметри та режими роботи

Основними параметрами лабораторного теплового насоса, які вимірювались в даній роботі, є температура подачі та зворотки теплоносія, температура зовнішнього середовища, потужність, а також рівень споживання електроенергії. Також, що є не менш важливо, враховується стан компресора та режим роботи кіберенергетичної системи.

Такий пристрій, як тепловий насос може працювати у багатьох режимах, зокрема режимі нагрівання, режимі підтримки температури та режимі охолодження. Перехід між різними режимами здійснюється в залежності від параметрів, які були задані та поточного загального стану системи. Весь контроль

цих наданих режимів дозволяє оцінювати загальну ефективність роботи теплового насоса та виявляти можливі відхилення.

З цього теплового насоса дані можуть бути зчитані багатьма методами. Було обрано найпростіший та найдешевший метод зчитати дані через інтерфейс CN105. А саме такі дані, як гетерогенні температурні показники, потужність насоса, різні сигнали керування та важлива службова інформація для моніторингу.

Всі отримані в результаті зчитування дані використовуються для майбутньої обробки на edge-пристрої (Raspberry Pi в даній роботі). Такий метод дозволяє виконувати аналіз стану лабораторного обладнання в режимі реального часу та швидко видавати відповідні чіткі висновки щодо його роботи.

1.4 Аналіз методів моніторингу стану обладнання

Постійний процес моніторингу стану лабораторного обладнання є дуже важливим. Він дозволяє забезпечити безпечну роботу усіх технічних об'єктів. Руденко І. у своїй роботі пише, що системи моніторингу здатні забезпечити автоматичне керування також диспетчеризацію інженерного обладнання будь-якої будівлі, роботу агрегатів, підтримання параметрів роботи обладнання, які були задані та їх оперативну зміну [5]. На даний час існує декілька підходів до проведення такого моніторингу. Вони відрізняються за архітектурою, принципом роботи.

1.4.1 Класичні методи моніторингу

Якщо говорити про класичні методи збору даних (моніторингу), то вони передбачають в основному використання простих локальних вимірювальних приладів, а також систем контролю, які просто встановлюються на обладнанні. До таких приладів належать:

- мультимодальні аналогові датчики параметрів таких як: температура, тиск, витрати енергії;

- контролери локального типу (ПЛК);
- сигналізаційні системи;

У подібних лабораторних систем майже вся обробка інформації найчастіше відбувається безпосередньо на прямо місці або дуже часто взагалі відсутня, а зібрані дані просто використовуються для оперативного контролю.

Найбільшою важливою перевагою більшості класичних методів моніторингу даних є простота в реалізації та надійність. Такі методи точно не потребують занадто складної інфраструктури та дуже легко можуть бути впроваджені у життя. Однак є і мінуси – так звані класичні підходи до збору даних мають ряд суттєвих обмежень, серед них можна виділити наступні: відсутність автоматизованого аналізу – висока залежність від оператора, також висока складність інтеграції з іншими різноманітними системами.

1.4.2 Централізовані системи моніторингу

Зараз у світі широкого поширення набули так звані централізовані системи моніторингу даних. Головним принципом даних систем є те, що необроблені дані з лабораторного обладнання передаються до єдиного сервера або обраної хмарної платформи, де і здійснюється їх основна обробка, зберігання та подальший аналіз.

Найтиповішою структурою такої централізованої системи є:

- датчики, контролери - іншими словами джерела, які зчитують необроблені дані;
- мережі зв'язку – канали передачі даних;
- центральний сервер;
- ліцензійне програмне забезпечення для детального аналізу та візуалізації даних.

Системи з централізованим сервером мають певні переваги. Вони з легкістю дозволяють:

- здійснювати моніторинг на відстані;

- накопичувати величезні обсяги зібраної інформації;
- використовувати досить складні алгоритми для проведення розрахунків;
- інтегрувати різноманітні типи обладнання в одну систему.

1.4.3 Недоліки централізованих підходів

Але централізовані системи мають також ряд суттєвих для роботи великих підприємств недоліків. Ці недоліки стають дуже критичними, якщо говорити в контексті кіберенергетичних систем.

У книзі Кучернюка П. можна прочитати, що такий варіант, як централізована система не може забезпечити повної необхідної масштабованості по продуктивності обладнання, тому що тільки єдиний оператор вимушений буде обробляти абсолютно весь потік повідомлень від всіх поточних агентів, що, при декількох величезній кількості керованих об'єктів, зажадає надвисокопродуктивної системи для роботи оператора і це все перенавантажуватиме службовою управляючою інформацією доступні канали передачі даних [6].

Також процес передачі великої кількості даних до конкретного сервера потребує багато часу. Особливо, коли використовуються мережі загального користування. У підсумку відбуваються затримки у прийнятті важливих рішень, які є критичним для систем загалом. Бо системи потребують дуже швидкої реакції на певні зміни стану обладнання.

Наприклад у випадку, коли існує дуже велика кількість підключених пристроїв величезний обсяг даних передається до одного центру обробки. Це все і створює непотрібне навантаження на єдину мережу, що в майбутньому може спричинити втрату багатьох пакетів, зниження швидкості передачі даних та загалом спричинити збої в роботі суцільної системи.

Чим більше пристроїв підключено, тим більше централізована система потребує потужних серверів і додаткового розширення інфраструктури. Це основний фактор ускладнення процесу масштабування, а також це все може

призвести до небажаного зростання витрат на починку та підтримку системи в майбутньому.

Ще одним недоліком є те, що всі централізовані системи мають одну точку відмови. Якщо ця точка виходить з ладу з певних причин або відбувається втрата зв'язку з ним система моніторингу може стати недоступною та припинити свою роботу. Це дуже негативно впливає на безпеку продукту.

1.5 Технологія граничних обчислень

В наступному пункті розглянуто детально альтернативу централізованим системам. А саме - так звана технологія граничних обчислень. Дана технологія передбачає фактично перенесення обробки даних набагато ближче до джерела їх створення.

Туагі Б. пише у своєму підручнику, що периферійні або граничні обчислення - це відома парадигма розподілених обчислень, вона орієнтована на максимальне наближення обчислень до джерела, яке надає дані. Це все забезпечує швидшу обробку та набагато вищу якість представленого аналізу і при цьому всьому одночасно зменшує затримки та пропускну здатність обладнання [7].

Граничні обчислення відрізняються від традиційних вищерозглянутих централізованих систем, тим що велика частина всіх обчислень виконується саме на конкретному пристрої або поруч із ним.

Глобальна ідея заключається в тому, що дані не обов'язково передавати до тільки одного центрального вузла - ці дані можна просто заздалегідь обробити десь локально та лише потім передати отриманий результат. Такий варіант дозволяє легко зменшити непотрібне навантаження на обмежену мережу.

Принцип роботи периферічних обрахунків засновується на тому, щоб рівномірно розподілити обчислення між різнорівневими пристроями. Всі дані, які отримуємо від генерації, наприклад тепловим насосом, спочатку надходять на певний edge-пристрій, наприклад Raspberry Pi - локальний комп'ютер.

На цьому локальному рівні можуть бути виконані наступні операції:

- зчитування необроблених сирих даних з обладнання через певний інтерфейс за допомогою обраного протоколу (наприклад, через Modbus);
- попередня мінімальна обробка (наприклад фільтрація, усереднення, або ж нормалізація);
- мінімальний аналіз стану системи загалом;
- формування загальних висновків.

І тільки після цих всіх процесів до єдиної центральної системи або певної хмари передається найнеобхідніша інформація. Такою інформацією можуть бути: певні агреговані дані, якщо трапились збої - повідомлення про них, загальні результати аналізу.

Використання такої технології, як edge computing є дуже актуальним для лабораторій кіберенергетичних систем.

До найголовніших переваг описано вище підходу належать наступні:

- значне зменшення загального обсягу переданих даних тому, що передаються тільки важливі результати;
- максимально швидке реагування - обробка відбувається локально, що дозволяє оперативно виявляти відхилення;
- зниження навантаження на мережу та сервери підприємства;
- глобальна автономність системи - система може працювати навіть при тривалій відсутності з'єднання з єдиною центральною системою;
- надання надійності системі - така локальна обробка зменшує залежність від інших зовнішніх сервісів.

Отже, можна зробити такий підсумок у порівнянні двох описаних вище підходів. Граничні обчислення відомі тим, що велика частина процесу обробки даних виконується локально, це дозволяє зменшити навантаження на спільну мережу, забезпечує дуже швидку реакцію на різноманітні події та надає можливість автономної роботи, яка не потребує безперебійного постійного підключення до сервера. Напротивагу інший підхід - хмарні або іншими словами централізовані

системи - передача абсолютно всіх даних на один сервер для майбутньої обробки. І це все призводить до набагато більшої затримки, в залежності від якості мережевого з'єднання та навантаження.

У підсумку, технологія граничних обчислень дозволяє максимально оптимізувати весь процес обробки отриманих даних, при цьому забезпечує набагато швидшу роботу систем моніторингу в кіберенергетичних лабораторіях.

1.6 Огляд існуючих програмних рішень

Кожне вже розроблене рішення має деякі переваги та деякі обмеження.

Наприклад, вже відомі SCADA-системи можуть забезпечувати надвисоку надійність та повний контроль, також не менш відомі IoT-платформи характеризуються гнучкістю і здатністю до масштабованості, а системи телеметрії означають максимально швидкий збір та детальний аналіз технічних даних.

Водночас вони всі мають спільні недоліки - залежність від описаних вище централізованих компонентів, також затримки передачі даних, все це обґрунтовує доцільність використання підходів граничної обробки даних у сьогоденних кіберенергетичних системах.

1.6.1 SCADA-системи

Детальніше про SCADA-системи (Supervisory Control and Data Acquisition). Вони є звичним, класичним підходом до слідкування за станом та керуванням промисловими об'єктами. Такі системи використовуються для швидкого збору, надійної обробки, а також візуалізації даних з лабораторного обладнання в реальному часі. SCADA-системи здатні забезпечити єдине централізоване керування всіма процесами, повну інтеграцію з контролерами та, що не менш важливо можливість побудови надскладних систем автоматизації.

Байлі Д. зазначає у своїй роботі, що SCADA (системи диспетчерського контролю та збору даних) існує стільки ж, скільки й системи керування. Перші

системи SCADA використовували збір даних за допомогою панелей різноманітних вимірювальних приладів, а також освітлювальних приладів та стрічкових самописців [8].

До значущих переваг SCADA-систем можна віднести: високу надійність, підтримку значної кількості пристроїв, а також, що дуже важливо розвинуті засоби візуалізації та керування пристроями. Ці системи дуже широко використовуються у виробництві, енергетиці та критичній інфраструктурі.

Однак SCADA-системи також мають і ряд недоліків. До таких недоліків відносяться надвисока вартість впровадження та обслуговування таких систем, складність їх налаштування, а також велика залежність від єдиної централізованої архітектури, все це може призводити до сильних затримок у передачі даних і також створювати єдину точку відмови.

1.6.2 IoT-платформи

IoT-платформи є сучасним популярним підходом до реалізації моніторингу даних, такий підхід базується на використанні мережі підключених пристроїв та хмарних технологій. Ці платформи дозволяють ефективно та швидко збирати дані з гетерогенних сенсорів, потім передавати їх у хмару та виконувати детальний аналіз з використанням зданих до масштабування обчислювальних ресурсів.

У своїй книзі Еванс Д. пояснює, що Інтернет речей (або іншими словами IoT) являє собою наступний етап еволюції Інтернету загалом, це величезний стрибок у здатності збирати, аналізувати та розповсюджувати різноманітні дані, в подальшому можуть бути перетворенні на інформацію, знання та в кінці на мудрість. У цьому значенні IoT стає дуже важливим [9].

Перевагами таких IoT-платформ є їх гнучкість, здатність до масштабованості та можливість надшвидкого розгортання системи в будь - якому середовищі. Інтернет речей забезпечує максимально зручний доступ до різноманітних даних з

усіх точок світу, також підтримує інтеграцію з різними сервісами та дозволяє використовувати мультимодальні методи аналітики.

Але використання вищеописаних IoT-платформ має також і певні обмеження. Такими недоліками цих платформ є висока залежність від стабільного з'єднання з мережею, можливі великі затримки передачі даних, безпека та конфіденційність переданих даних. А також, що не менш важливо, передача дуже великих обсягів даних у єдину хмару може створювати додаткове непотрібне навантаження на мережу.

1.6.3 Системи збору телеметрії

Ще одним методом моніторингу є системи збору телеметрії. Вони в більшості випадків орієнтовані на отримання, збереження, а також майбутній аналіз саме технічних параметрів обладнання. Такий метод використовується для загального моніторингу стану систем, виявлення помилок та прогнозування майбутніх збоїв. Системи телеметрії мають два підходи: можуть працювати як у єдиному централізованому, так і в розподіленому середовищах.

Верма Н. у своєму посібнику зазначає, що загалом термін телеметрія походить від грецьких слів: «tele» та «metron», вони означають «віддалений» або «той, що знаходиться дуже далеко», та «вимірювання» відповідно. Зібравши до купи, телеметрія - процес виміру віддалених величин або певних параметрів [11].

До суттєвих переваг телеметричних систем належить можливість обробки надвеликих обсягів даних, підтримка мультимодальних форматів передачі, а також інтеграція з аналітичними інструментами. Ці системи дуже часто використовуються у поєднанні з вищеописаними IoT-рішеннями та іншими системами моніторингу інфраструктури.

Явними недоліками є велика складність у налаштуванні, також необхідність додаткової інфраструктури для того, щоб ці дані можна було зберігати, а також можливі затримки при єдиній централізованій обробці. У випадку, коли існує

велика кількість джерел даних можуть виникати зайві проблеми масштабованості та управління потоками інформації.

1.7 Аналіз недоліків існуючих рішень та постановка задачі

Попередньо проведений аналіз вже існуючих програмних рішень для такої задачі, як моніторинг та аналіз стану обладнання - показав, що, незважаючи на всі їх переваги та функціональні можливості, ці системи мають ряд певних суттєвих обмежень. Ці обмеження дуже ускладнюють використання таких рішень в даних умовах кіберенергетичних систем лабораторії.

Найбільшим недоліком для представленої задачі є явний централізований характер обробки інформації, що є притаманно більшості вищеописаних SCADA-систем та IoT-платформ. Дані системи проводять значну частину обчислень на віддалених далеких серверах або ж в хмарному середовищі. Це і призводить до непотрібних затримок у передачі та обробці. Максимально критично для подібних задач, які потребують тільки оперативного реагування на зміни стану обладнання.

Ще однією суттєвою проблемою таких систем є перевантаження мережі, що може виникати при передачі дуже великих обсягів даних від величезної кількості джерел до одного вузла переробки. У подібних лабораторних умовах, де використовується різне обладнання з дуже високою частотою збору інформації, це все може призводити до втрати великої кількості пакетів, також зниження швидкості передачі та максимально нестабільності роботи системи.

А також, описані вище вже існуючі рішення дуже часто мають недостатню масштабованість. Коли збільшується кількість пристроїв може виникнути необхідність суттєвого розширення серверної інфраструктури, це все в подальшому ускладнює впровадження системи та дуже підвищує витрати на підтримку її.

Дуже важливим аспектом є залежність від одного вузла. У випадку, якщо виникне його відмова або загальна втрата зв'язку - система моніторингу може

втратити свою функціональність. Це максимально знижує надійність кіберенергетичної системи загалом, особливо в таких умовах, де безперервність моніторингу є необхідною.

Додатково варто окреслити проблему неточності та сильної зашумленості даних. Цей недолік може виникати під час передавання або обробки інформації. Відсутність попередньої локальної обробки сильно ускладнює фільтрацію даних, які передаються і дуже негативно впливає на якість майбутніх прийнятих рішень.

Враховуючи всі зазначені вище недоліки, дуже актуальним є впровадження описаного вище підходу граничної обробки даних. Цей підхід передбачає виконання більшої частини обчислень прямо на рівні пристрою збору даних. Також цей підхід дозволяє сильно зменшити затримки, та знизити непотрібне навантаження на мережу, також підвищити надійність системи загалом.

У межах даної роботи ставиться важлива задача розробки системи з підходом граничної обробки даних для моніторингу стану обладнання кіберенергетичної лабораторії. Така система повинна чітко забезпечувати локальний збір та подальшу обробку даних на базі одноплатного мікрокомп'ютера Raspberry Pi, формування висновків щодо стану, а також в результаті виконувати передачу результатів кінцевому користувачу через мережевий інтерфейс.

Висновки до розділу 1

У першому розділі проведено загальний теоретичний аналіз предметної області кіберенергетичних систем, детально досліджено принципи їх побудови у сьогоденних енергетичних лабораторіях.

Розглянуто, як об'єкт дослідження - тепловий насос Mitsubishi Electric, проаналізовано головний принцип його роботи, найважливіші параметри та режими функціонування.

Також виконано огляд вже існуючих систем моніторингу, популярних SCADA-систем, IoT-платформ та систем збору телеметрії, визначено їх найголовніші переваги та недоліки.

Описано, що ці централізовані підходи мають великий ряд обмежень. У результаті глибокого проведеного аналізу обґрунтовано переваги використання технології граничних обчислень для обробки даних безпосередньо поруч з джерелом їх виникнення. Таким чином підвищується загальна швидкість реагування системи.

На основі даного дослідження сформульовано конкретну задачу розробки системи для проведення аналізу стану обладнання кіберенергетичної лабораторії засобами граничних обчислень. У наступному розділі детальніше описано саме засоби розробки для виконання поставленої задачі.

2 ЗАСОБИ РОЗРОБКИ СИСТЕМИ

У другому розділі цієї роботи розглянуто використані апаратні та програмні засоби розробки для даного продукту. Ці засоби розробки використовувалися для реалізації системи локальної обробки даних з лабораторного обладнання. Також дуже детально описано всі головні компоненти апаратної платформи, зокрема одноплатний мікрокомп'ютер Raspberry Pi, а також програмне забезпечення, саме воно забезпечує повноцінне функціонування системи. Велику увагу в розділі приділено вибору операційної системи (OS), головної мови програмування та важливих інструментів керування розробкою. Все це разом і формує сильну основу для побудови edge-рішення.

2.1 Апаратні засоби системи

Для того щоб реалізувати задуману в процесі дослідження систему граничної обробки отриманих з лабораторного обладнання даних використано широко відомий мікрокомп'ютер Raspberry Pi (версія 1.3 Wifi Zero).

Монк С. у своєму посібнику говорить, коли комп'ютер Raspberry Pi завантажується, ви отримуєте повноцінний робочий стіл Linux. Це реально справжній комп'ютер, здатний запускати звичайний офісний пакет, має можливості відтворення відео, ігри тощо. Це не Microsoft Windows; натомість це сильний конкурент Windows з відкритим кодом Linux (Debian Linux), а віконне середовище називається LXDE [10]. Саме він виконує головні функції, так як є центральним обчислювальним вузлом edge-рівня. Описаний пристрій повністю здатний забезпечити локальну обробку отриманих даних, також надає взаємодію з мережею та можливість поєднання з іншими компонентами системи моніторингу.

На рисунку 2.1 зображено зібрану систему та як налаштовано комп'ютер Raspberry Pi.



Рисунок 2.1 – Фізичне підключення Raspberry Pi Zero V1.3

Для побудови повністю функціональної апаратної частини описаної системи було використано наступні фізичні компоненти:

- одноплатний мікрокомп'ютер Raspberry Pi (версія 1.3 Wifi Zero - модель без вбудованого модуля Wi-Fi);
- карта пам'яті microSD обсягом 32 ГБ. Вона використовується для збереження операційної системи (ОС) та програмного забезпечення (ПЗ);
- адаптер живлення (micro-USB). Блок та кабель. Це надає стабільне електроживлення мікрокомп'ютера;
- USB OTG-кабель. Даний кабель використовується для підключення периферійних пристроїв – а в даному випадку Wi-Fi адаптер до мікрокомп'ютера;
- зовнішній Wi-Fi модуль (USB) використовується для підключення до бездротової мережі (Wi-Fi);

- ноутбук (персональний комп'ютер). Він використовується для початкового налаштування системи та подальшого керування нею.

Запуск комп'ютера Raspberry Pi здійснюється простим шляхом завантаження операційної системи з microSD карти. Попередньо обрано операційну систему було завантажено на карту. Мікрокомп'ютер не має вбудованого накопичувача, тому карта пам'яті в даному випадку є дуже важливим елементом системи.

Завдячуючи максимально компактним розмірам, наднизькому енергоспоживанню та наявності важливих для роботи вбудованих мережевих інтерфейсів Raspberry Pi є дуже хорошим рішенням для реалізації граничних обчислень у подібних системах моніторингу та аналізу стану технічного лабораторного обладнання.

2.2 Програмні засоби розробки

У цьому підпункті нижче детальніше описано саме ПЗ, яке використовувалось при розробці програмного продукту для поставленої задачі. Зроблено обґрунтований повний аналіз вибору мови програмування та додаткових інструментів для впровадження системи.

2.2.1 Операційна система та налаштування мережі

Для реалізації програмної частини системи було використано популярну операційну систему Raspberry Pi OS (32-bit). Вона є гарно оптимізованою для роботи на мікрокомп'ютерах Raspberry Pi. Встановлення такої операційної системи здійснювалося просто за допомогою утиліти Raspberry Pi Imager. Ця утиліта дозволяє швидко записати образ системи на microSD карту пам'яті.

Гайсов у своєму посібнику пояснює, що окрім офіційної ОС, існує також багато інших операційних систем для мікрокомп'ютера Raspberry Pi. Наприклад Raspbian був розроблений Майком Томпсоном та Пітером Гріном на базі дистрибутива Linux Debian. Перша початкова версія випущена в 2012 року, для

легкого встановлення доступно більш ніж 5000 пакетів [15].

Також, під час налаштування системи попередньо задано параметри для підключення до бездротової мережі Wi-Fi: ім'я хоста пристрою та облікові дані користувача. Це все дозволило миттєво після запуску пристрою (мікрокомп'ютера) забезпечити його видимість та доступність у локальній мережі без якоїсь додаткової конфігурації.

2.2.2 Засоби віддаленого доступу

Для віддаленого керування даним мікрокомп'ютером використано популярний протокол SSH (Secure Shell). Такий протокол здатний забезпечити повну автентифікацію, шифрування та цілісність даних. У офіційній інструкції розробників можна знайти таку інформацію, що реалізація Secure Shell пропонує наступні можливості: безпечну командну оболонку, безпечну передачу усіх файлів та віддалений доступ до інших різноманітних програм TCP/IP через безпечний тунель [12].

Завдяки використанню саме SSH стало можливим здійснювати повноцінне налаштування та керування мікрокомп'ютером Raspberry Pi без підключення до монітора, клавіатури або миші. Підключення повністю виконувалося з одного персонального комп'ютера через закриту локальну мережу за IP-адресою, призначеною роутером.

2.2.3 Мова програмування

Цільовою мовою програмування для реалізації всіх алгоритмів збору та обробки даних обрано популярний Python. МакКінні В. говорить у своїй книзі, що з моменту своєї появи в 1991 році, мова Python стала однією з найкращих інтерпретованих мов програмування, поряд з такими як Perl, Ruby та іншими. Python та Ruby стали особливо популярними в 2005 році, використовувались для створення красивих веб-сайтів з використанням їхніх могутніх та численних веб-

фреймворків, таких як Rails (Ruby) та Django (Python) [13].

Такий вибір є очевидним та обумовлений надпростим синтаксисом, також дуже широкими можливостями та наявністю численної кількості різноманітних бібліотек для роботи з мережею, обробки отриманих даних та взаємодії загалом зі всіма апаратними компонентами.

2.2.4 Система контролю версій

Для організації розробки такого програмного забезпечення та управління всіма доданими змінами у коді використано найпопулярнішу систему контролю версій Git. Чакон С. пише у своєму підручнику, що система контролю версій - це така система, що записує зміни у файл або якийсь набір файлів протягом певного часу, так що кожен зможете повернутися до певної версії пізніше [14].

Використання саме Git дозволило легко синхронізувати програмний код між персональним комп'ютером та одноплатним мікрокомп'ютером Raspberry Pi через створений віддалений репозиторій GitHub. Це все забезпечує зручність розробки, можливість відстеження усіх змін та сильно спрощує командну роботу над проєктом.

2.3 Інтеграція апаратної та програмної частин системи

У розробленій в межах даної роботи системі, Raspberry Pi виступає центральним вузлом. Цей вузол поєднує апаратну та програмну частини в єдину так звану edge-архітектуру. Як описувалось вище, завантаження Raspberry Pi OS та програмного забезпечення здійснюється з карти пам'яті. Після подачі живлення до мікрокомп'ютера через micro-USB кабель автоматично запускається все що потрібно для роботи: операційна система, мережеві служби та всялякі необхідні програмні модулі обробки даних.

Саме підключення Raspberry Pi до доступної локальної Wi-Fi мережі легко реалізовано за допомогою зовнішнього USB Wi-Fi адаптера. Він під'єднаний через

OTG-кабель. Це дає змогу мати віддалений доступ до системи з обраного персонального комп'ютера через SSH-з'єднання для того, щоб налаштовувати та керувати пристроєм.

Якщо говорити про програмну частину, то вона реалізована на мові Python. В основному виконує обробку, аналіз та локальне збереження усіх даних. Усі головні обчислення виконуються повністю на маленькому Raspberry Pi. Це, як описувалось вище - зменшує навантаження на мережу та забезпечує надшвидку обробку інформації.

Усі отримані результати пізніше можуть передаватися іншим модулям системи та використовуватись для майбутнього аналізу. Таким чином, можна дійти до висновку, що усі апаратні компоненти забезпечують повноцінне функціонування та гарну мережеву взаємодію системи, а не менш важлива програмна частина реалізує загалом логіку обробки даних і керує всіма процесами.

Висновки до розділу 2

У другому розділі повністю обґрунтовано доцільність вибору усіх апаратних та програмних засобів для повноцінної реалізації системи. Загалом використання одноплатного мікрокомп'ютера Raspberry Pi дозволяє покрити всю локальну обробку даних методом граничних обчислень завдяки його компактності, легкому впровадженню, низькому енергоспоживанню та мережевим інтерфейсам.

Доведено, що обрана в процесі розробки операційна система Raspberry Pi OS повністю забезпечує постійну безперебійну роботу пристрою та легкість налаштування, а протокол SSH дозволяє дуже ефективно здійснювати керування на відстані. Використання такої мови програмування, як Python забезпечує гнучкість реалізації усіх алгоритмів обробки та аналізу даних саме завдяки широкому набору бібліотек. Застосування популярної системи контролю версій Git дозволяє легко організувати процес розробки, управління змінами.

У наступному розділі детальніше описано архітектуру системи.

3 ОПИС АРХІТЕКТУРИ СИСТЕМИ

В наступному розділі здійснено опис архітектури всієї системи загалом та представлено UML діаграми: діаграму прецедентів, діаграму послідовностей та діаграму компонентів.

3.1 Загальна архітектура

Суцільна система має багаторівневу складну архітектуру. Вона складається з трьох повноцінних, основних підсистем, кожна з яких має велике значення. Ці три підсистеми описуються в трьох різних дипломних роботах. (рисунок 3.1)

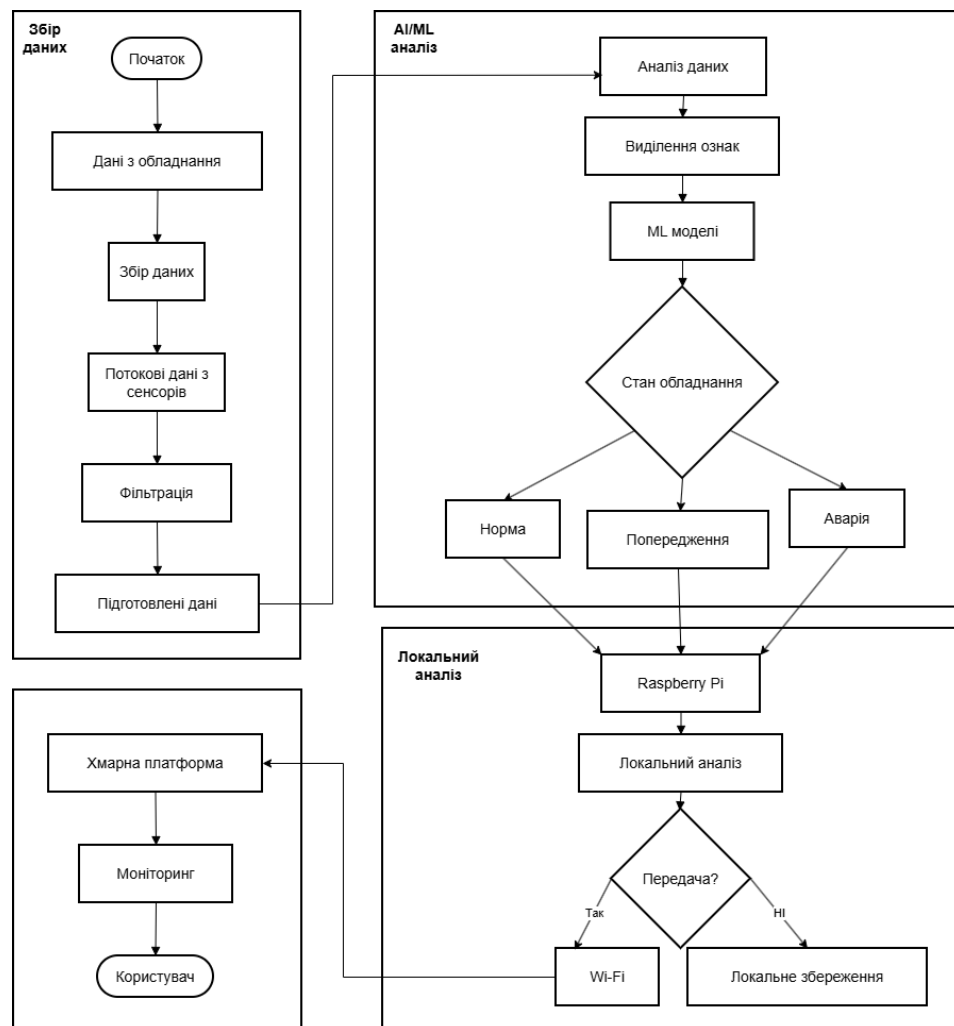


Рисунок 3.1 – Алгоритм функціонування системи

На першому рівні (першій підсистемі) здійснюється збір сирих, необроблених даних з обладнання кіберенергетичної лабораторії. У цьому випадку з теплового насоса, в майбутньому є можливість отримувати дані з інших пристроїв також. Цей рівень містить датчики температури, витрати енергоспоживання, які формують первинні дані. Уся зібрана з датчиків інформація проходить початкову обробку. В цю обробку включається: фільтрацію шумів, нормалізацію та перевірку коректності значень перед передачею далі.

Другий не менш важливий рівень відповідає за формування знань та обробку інструкції до приладів з використанням методів інтелектуального аналізу. На цьому етапі відбувається наступне: виділяються ознаки, формується статистичний аналіз та застосовуються алгоритми класифікації. В підсумку маємо сформовану онтологічну модель.

Третій рівень реалізує саме вищеописану граничну обробку даних. Як зазначалось це дозволяє виконувати аналіз стану лабораторного обладнання прямо поблизу джерела даних. В даній підсистемі використовується одноплатний мікрокомп'ютер Raspberry Pi, який відповідальний за здійснення локальної обробки та майбутнє прийняття рішень та, за необхідності, передачу інформації про виявлені збої через бездротову мережу. Саме такий підхід дозволяє максимально зменшити можливі затримки обробки та підвищує надійність системи загалом.

Усі отримані в ході описаного процесу результати передаються кінцевому користувачу у вигляді інтерфейсу для моніторингу. Такий інтерфейс може бути реалізований, наприклад у вигляді веб-застосунку або певного дашборду, що відображає саме поточний стан системи, певні історичні дані та важливі повідомлення про усі можливі відхилення в роботі лабораторного обладнання. (рисунок 3.1)

3.1 Діаграма прецедентів

Діаграма прецедентів (Use Case diagram) використовується для простого відображення усіх основних сценаріїв взаємодії користувача та інших компонентів із системою моніторингу кіберенергетичного лабораторного обладнання. (рисунк 3.2)

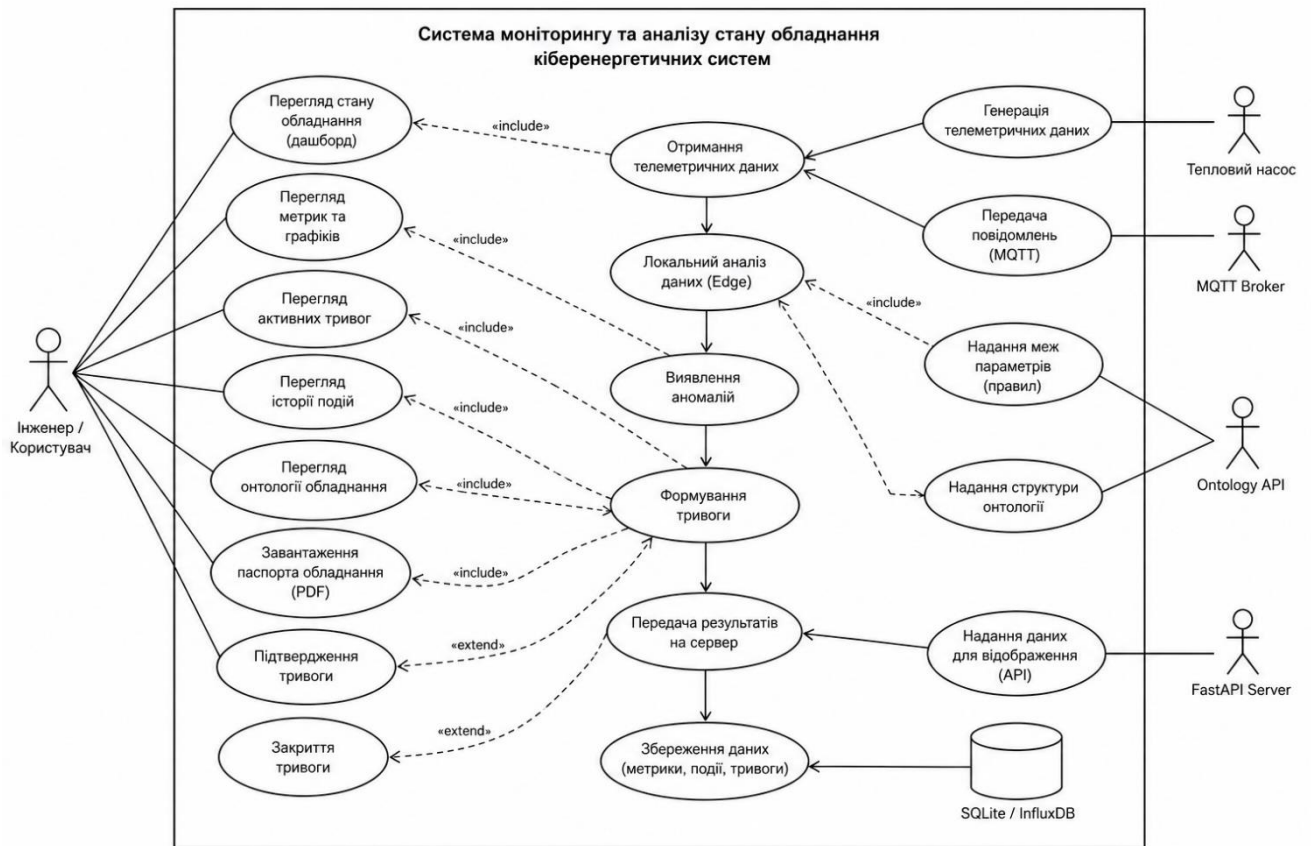


Рисунок 3.2 – Діаграма прецедентів системи EnergyLab

Головним актором (учасником) системи є оператор (інженер), який взаємодіє з веб-інтерфейсом EnergyLab для перегляду поточного стану лабораторного обладнання, аналізу всіх метрик, роботи з отриманими тривогами та перегляду онтології с-пристроїв. Кінцевий користувач також має можливість переглядати історію подій - які тривоги вже було вирішено, підтверджувати або закривати тривоги, а також завантажувати технічні паспорти всього обладнання для того, щоб автоматично сформувати онтологічні дані.

Додатковими учасниками системи виступають MQTT-брокер, Ontology API та edge-аналітика на Raspberry Pi. MQTT-брокер здатний забезпечити повноцінну передачу телеметричних даних від обладнання, Ontology API надає допустимі межі параметрів та зв'язки між усіма компонентами системи EnergyLab, а edge-analyzer виконує важливий локальний аналіз стану обладнання та формує повідомлення про аномалії (збої). Таким чином, дана діаграма демонструє взаємозв'язок між кінцевим користувачем EnergyLab, обладнанням та програмними компонентами.

3.2 Діаграма послідовностей

Діаграма послідовностей (Sequence diagram) описує весь процес обробки телеметричних даних у даній системі.

Нікіфорова О. у своїй роботі пояснює, що Sequence diagram UML є одним з дуже добре відомих артефактів UML, що зазвичай використовуються для представлення динамічних характеристик спроектованої системи. Така діаграма зосереджена на визначенні якоїсь взаємодії об'єктів у правильній підібраній послідовності [16].

Робота всієї системи починається з того, що тепловий насос надсилає телеметричні параметри через протокол MQTT. Після цього локальний аналізатор, який працює на Raspberry Pi, отримує ці дані та виконує їхню локальну обробку.

На наступному етапі аналізатор звертається до онтології для отримання усіх допустимих меж параметрів лабораторного обладнання та пов'язаних правил. Після отримання таких даних виконується rule-based перевірка та аналіз за допомогою моделі штучного інтелекту. Якщо система помічає якісь відхилення від норми роботи, формується тривога щодо цього із зазначенням типу проблеми та діагностичної інформації для її вирішення. (рисунок 3.3)

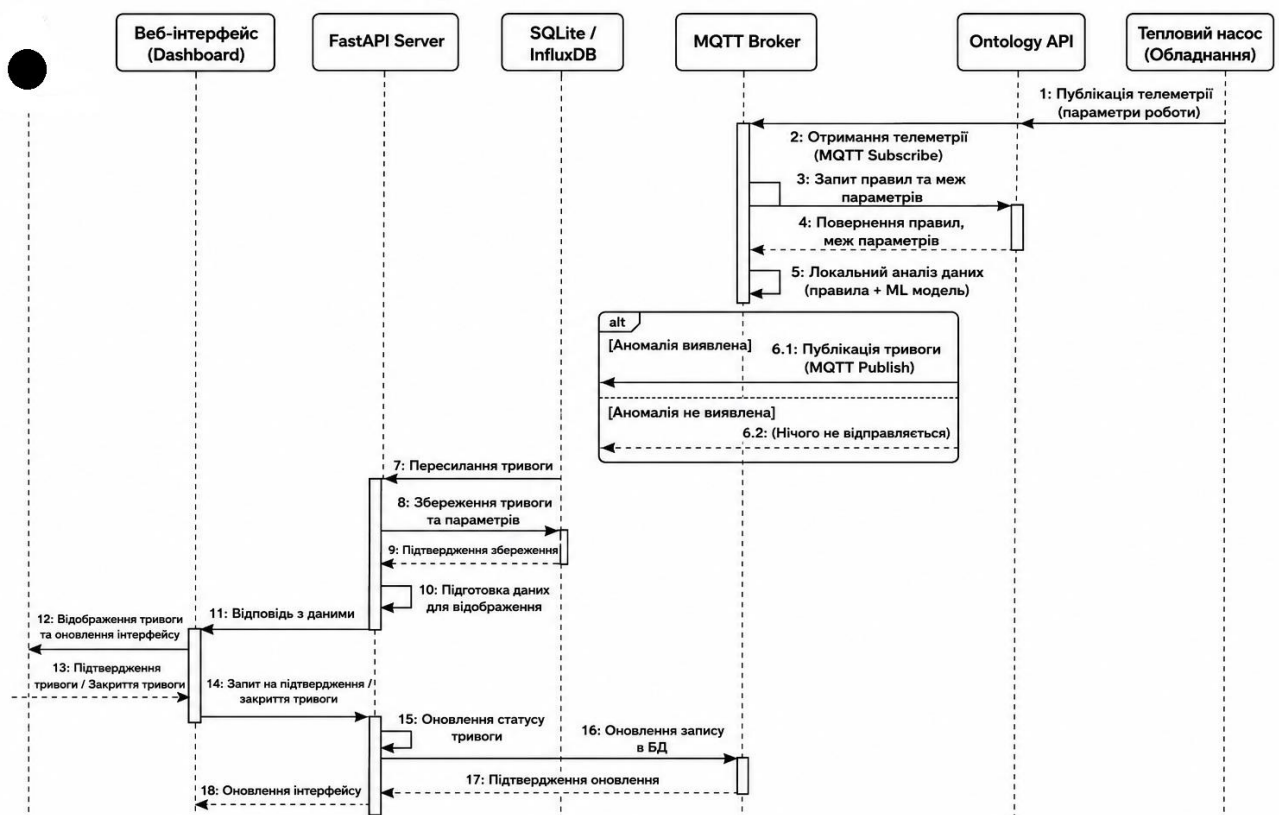


Рисунок 3.3 – Діаграма послідовностей системи EnergyLab

Далі локальний аналізатор передає оформлену тривогу через HTTP-запит до сервера. Сервер зберігає цю подію у базі даних SQLite та оновлює поточний стан обладнання. Останнім етапом цього процесу - веб-інтерфейс отримує актуальну інформацію через REST API та відображає усі знайдені тривоги, графіки й стан обладнання кінцевому користувачу у режимі реального часу.

Висновки до розділу 3

Спроековано та описано архітектуру системи EnergyLab. Проведено моделювання загальної структури програмного забезпечення за допомогою популярного підходу - UML-діаграм, зокрема діаграми прецедентів, діаграми послідовностей та діаграми компонентів.

Визначено основні програмні модулі, досліджено як відбувається їх взаємодія та описано загальний процес обробки телеметричних даних. Велику

увагу приділено реалізації граничних обчислень на базі Raspberry Pi. Саме це дозволяє легко виконувати локальну обробку певної інформації, зменшувати затримки передачі отриманих даних та забезпечувати автономність роботи системи загалом. Також у розділі детальніше описано використання MQTT протоколу для передачі телеметрії, FastAPI бібліотеки для реалізації серверної частини, SQLite та InfluxDB баз даних для зберігання даних, а також Ontology API для роботи зі самою структурою знань та семантичними зв'язками між різноманітними компонентами обладнання.

Усі отримані результати остаточно підтвердили доцільність використання такої багаторівневої архітектури та технології граничних обчислень для побудови подібних систем моніторингу кіберенергетичного обладнання.

У наступному розділі детальніше описано саму реалізацію даної спроектованої системи.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У даному розділі детальніше описано саме програмну реалізацію представленої системи. Надано опис використаних бібліотек та обґрунтовано доцільністю такого вибору.

4.1 Реалізація головного модуля аналізу стану обладнання

Центральним компонентом цієї програмної частини системи виступає модуль аналізу стану лабораторного обладнання. Даний модуль реалізований у папці `analyzer`. Саме цей модуль головний за прийняття телеметричних даних від справжнього обладнання, їх локальну обробку на рівні джерела поблизу та формування підсумкового висновку про загальний стан пристрою.

Особлива увага в кінцевій реалізації приділяється можливості роботи програми на дуже малопотужних edge-пристроях, зокрема Raspberry Pi Zero V1.3. Це сильно повпливало на зміну архітектури в процесі розробки та обрання такого підходу до використання інструментів машинного навчання.

Основна ідея такої системи полягає у тому, щоб виконувати повноцінний аналіз безпосередньо на пристрої поблизу джерела, а не передавати весь величезний потік даних на єдиний центральний сервер. Також обраний такий підхід дозволяє значно зменшити будь-які затримки, та сильно знизити навантаження на єдину мережу та забезпечити безперебійну роботу системи навіть у випадку дуже нестабільного з'єднання із серверною інфраструктурою.

Загалом аналізатор локально отримує метрики через протокол MQTT, далі виконує перевірку усіх параметрів, потім звертається до онтологічного API для отримання очікуваних меж цих параметрів та формує кінцевий результат аналізу у режимі реального часу, це відбувається кожні 5 секунд. Після цього система передає весь вже готовий висновок у вигляді певного стану `normal`, `warning` або `anomaly`.

4.1.1 Реалізація ML-модуля

У представленій реалізації ключовим файлом є `pi_zero_analyzer.py`. Цей файл спеціально адаптований для взятого Raspberry Pi Zero V1.3, який має лише один ARM11-процесор та 512 MB оперативної пам'яті. Через такі суттєві апаратні обмеження на представленою пристрої неможливо ефективно використовувати повноцінні бібліотеки TensorFlow або TensorFlow Lite.

Саме тому у даній роботі було прийнято рішення перейти саме до такого підходу як використання lightweight-підходу - запуску автоенкодера на базі чистого NumPy. В офіційній документації до бібліотеки зазначається, що NumPy - це дуже важливий пакет для всіляких наукових обчислень на Python. Це відома бібліотека Python, яка надає багатовимірний масив, також різні похідні об'єкти (такі як масковані масиви та матриці) та набір різноманітних процедур для швидких операцій з масивами [17]. Такий обраний підхід дозволяє повністю відмовитися від якихось важких ML-бібліотек на рівні граничних обчислень та залишити тільки мінімально необхідні залежності.

Автоенкодер у представленій системі має надкомпактну архітектуру та містить невелику кількість тільки необхідних параметрів. Завдяки вищеописаному факту всі математичні операції можуть бути виконані звичайними матричними обчисленнями бібліотеки NumPy без використання GPU чи якихось інших спеціалізованих ML-фреймворків. Це все є важливими перевагами системи, оскільки такий Raspberry Pi Zero здатний виконувати детальний чіткий аналіз практично миттєво навіть при своїх дуже обмежених ресурсах. Фактично така модель працює як дуже легкий механізм, який взагалі не потребує надскладного середовища виконання.

У процесі розробки дана модель навчається ніби окремо на додатковій машині за допомогою бібліотеки TensorFlow. Після завершення всього навчання зберігаються лише отримані ваги нейромережі у файл `anomaly_ae.npz`. На локальний пристрій переноситься тільки отриманий в процесі файл та сам модуль, який виконує аналіз. У висновку, мікрокомп'ютер Raspberry Pi не виконує весь

процес навчання моделі, а просто використовує вже готові надані параметри для надання висновку. Це дозволяє максимально мінімізувати навантаження на локальному рівні та зробити загалом систему придатною навіть для критично слабких платформ.

4.1.2 Використані бібліотеки та інструменти

Під час роботи над аналізатором використовувалось багато сучасних інструментів.

Наприклад, протокол MQTT використовується як основний для проведення обміну телеметрією між всіма підсистемами. Перевага згаданого протоколу полягає у його легковаговості та дуже маленькому мережевому навантаженні, що критично важливо для подібних IoT та edge-систем. Саме за допомогою цього протоколу аналізатор отримує необхідні метрики обладнання та публікує фінальні результати аналізу.

Для правильного зберігання та передачі повністю структурованих даних використовуються Pydantic-схеми. На офіційному сайті бібліотеки можна довідатись, що Pydantic - це бібліотека Python для перевірки та трансформації всіляких даних. Вона повністю ґрунтується на анотаціях і дозволяє розробникам ефективно перевіряти, чи відповідають усі вхідні дані визначеним типам і правилам [18]. Pydantic-схеми в даному випадку забезпечують повну валідацію повідомлень, контроль усіх типів даних та сильно зменшують ризик помилок під час роботи з певними JSON-повідомленнями. Завдяки цьому розроблена система може повністю безпечно обробляти повідомлення навіть коли існують нестабільні або частково некоректні дані.

Не менш важливим компонентом описаної системи є онтологічний API. Аналізатор в даному випадку не містить ніяких жорстко зашитих нормативних значень у самому коді. Замість цього усі межі параметрів запитуються через додатковий інструмент - REST API у окремої створеної онтологічної підсистеми.

Це надає можливість централізовано змінювати усі характеристики обладнання без додаткової модифікації коду. Ця система може отримати допустимі значення СОР, конкретних температур або потужності для обраного пристрою через спеціальні запити.

Існує можливість надмірного навантаження на API. Для уникнення цього у модулі спеціально реалізовано локальний кеш для меж параметрів. Всі отримані в результаті значення зберігаються протягом певного визначеного часу, після чого мають оновитися повторно. Такий обраний підхід дозволяє в сукупності забезпечити актуальність даних й високу швидкодію представленої системи.

4.1.3 Модуль діагностики та онтологічного аналізу

У цій архітектурі надважливу роль відіграє окремий маленький модуль `diagnose.py`. Основна задача цього модуля - не лише просто виявити аномалію, а й пояснити причину її виникнення. Таким чином можна перейти від простого детектування якоїсь проблеми до максимально інтелектуальної діагностики лабораторного обладнання.

Модуль `DiagnosticEngine` повністю працює із онтологічним контекстом певного пристрою та формує чіткі пояснення для кожної знайденої аномалії. Загалом аналізатор отримує інформацію про можливі помилки, коди цих помилок та сервісні рекомендації через `endpoint`.

Після цього етапу система зіставляє коди аномалій з наданими онтологічними правилами та формує чіткий структурований список діагнозів. Можна уявити так, якщо спостерігається перевищення температури обладнання або зниження ефективності роботи, система може не просто зафіксувати аномалію, а й надати ймовірну причину такої несправності та рекомендовану дію для спеціаліста.

Весь цей описаний підхід є набагато ефективнішим за класичний `rule-based` аналіз, оскільки вся система переходить від просто контролю меж до вже

семантичної інтерпретації стану конкретного пристрою. Це дуже важливо для подібних кіберенергетичних систем, де одна й та сама певна аномалія може мати абсолютно різні причини виникнення залежно від типу пристрою.

4.1.4 Комбінований аналіз: ML та rule-based підхід

Rule-based перевірки у наданій системі є дуже важливою складовою всього аналізу. Вони повністю реалізовані у функції `rule_based_checks()` та використовуються для дуже швидкого виявлення критичних відхилень. У описаному коді перевіряються можливі перевищення потужності, недопустимі температури, а також зниження коефіцієнта ефективності COP. Перевага саме такого підходу в тому, що він прозорий та зрозумілий для багатьох інженера. Кожне таке спрацювання супроводжується зрозумілими текстовим поясненням.

На відміну від стандартних rule-based перевірок, ML-модель може дозволити знайти будь-яку нетипову поведінку пристрою навіть тоді, коли параметри ще і не вийшли за допустимі надані межі. А саме поєднання цих двох базових підходів є головною перевагою системи загалом. Описаний rule-based механізм може забезпечити надійний контроль найбільш критичних параметрів, а такий підхід, як ML-модель дозволяє легко виявляти приховані або певні нестандартні відхилення.

Під час такого аналізу формується вектор конкретних ознак:

- потужність в кВт;
- різниця температур подачі та зворотки;
- температура подачі теплоносія;
- зовнішня температура на вулиці;
- ефективність роботи теплового насосу.

На основі вищеописаного сформованого вектора. Інструмент автоенкодер виконує можна сказати реконструкцію даних та обчислює середню похибку (MSE). Якщо помилка перевищує поріг (threshold), система може вважати поведінку

нетипічною - аномальною. Таким чином, модель точно оцінює, наскільки поточний стан відрізняється від звичайного режиму роботи.

Кінцевий стан системи загалом формується шляхом об'єднання результатів обох ML-аналізу та rule-based перевірок. Якщо спрацьовули правила та одночасно ML-модель визначає дуже нетипову поведінку, формується стан аномальний. Якщо спрацьовує тільки один із механізмів, система повертає просто попередження. А у випадку повної відсутності відхилень формується стан нормальний.

Після завершення всього аналізу формується спеціальний об'єкт StateMessage. Такий об'єкт містить всю важливу інформацію про пристрій, список усіх знайдених аномалій. Результат потім публікується назад у так званий MQTT-топик, що дозволяє іншим компонентам усієї системи працювати вже не із сирими метриками, а з готовими наданими висновками.

4.2 Сервер тривоги та моніторингу системи

Сервер тривоги - завершальний компонент програмної частини представленої системи - написаний у файлі alerts_server.py. Він відповідає за отримання кінцевих результатів від модуля analyzer, та потім зберігає їх у базі даних SQLite і надає готовий веб-інтерфейс для оператора. Сервер тривоги забезпечує єдиний централізований моніторинг цих даних, історію подій і взаємодію інженера з результатами роботи пристроїв.

У даній реалізації він побудований за допомогою інструмента FastAPI, таким чином забезпечується значно вища продуктивність та зручність роботи з підходом REST API порівняно з класичним Flask.

Використання такої бази даних, як SQLite у даній системі є максимально гарним рішенням цієї задачі. Сервер тривоги не потребує надскладної розподіленої СУБД. Бд SQLite дозволяє зберігати всю історію тривоги та поточний стан обладнання лабораторії з мінімальними вимогами до використовуваних ресурсів.

Це все, у підсумку, робить систему дуже легкою у розгортанні та придатною для простих локальних лабораторних середовищ.

Сервер тривоги також працює з такою базою даних, як InfluxDB. Вона використовується для зберігання всіх історичних метрик та майбутньої побудови графіків. Такий підхід дуже легко дозволяє розділити оперативні події та всі часові ряди.

База даних SQLite використовується для легкої та максимально зручної роботи з отриманими тривогами та статусами, тоді як додатково база даних InfluxDB спеціально оптимізована саме для роботи з часовими даними.

Загалом веб-інтерфейс системи (основний дашборд EnergyLab) повністю побудований на відомих технологіях FastAPI та Jinja2. Інженер (оператор системи) може швидко переглядати усі активні тривоги, стан пристроїв лабораторії, історію подій та усі графіки пов'язані зі змінами параметрів у часі. Додатково для цієї наукової роботи реалізовано API. Дозволяє підтвердити або закрити тривогу, що дозволяє легко інтегрувати усю систему з іншими зовнішніми сервісами.

Висновки до розділу 4

У четвертому розділі розглянуто саме реалізацію всієї системи аналізу стану лабораторного обладнання, побудованої спеціально для роботи на пристрої Raspberry Pi Zero V1.3. Основна увага в реалізації приділялась створенню легкої архітектури, яка здатна легко виконувати локальний аналіз даних (телеметрії) без використання дуже важких ML-фреймворків.

Реалізовано розумний комбінований підхід до аналізу. Це поєднання rule-based перевірки, надлегкої ML-моделі та онтологічної діагностики. Також який підхід дозволяє не просто виявляти відхилення, а й сформувані структуровані пояснення та рекомендації для інженера (оператора).

Також дуже важливою перевагою цієї системи є використання відомого NumPy-автоенкодера замість важкого TensorFlow на рівні граничних обчислень. Це

дозволило максимально зменшити вимоги до використаних ресурсів та забезпечити повноцінну роботу системи EnergyLab навіть на Raspberry Pi Zero.

Основним продуктом виступає - сервер тривоги на FastAPI, він має інтеграцію з SQLite та InfluxDB, а також веб-інтерфейс для слідкування за станом обладнання. У результаті всієї роботи створено повноцінну систему моніторингу та аналізу кіберенергетичного обладнання за допомогою граничних обчислень. Вона повністю забезпечує дуже швидко локальну обробку усіх даних, зменшення непотрібного мережевого навантаження та максимальне підвищення автономності роботи системи загалом.

5 ІНТЕРФЕЙС ПРОГРАМНОГО ПРОДУКТУ

В розділі нижче представленої роботи описано взаємодію, яка буде відбуватись між кінцевим користувачем та продуктом, а саме дашбордом системи EnergyLab. Дашборд забезпечує зручний інтерфейс для перегляду стану обладнання в реальному часі, відображає активні тривоги, статистику роботи системи та основні параметри теплових насосів. Також має сторінки, де показуються: зібрані метрики, онтологічні графи. Існує ще сторінка для завантаження інструкції певного обладнання для подальшого створення онтологічної моделі.

5.1 Головна сторінка дашборду EnergyLab

Для дуже зручної взаємодії інженера (оператора) з системою спеціально реалізовано веб-інтерфейс для моніторингу під назвою EnergyLab. Він забезпечує потрібний централізований перегляд стану всього необхідного обладнання, отриманих активних тривог, історії подій.

Головна сторінка представленого дашборду містить верхню зручну навігаційну панель, у ній розміщено логотип з назвою системи EnergyLab, а також доступні основні розділи інтерфейсу, а саме: «Огляд», «Метрики», «Онтологія» та «Завантаження паспортів». Така чітка структура дозволяє інженеру (кінцевому користувачу) дуже швидко переходити між різними функціональними модулями системи EnergyLab. У правій частині даної панелі відображається індикатор, який показує, як часто оновлюються дані та поточний час. Ці всі індикатори є важливими для користувача, оскільки дозволяють легко контролювати достовірність та актуальність інформації та загалом переконатися, що система продовжує постійно отримувати нові дані від локального пристрою. (рисунок 5.1)

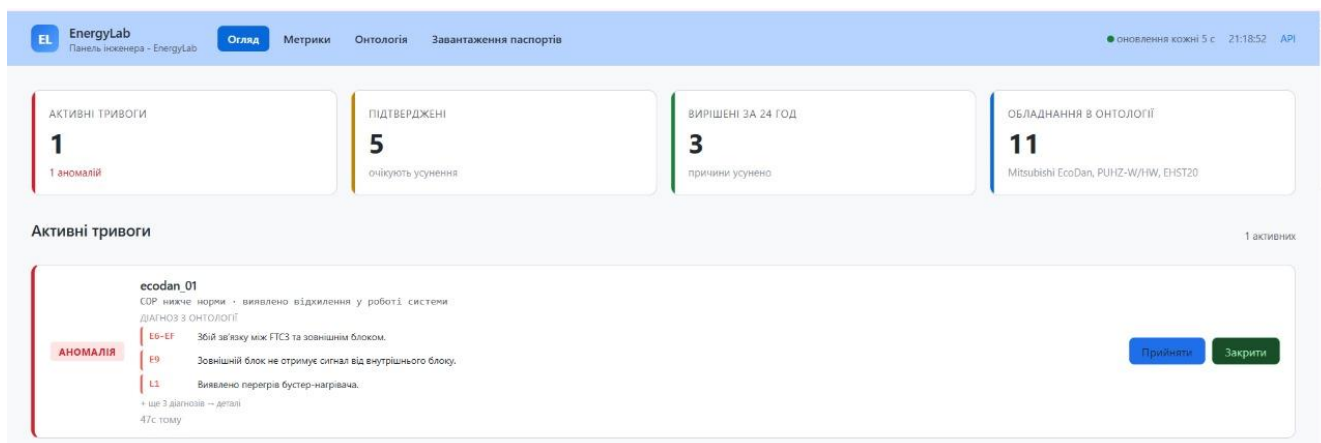


Рисунок 5.1 – Головна сторінка дашборду

У верхній частині дашборду після навігаційної панелі розташовано спеціальні інформаційні картки зі статистикою системи загалом.

Перша картка показує загалом кількість відкритих активних тривог, друга - кількість тривог, які були підтверджені оператором, третя - кількість усіх усунених оператором проблем за останню добу (24 години), а четверта - кількість всього обладнання в онтології.

Для кожного такого блока використовується окремий яскравий кольоровий індикатор: червоний (коір небезпеки) для аварійних станів, жовтий для попереджень, зелений для всіх успішно усунутих (вирішених) ситуацій та блакитний для інформаційних даних. Така обрана кольорова диференціація дозволяє кінцевому користувачу дуже легко оцінити стан без якоїсь необхідності детально аналізувати кожен окремий показник.

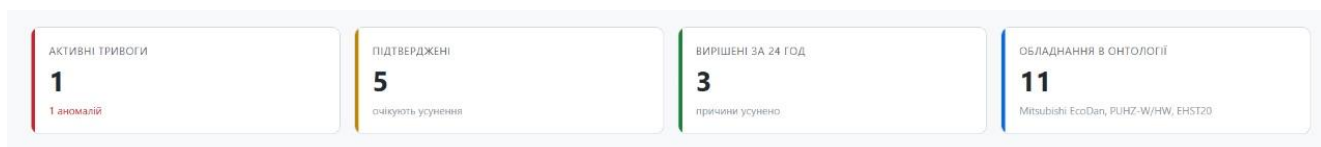


Рисунок 5.2 – Інформаційні картки

Особливу велику увагу приділено блоку під назвою «Активні тривоги». Тут відображаються усі критичні або попереджувальні ситуації, які сформовані

локальним аналізатором. Для кожної отриманої тривоги показується спеціальний ідентифікатор пристрою, тип аномалії, яку було знайдено, короткий опис проблеми загалом та результати онтологічної діагностики. Наприклад, у кіберенергетичній системі може бути виявлено значне зниження коефіцієнта ефективності COP або сильні порушення зв'язку між різними блоками теплового насоса - внутрішнім та зовнішнім.

Дуже важливою особливістю інтерфейсу для роботи оператора є те, що система EnergyLab не просто повідомляє про факт, що з'явилась аномалія, а й чітко відображає усі можливі причини такої несправності. На екрані можна побачити коди помилок, які спеціально отримані з доступної онтологічної бази знань, наприклад такі коди, як E6-EF, а також широкі та доступні текстові пояснення до них. Це сильно спрощує роботу інженера (оператора), оскільки дозволяє максимально швидко зрозуміти причину, яка викликала проблема без необхідності окремо розбирати технічну документацію виробника конкретного лабораторного обладнання.

Для кожної отриманої тривоги доступні кнопки «Прийняти» та «Закрити» проблему. Кнопка підтвердження «Прийняти» дозволяє користувачу швидко зафіксувати факт обробки певної проблеми, а кнопка закриття «Закрити» використовується після вже усунення несправності. Такий спроектований механізм є дуже важливим для організації всієї роботи персоналу та зручного ведення журналу усіх подій у системі моніторингу та аналізу стану лабораторного обладнання.

Нижче розташовано невеличкий блок «Стан обладнання». Такий блок містить окремі картки для кожного пристрою, який зареєстровано в системі. У картці зручно відображається наступні характеристики обладнання: назва конкретного пристрою, його модель, стан, який система отримує на даний момент часу та останні отримані метрики з локального аналізатора. Для активного пристрою (ввімкнений пристрій) система показує такі метрики: потужність,

температуру подачі, температуру зворотки, коефіцієнт ефективності COP та режим роботи. (рисунок 5.3)

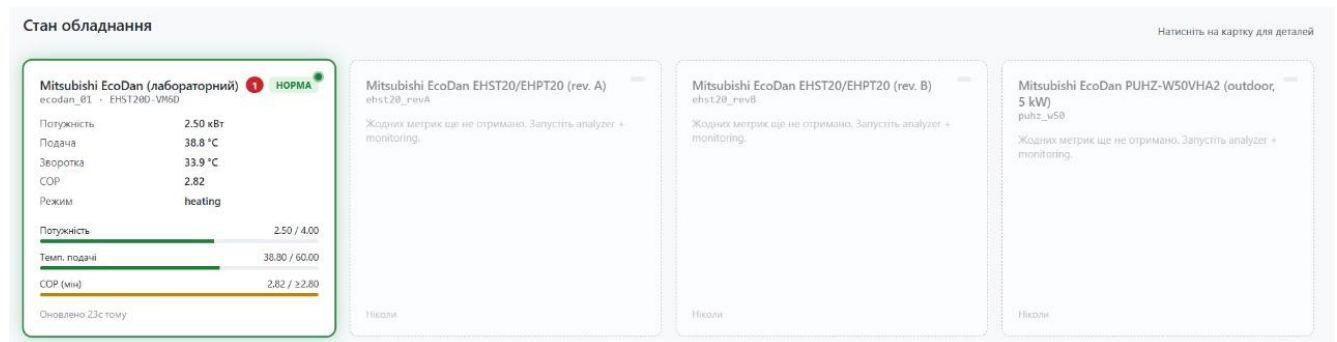


Рисунок 5.3 – Стан обладнання

Для візуалізації усіх параметрів у зручній формі використовуються горизонтальні індикатори. Наприклад, окремо зручно відображається співвідношення поточної потужності та максимально допустимої. Це все дозволяє оператору дуже швидко оцінити наближення кіберенергетичної системи до критичних режимів роботи.

Такі картки лабораторного обладнання також мають яскраві кольорові статуси. Зелений колір, як завжди означає саме нормальний режим роботи, жовтий у свою чергу виконує попереджувальну функцію, а червоний, як завжди означає якусь серйозну проблему - аномалію. Така продумана система індикації забезпечує максимально швидке візуальне сприйняття всього стану дуже великої кількості пристроїв. У випадку, коли дані від конкретного обладнання не надходять, розумна система показує чітке повідомлення про відсутність метрик. Це дозволяє максимально оперативно виявляти усі проблеми пов'язані зі зв'язком або зупинкою локального аналізатора.

Окремим не менш важливим елементом всього інтерфейсу є блок «Історія подій». У ньому повинні зберігатися всі попередньо знайдені тривоги та дії оператора, наприклад закриття проблеми. Для кожного такого запису відображається чіткий час виникнення події, описано тип проблеми, стан обробки

та діагностична інформація. Завдяки такому детальному опису інженер може легко аналізувати повторювані несправності, які часто виникають та легко відстежувати динаміку роботи всього обладнання та оцінювати якість обслуговування наданої системи. (рисунок 5.4)

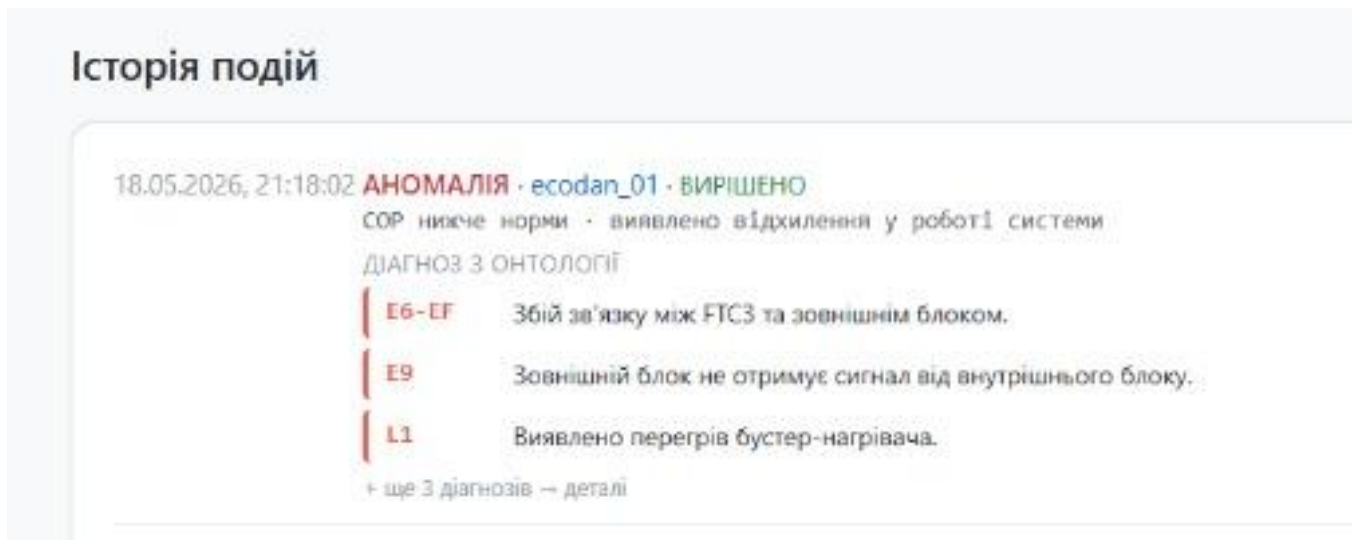


Рисунок 5.4 – Історія подій

Не менш важливою перевагою даного реалізованого інтерфейсу є зручна інтеграція з розробленою онтологічною підсистемою.

Система працює так, що автоматично пов'язує коди аномалії з онтологічними знаннями та здатно відобразити помилку та можливі причини, рекомендовані дії по усуненню. Отже, веб-інтерфейс має наступні можливості: функцію моніторингу, функцію підтримки прийняття рішень для користувача.

У результаті роботи реалізований веб-інтерфейс EnergyLab забезпечує повний зручний контроль стану всього обладнання у реальному часі, а також набагато спрощує роботу з можливими аварійними ситуаціями та дозволяє максимально ефективно взаємодіяти з локальним аналізатором та онтологічною системою діагностики.

5.2 Сторінка зібраних метрик

Сторінка «Метрики» даного дашборду EnergyLab призначена для дуже детального моніторингу отриманих даних від обладнання у реальному часі. У верхній частині наданого інтерфейсу користувач (оператор) може обрати бажаний для огляду пристрій та період часу для детального аналізу даних. Це дозволяє дуже швидко переключатися в даному випадку між різними тепловими насосами та швидко переглядати поточний стан, історичну динаміку роботи. (рисунк 5.5)

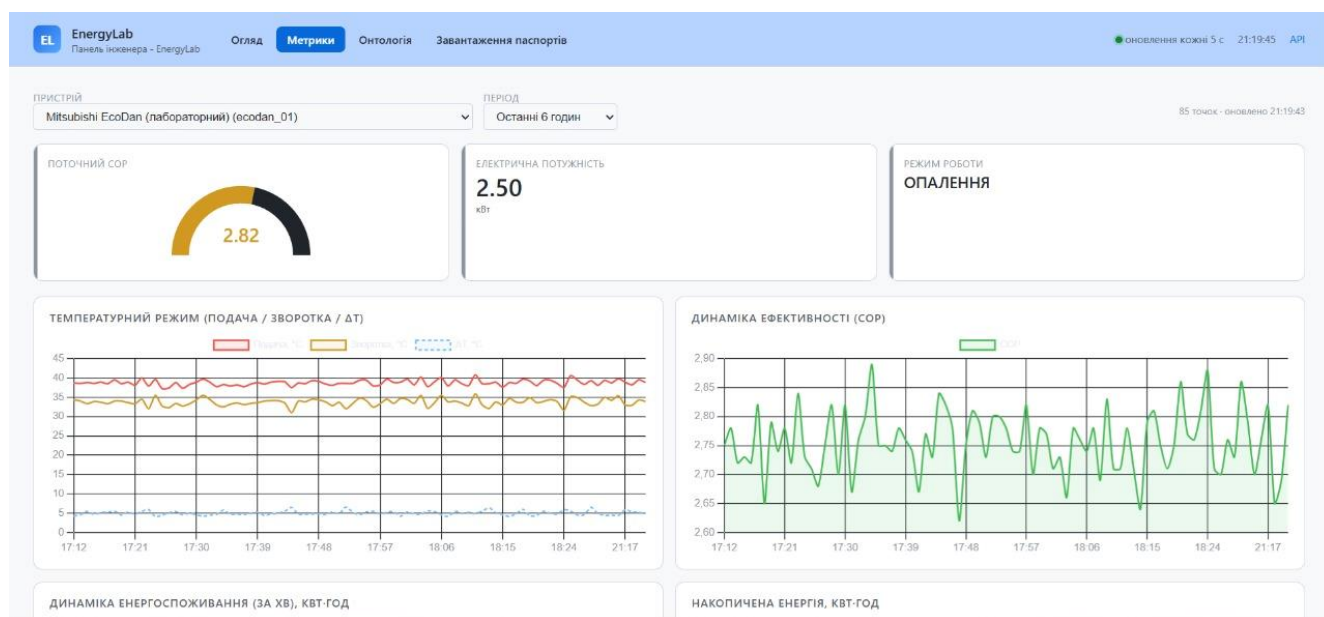


Рисунок 5.5 – Сторінка «Метрики»

У центральній частині описаної сторінки відображаються всі обрані ключові показники системи. Наприклад, показується поточний коефіцієнт ефективності COP у вигляді індикатора, електрична потужність теплового насоса та його активний режим роботи. Така візуальна форма подання дозволяє оператору дуже швидко оцінити доступний загальний стан системи без зайвої необхідності аналізувати величезні таблиці з числовими даними.

Нижче маємо блок з графіками часових рядів. Ці графіки відображають важливі параметри: динаміку температур подачі, зворотної лінії та загалом різниці температур. Окремо також показано графік зміни коефіцієнта COP. Це є важливим

для кінцевого користувача, оскільки дозволяє швидко виявляти нестабільність роботи теплового насосу, певні короточасні відхилення або поступове погіршення ефективності всієї системи EnergyLab.

Також описана вище сторінка містить блоки для аналізу енергоспоживання та накопиченої енергії теплового насосу. Завдяки всьому цьому інженер (оператор) може оцінювати загалом технічний стан обладнання, його ефективність роботи, якщо розглядати енергетичні витрати. Загалом вся сторінка «Метрики» виконує важливу функцію детального аналітичного моніторингу обраної кіберенергетичної системи та дозволяє проводити детальний аналіз роботи.

5.3 Сторінка з онтологією

Сторінка «Онтологія» дашборду EnergyLab призначена саме для візуалізації бази знань представленої системи та зв'язків між різними компонентами обладнання, його параметрами, помилками й всіма діагностичними правилами. У верхній частині наданого інтерфейсу оператор може обрати потрібний пристрій, після чого вся система завантажує відповідний онтологічний граф для нього. (рисунок 5.6)

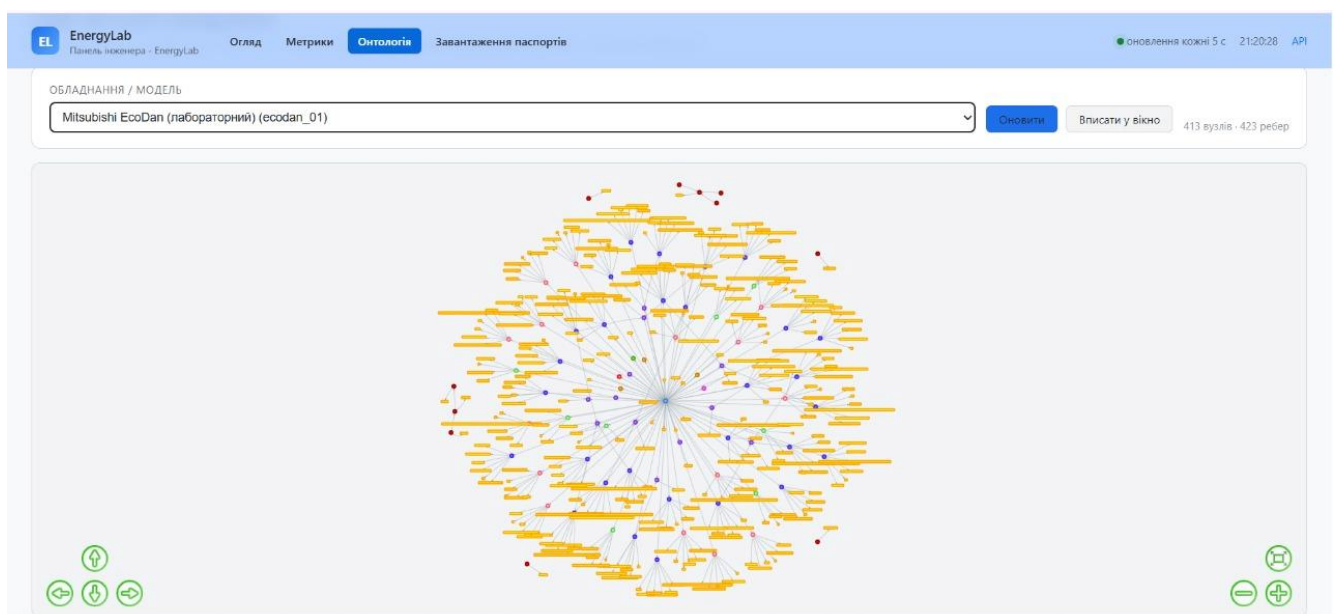


Рисунок 5.6 – Сторінка «Онтологія»

Основним центральним елементом описаної сторінки є інтерактивна графова схема. У цій схемі всі вузли та зв'язки відображають повну структуру онтології. На такому графі показуються усі компоненти теплового насоса, та параметри його роботи, коди помилок, допустимі введені межі та усі взаємозв'язки між ними. Для зручності при роботі різні типи об'єктів в онтології мають окремі яскраві кольори та структуру зв'язків.

Оператор має можливість масштабувати утворений граф, переміщуватися між різними вузлами та досліджувати надані взаємозалежності між елементами системи загалом. Це все є дуже важливим для працівника. Такі можливості дозволяють не просто бачити окрему знайдену помилку, а і розуміти повністю контекст у межах всієї бази знань. Наприклад, дашборд може показати, що якісь компоненти пов'язані з певним кодом проблеми або як певні параметри впливають на знайдену несправність.

У верхній правій частині сторінки відображається інформація про кількість вузлів та ребер отриманого графа. Це все дозволяє оцінити масштаб онтології загалом та складність усіх зв'язків між різними об'єктами. У підсумку, сторінка «Онтологія» повністю виконує функцію візуального представлення створеної бази знань та забезпечує якісну підтримку інженера під час проведення аналізу причин несправностей або структури обладнання.

5.4 Сторінка завантаження паспортів

Остання сторінка «Завантаження паспортів» має призначення для того, щоб автоматично додавати нове обладнання до онтологічної системи через PDF-документації виробника. Вона легко реалізує потрібну інтеграцію між веб-інтерфейсом EnergyLab, LLM-моделлю та створеною онтологічною базою знань. (рисунок 5.7)

EnergyLab
Панель інженера - EnergyLab

Огляд Метрики Онтологія **Завантаження паспортів**

оновлення кожні 5 с 21:20:38 API

PDF → Онтологія

Завантажте паспорт обладнання у форматі PDF. ШІ-конвертер витягне характеристики, додасть тріплети у equipment.ttl та перезавантажить Fuseki.

PDF-файл

Вибрати файл Файл не вибрано

DEVICE ID (латиниця/цифри/./-; починається з літери, напр. ehst20)

ehst20

МОДЕЛЬ LLM

claude-opus-4-7 — точніше

Обробити

ЯК ЦЕ ПРАЦЮЄ

1. PDF розбивається на сторінки, лише сторінки зі специфікаціями відбираються (≤30).
2. Відфільтрований текст відправляється до LLM з примусовим викликом інструменту record_specs.
3. Структурований профіль валідують Rydantic-схема та перетворюється у Turtle.
4. Тріплети дописуються в equipment.ttl, після чого датасет перезавантажується у Fuseki.
5. Новий пристрій одразу з'являється на сторінці «Онтологія» та на головній панелі.

Рисунок 5.7 – Сторінка «Завантаження паспортів»

У лівій частині представленої сторінки розташована форма для завантаження документів-інструкцій. Користувач може завантажити PDF-файл зі свого комп'ютера. Файл має містити технічний паспорт обладнання. Далі користувач має вказати device_id, який має бути унікальним та обрати модель штучного інтелекту для обробки завантаженого документа. Після цього дана система запускає довгий процес аналізу всієї документації через кнопку «Обробити».

Особливістю такої реалізації цієї сторінки є використання великої мовної моделі для того, щоб автоматично виділити характеристики пристроїв за допомогою технічної документації. Система EnergyLab поступово аналізує весь текст PDF, знаходить певні параметри роботи, коди помилок та всю необхідну для роботи інженера сервісну інформацію, після чого перетворює всі ці знайдені дані у один структурований формат для майбутньої онтології.

В іншій правій частині цієї сторінки розміщено блок з поясненнями «Як це працює». Даний блок пояснює послідовність обробки всього документа. Спочатку завантажений PDF розбивається на певні сторінки, після чого із цього документа вибираються лише потрібні сторінки зі специфікаціями. Далі весь текст передається до обраної моделі. І вже модель формує єдиний структурований

профіль обраного обладнання у форматі Pydantic-схеми. Після цього процесу система автоматично генерує спеціальні RDF/TTL-триплети та додає їх до єдиної онтологічної бази знань за допомогою інструменту Fuseki.

Описаний підхід є дуже важливим для кінцевого користувача, оскільки він значно спрощує весь процес наповнення онтологічної бази знань. Інженеру (оператору) більше не треба просто вручну переносити всі характеристики пристрою у систему. Це значно зменшує кількість помилок загалом, та пришвидшує додавання нових пристроїв.

Висновки до розділу 5

У п'ятому розділі описано інтерфейс дашборду EnergeLab, який забезпечує максимально зручну взаємодію користувача з усіма модулями системи. Інтерфейс об'єднує наступні функції: перегляд стану всіх доданих пристроїв, аналізу потрібних метрик, роботу зі знайденими тривогами, візуалізації наданої онтології та автоматичне завантаження технічної документації обладнання.

Використання інтерактивних графіків, яскравих кольорових статусів та чіткої онтологічної діагностики дозволяє оператору максимально швидко оцінювати стан кіберенергетичної системи та дуже ефективно реагувати на аварійні ситуації, які виникають.

Крім того, успішна інтеграція з моделями штучного інтелекту та онтологічною базою знань дуже спрощує весь процес додавання нових пристроїв та загалом розширює можливості інтелектуального аналізу подібних систем.

ВИСНОВКИ

У представленій науковій роботі проаналізовано відомі підходи до побудови різних систем моніторингу та аналізу кіберенергетичних об'єктів. Також визначено ряд суттєвих обмежень існуючих рішень, а саме розглянуто популярні централізовані SCADA-систем і IoT-платформ. Доведено, що такі популярні рішення не можуть забезпечити достатнього рівня швидкості обробки отриманих даних від обладнання кіберенергетичних лабораторій. Ці рішення мають дуже сильну залежність від центрального вузла.

Тому в даній роботі буде використовуватись протилежний підхід з використанням граничних обчислень. Обґрунтовано доцільність використання такого підходу для реалізації подібних систем моніторингу, що дозволяє також виконувати аналіз безпосередньо поблизу джерела джерела даних.

Розроблено складну архітектуру локального аналізатора на базі мікрокомп'ютера Raspberry Pi Wifi Zero V1.3. Така архітектура здатна забезпечити локальний збір, швидку обробку та передачу отриманих даних у межах кіберенергетичної лабораторії. Визначено всю апаратну конфігурацію системи та реалізовано її повноцінне розгортання з використанням відомої операційної системи Raspberry Pi OS, засобів віддаленого доступу та мережевої взаємодії.

Реалізовано робочу програмну частину системи EnergyLab з використанням популярної мови програмування Python. Це забезпечує швидкий збір даних, їх попередню обробку та аналіз стану обладнання в реальному часі. Визначено алгоритм обробки даних, який включає нормалізацію параметрів та формування висновків щодо стану обладнання.

Встановлено можливість інтеграції розробленої системи з онтологічною моделлю, що дозволяє використовувати структуровані знання для інтерпретації даних моніторингу та підвищення якості аналізу.

На основі даних моніторингу визначено підходи до оцінки стану обладнання, що дозволяє виявляти відхилення від нормального режиму роботи та формувати відповідні повідомлення для користувача.

Отримані результати підтверджують доцільність використання мікрокомп'ютера Raspberry Pi як edge-пристрою для реалізації подібних систем моніторингу з локальною обробкою даних, що забезпечує зменшення затримок обробки сирих даних, зменшення навантаження на мережу, підвищення автономності системи загалом та можливість подальшого масштабування.

Таким чином, розроблена в ході даної наукової роботи система EnergyLab граничної обробки даних забезпечує максимально швидкий моніторинг стану обладнання та здійснює повноцінний інтелектуальний аналіз даних отриманих від обладнання кіберенергетичної лабораторії засобами граничних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Parashar J. Cyber Physical Energy Systems for Smart Grid: Reliable Distribution. URL: https://www.researchgate.net/publication/385984113_Cyber_Physical_Energy_Systems_for_Smart_Grid_Reliable_Distribution (дата звернення 20.05.2026)
2. Guan X. Port cyber-physical energy systems: Advances, challenges, and future directions. URL: <https://www.sciencedirect.com/science/article/pii/S305074482500012> (дата звернення 20.05.2026)
3. Невлюдов І. Ш. Комп'ютерно-інтегровані технології виробництва технічних засобів автоматизації. Частина 1: підручник для студентів закладів вищої освіти галузей знань «Автоматизація та приладобудування», «Електроніка та телекомунікації». URL: <https://openarchive.nure.ua/entities/publication/640b1a6f-a561-42f2-9bf1-7cd7c5a202a6> (дата звернення 20.05.2026)
4. Арсен'єв В. Теплові насоси: основи теорії і розрахунку: навчальний посібник. URL: <https://essuir.sumdu.edu.ua/server/api/core/bitstreams/b34235c0-8b92-43ae-9c8b-8641b4814be4/content> (дата звернення 20.05.2026)
5. Руденко І., Бронніков А. Система моніторингу автоматизованих систем на підприємстві. URL: <https://openarchive.nure.ua/entities/publication/d74a94e9-2af7-45ea-bdb2-2b98db7e1ed8> (дата звернення 20.05.2026)
6. Кучернюк П. В. Технології моніторингу та трафік-інжинірингу в телекомунікаційних мережах : підручник для здобувачів ступеня магістра за спеціальністю 172 «Телекомунікації та радіотехніка». URL: <https://ela.kpi.ua/server/api/core/bitstreams/cb5f54ef-a510-46f7-adbb-b5b3b28016a6/content> (дата звернення 20.05.2026)
7. Tyagi B. Edge Computing: Everything You Need to Know. URL: <https://www.globallogic.com/insights/white-papers/edge-computing-everything-you-need-to-know/> (дата звернення 20.05.2026)

8. Bailey D. Practical SCADA for Industry. Oxford : Newnes, Elsevier. URL:https://s2.smu.edu/~nair/ftp/senior_design/scada/Practical_SCADA_for_Industry.pdf (дата звернення 20.05.2026)
9. Evans D. The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. URL:https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (дата звернення 20.05.2026)
10. Monk S. Programming the Raspberry Pi: Getting Started with Python. URL:https://climber.uml.edu.ni/Download_PDFS/libweb/4050177/ProgrammingTheRaspberryPiGettingStartedWithPythonSimonMonk.pdf (дата звернення 20.05.2026)
11. Verma H. K. Telemetry Basics. School of Engineering and Technology. URL:<https://www.scribd.com/document/203051623/Chapter-1-Telemetry-Basics-9-Pages> (дата звернення 20.05.2026)
12. An Overview of the Secure Shell (SSH). Albuquerque. URL:https://www.vandyke.com/solutions/ssh_overview/ssh_overview.pdf (дата звернення 20.05.2026)
13. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy & Jupyter. URL: <https://www.lkhibra.ma/books/Python-for-Data-Analysis.pdf> (дата звернення 20.05.2026)
14. Chacon S., Straub B. Pro Git. URL: https://chamber.ua/wp-content/uploads/2019/12/progit_v2.1.2.pdf (дата звернення 20.05.2026)
15. Rižnar I., Gaisov A. Raspberry Pi as a Foundation for Boosting Computer and Technology Literacy: Amplifying Understanding through Projects. URL:<https://www.hippocampus.si/ISBN/978-961-293-378-4.pdf> (дата звернення 20.05.2026)
16. Nikiforova O., An Approach to Generation of the UML Sequence Diagram. URL:https://personales.upv.es/thinkmind/dl/conferences/icsea/icsea_2016/icsea_2016_6_10_10024.pdf (дата звернення 20.05.2026)
17. NumPy Developers. What is NumPy? NumPy Documentation. URL:

<https://numpy.org/devdocs/user/whatisnumpy.html> (дата звернення 20.05.2026)

18. IT Notes Wiki. Pydantic 2: загальний огляд. URL: <https://www.it-notes.wiki/python/pydantic-2-general-overview/> (дата звернення 20.05.2026)

ДОДАТОК А Лістинг розробленої системи

Файл pi_zero_analyzer.py

```
from __future__ import annotations

import logging
import os
import sys
import time
from pathlib import Path

import numpy as np
import paho.mqtt.client as mqtt
import requests
from dotenv import load_dotenv

sys.path.insert(0, str(Path(__file__).resolve().parent.parent))
from shared.schemas import (
    AlertPayload,
    Diagnosis,
    MetricsMessage,
    StateMessage,
    TOPIC_METRICS_WILDCARD,
    TOPIC_STATE,
    utcnow_iso,
)

sys.path.insert(0, str(Path(__file__).resolve().parent))
from diagnose import DiagnosticEngine # noqa: E402

load_dotenv(Path(__file__).resolve().parent.parent / ".env")
logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")
log = logging.getLogger("pi_zero_analyzer")

MQTT_BROKER = os.getenv("MQTT_BROKER", "localhost")
MQTT_PORT = int(os.getenv("MQTT_PORT", "1883"))
ONTOLOGY_API = os.getenv("ONTOLOGY_API", "http://localhost:5000")
ALERTS_API = os.getenv("ALERTS_API", "http://localhost:5003")
REMIND_INTERVAL_SEC = int(os.getenv("REMIND_INTERVAL_SEC", "900"))
BOUNDS_TTL_SEC = int(os.getenv("BOUNDS_TTL_SEC", "300"))

HERE = Path(__file__).parent

def _resolve_npz() -> Path:
    override = os.getenv("AE_WEIGHTS")
    if override:
        return Path(override)
    for name in ("anomaly_ae_strict.npz", "anomaly_ae.npz"):
        p = HERE / name
        if p.exists():
            return p
    return HERE / "anomaly_ae.npz"

NPZ_PATH = _resolve_npz()

FEATURES = ["power_kw", "delta_t_c", "flow_temp_c", "outdoor_temp_c", "cop"]

class NumpyAutoencoder:
```

```

def __init__(self, npz_path: Path):
    data = np.load(npz_path)
    self.mean = data["mean"].astype(np.float32)
    self.std = data["std"].astype(np.float32)
    self.threshold = float(data["threshold"])
    weight_keys = sorted(k for k in data.files
                          if k.endswith("_W") and not k.startswith("__"))
    self.layers: list[tuple[np.ndarray, np.ndarray]] = []
    for w_key in weight_keys:
        b_key = w_key[:-2] + "_b"
        W = data[w_key].astype(np.float32)
        b = data[b_key].astype(np.float32)
        self.layers.append((W, b))
    log.info("Loaded autoencoder: %d layers, threshold MSE=%.6f",
            len(self.layers), self.threshold)

def _forward(self, x: np.ndarray) -> np.ndarray:
    for i, (W, b) in enumerate(self.layers):
        x = x @ W + b
        if i < len(self.layers) - 1:
            np.maximum(x, 0.0, out=x) # in-place relu
    return x

def _vectorize(self, metrics: dict) -> np.ndarray:
    flow = metrics.get("flow_temp_c", 0.0) or 0.0
    ret = metrics.get("return_temp_c", 0.0) or 0.0
    values = {
        "power_kw": metrics.get("power_kw", 0.0) or 0.0,
        "delta_t_c": flow - ret,
        "flow_temp_c": flow,
        "outdoor_temp_c": metrics.get("outdoor_temp_c", 0.0) or 0.0,
        "cop": metrics.get("cop", 3.0) if metrics.get("cop") is not None else 3.0,
    }
    vec = np.array([values[f] for f in FEATURES], dtype=np.float32)
    return (vec - self.mean) / self.std

def score(self, metrics: dict) -> tuple[bool, float, float]:
    x = self._vectorize(metrics).reshape(1, -1)
    recon = self._forward(x.copy())
    mse = float(np.mean((x - recon) ** 2))
    return mse > self.threshold, mse, mse / max(self.threshold, 1e-9)

def _try_load_detector() -> NumpyAutoencoder | None:
    if not NPZ_PATH.exists():
        log.warning(
            "Ваги моделі не знайдено (%s) — fallback до rule-based-only. "
            "На тренуйте на dev-машині: python analyzer/train_tf_model.py",
            NPZ_PATH,
        )
        return None
    try:
        return NumpyAutoencoder(NPZ_PATH)
    except Exception:
        log.exception("Не вдалось завантажити ваги — fallback до rule-based")
        return None

DETECTOR: NumpyAutoencoder | None = _try_load_detector()

```

```

DIAGNOSER = DiagnosticEngine(
    ontology_api_url=ONTOLOGY_API,
    cache_ttl_sec=BOUNDS_TTL_SEC,
)

BOUNDS_CACHE: dict[str, dict] = {}
BOUNDS_FETCHED: dict[str, float] = {}
LAST_STATE: dict[str, str] = {}
LAST_REMIND: dict[str, float] = {}

def get_bounds(device_id: str) -> dict:
    now = time.time()
    if device_id in BOUNDS_CACHE and now - BOUNDS_FETCHED.get(device_id, 0) < BOUNDS_TTL_SEC:
        return BOUNDS_CACHE[device_id]
    try:
        r = requests.get(f"{ONTOLOGY_API}/device/{device_id}/expected-bounds", timeout=3)
        r.raise_for_status()
        BOUNDS_CACHE[device_id] = r.json()
    except requests.RequestException as exc:
        log.warning("Ontology API unreachable for %s (%s)", device_id, exc)
        BOUNDS_CACHE[device_id] = BOUNDS_CACHE.get(device_id, {})
    BOUNDS_FETCHED[device_id] = now
    return BOUNDS_CACHE[device_id]

def rule_based_checks(metrics: dict, bounds: dict) -> list[str]:
    anomalies: list[str] = []
    cop = metrics.get("cop")
    power = metrics.get("power_kw", 0.0)
    flow = metrics.get("flow_temp_c", 0.0)

    if bounds.get("min_cop") is not None and cop is not None and cop < bounds["min_cop"]:
        anomalies.append(f"cop_below_nominal({cop:.2f}<{bounds['min_cop']:.2f})")
    if bounds.get("max_power_kw") is not None and power > bounds["max_power_kw"]:
        anomalies.append(f"power_over_limit({power:.2f}>{bounds['max_power_kw']:.2f})")
    if bounds.get("max_flow_c") is not None and flow > bounds["max_flow_c"]:
        anomalies.append(f"flow_temp_over_limit({flow:.1f}>{bounds['max_flow_c']:.1f})")
    if bounds.get("min_flow_c") is not None and flow < bounds["min_flow_c"]:
        anomalies.append(f"flow_temp_under_limit({flow:.1f}<{bounds['min_flow_c']:.1f})")
    return anomalies

def classify(rule_anomalies: list[str], ml_anomaly: bool) -> tuple[str, list[str]]:
    if rule_anomalies:
        codes = rule_anomalies + (["ml_outlier"] if ml_anomaly else [])
        return "anomaly", codes
    if ml_anomaly:
        return "warning", ["ml_outlier"]
    return "normal", []

def analyze(msg: MetricsMessage) -> StateMessage:
    metrics = msg.metrics.model_dump()
    bounds = get_bounds(msg.device_id)
    rule_anomalies = rule_based_checks(metrics, bounds)

    ml_anomaly = False
    ml_explanation = "ML detector disabled"

```

```

confidence = 1.0
if DETECTOR is not None:
    ml_anomaly, mse, ratio = DETECTOR.score(metrics)
    ml_explanation = f"ML mse={mse:.4f} ratio={ratio:.2f} × threshold"
    confidence = float(np.clip(0.5 + (ratio - 1.0) * 0.25, 0.0, 1.0))

state, anomalies = classify(rule_anomalies, ml_anomaly)
diagnoses: list[Diagnosis] = (
    DIAGNOSER.enrich(anomalies, msg.device_id) if state != "normal" else []
)

return StateMessage(
    device_id=msg.device_id,
    timestamp=utcnow_iso(),
    state=state,
    anomalies=anomalies,
    confidence=confidence if state != "normal" else 1.0,
    explanation=f"{ml_explanation}; rules matched: {len(rule_anomalies)}",
    diagnoses=diagnoses,
)

def _should_remind(device_id: str) -> bool:
    now = time.time()
    if now - LAST_REMIND.get(device_id, 0) >= REMIND_INTERVAL_SEC:
        LAST_REMIND[device_id] = now
        return True
    return False

def send_heartbeat(state: StateMessage, metrics_dump: dict, bounds: dict) -> None:
    try:
        requests.post(
            f"{ALERTS_API}/api/heartbeat",
            json={
                "device_id": state.device_id,
                "timestamp": state.timestamp,
                "state": state.state,
                "metrics": metrics_dump,
                "bounds": bounds,
            },
            timeout=2,
        )
    except requests.RequestException as exc:
        log.debug("heartbeat dropped for %s (%s)", state.device_id, exc)

def forward_alert(state: StateMessage, metrics_dump: dict, bounds: dict) -> None:
    if state.state not in ("warning", "anomaly"):
        return
    payload = AlertPayload(
        device_id=state.device_id,
        timestamp=state.timestamp,
        severity=state.state,
        anomaly_codes=state.anomalies,
        explanation=state.explanation,
        confidence=state.confidence,
        metrics_snapshot=metrics_dump,
        bounds_snapshot=bounds,
        diagnoses=state.diagnoses,
    )
    try:

```

```

requests.post(
    f"{ALERTS_API}/api/alerts",
    data=payload.model_dump_json(),
    headers={"Content-Type": "application/json"},
    timeout=3,
).raise_for_status()
except requests.RequestException as exc:
    log.warning("alerts_server unreachable (%s) — alert dropped (device=%s)",
        exc, state.device_id)

def on_message(client: mqtt.Client, _userdata, mqtt_msg: mqtt.MQTTMessage) -> None:
    try:
        incoming = MetricsMessage.model_validate_json(mqtt_msg.payload)
    except Exception:
        log.exception("Bad metrics payload on %s", mqtt_msg.topic)
        return

    state = analyze(incoming)
    out_topic = TOPIC_STATE.format(device_id=state.device_id)
    client.publish(out_topic, state.model_dump_json(), qos=0)

    bounds = get_bounds(state.device_id)
    metrics_dump = incoming.metrics.model_dump()
    send_heartbeat(state, metrics_dump, bounds)

    prev = LAST_STATE.get(state.device_id, "normal")
    LAST_STATE[state.device_id] = state.state
    state_changed = prev != state.state
    is_problem = state.state in ("warning", "anomaly")

    should_forward = is_problem and (state_changed or _should_remind(state.device_id))
    if should_forward:
        forward_alert(state, metrics_dump, bounds)

    log.info("%s → %s (conf=%.2f) anomalies=%s%s",
        state.device_id, state.state, state.confidence, state.anomalies,
        " → ALERT FORWARDED" if should_forward else "")
    for diag in state.diagnoses:
        log.info(" · %s", DIAGNOSER.format_for_log(diag))

def on_connect(client, *_):
    log.info("MQTT connected - subscribing %s", TOPIC_METRICS_WILDCARD)
    client.subscribe(TOPIC_METRICS_WILDCARD, qos=0)

def main() -> None:
    mode = "ML+rule-based (numpy)" if DETECTOR is not None else "rule-based only (fallback)"
    log.info("Pi Zero analyzer starting [%s] (broker=%s:%d, ontology=%s, alerts=%s)",
        mode, MQTT_BROKER, MQTT_PORT, ONTOLOGY_API, ALERTS_API)
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, client_id="pi-zero-analyzer")
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_BROKER, MQTT_PORT, keepalive=60)
    client.loop_forever()

if __name__ == "__main__":
    main()

```

Файл diagnose.py

```

from __future__ import annotations

import json
import logging
import sys
import time
from pathlib import Path

import requests

sys.path.insert(0, str(Path(__file__).resolve().parent.parent))
from shared.schemas import Diagnosis # noqa: E402

log = logging.getLogger("diagnose")

_DEFAULT_MAPPING_PATH = Path(__file__).parent / "diagnostic_mapping.json"

class DiagnosticEngine:
    def __init__(
        self,
        ontology_api_url: str,
        mapping_path: Path = _DEFAULT_MAPPING_PATH,
        cache_ttl_sec: int = 300,
        max_diagnoses_per_alert: int = 6,
    ):
        self.api_url = ontology_api_url.rstrip("/")
        self.mapping_path = mapping_path
        self.cache_ttl = cache_ttl_sec
        self.max_diagnoses = max_diagnoses_per_alert

        self._cache: dict[str, dict] = {} # device_id → context dict
        self._fetched_at: dict[str, float] = {}

        self.rules = self._load_mapping()
        log.info("DiagnosticEngine: loaded %d mapping rules from %s",
            len(self.rules), self.mapping_path)

    def _load_mapping(self) -> list[dict]:
        if not self.mapping_path.exists():
            log.warning("Diagnostic mapping not found at %s — engine will only "
                "emit empty diagnoses.", self.mapping_path)
            return []
        try:
            data = json.loads(self.mapping_path.read_text(encoding="utf-8"))
        except json.JSONDecodeError as e:
            log.error("Bad JSON in %s: %s", self.mapping_path, e)
            return []
        return data.get("rules", [])

    def reload_mapping(self) -> int:
        self.rules = self._load_mapping()
        return len(self.rules)

    def _is_fresh(self, device_id: str) -> bool:
        if device_id not in self._cache:
            return False
        return (time.time() - self._fetched_at.get(device_id, 0)) < self.cache_ttl

```

```

def _fetch_context(self, device_id: str) -> dict | None:
    try:
        r = requests.get(
            f"{self.api_url}/device/{device_id}/diagnostic-context",
            timeout=3,
        )
        r.raise_for_status()
        return r.json()
    except requests.RequestException as exc:
        log.warning("Failed to fetch diagnostic context for %s: %s",
            device_id, exc)
        return None

def get_context(self, device_id: str) -> dict:
    if not self._is_fresh(device_id):
        fresh = self._fetch_context(device_id)
        if fresh is not None:
            self._cache[device_id] = fresh
            self._fetched_at[device_id] = time.time()
    return self._cache.get(device_id, {"bounds": {}, "faults": [], "error_codes": []})

@staticmethod
def _code_prefix(anomaly_code: str) -> str:
    return anomaly_code.split("(", 1)[0].strip()

def _match_rule(self, anomaly_code: str) -> dict | None:
    prefix = self._code_prefix(anomaly_code)
    for rule in self.rules:
        if rule.get("code_prefix") == prefix:
            return rule
    return None

@staticmethod
def _affects_match(affects: str | None, keywords: list[str]) -> bool:
    if not affects:
        return False
    a = affects.lower()
    return any(k.lower() in a for k in keywords)

def _diagnose_one_code(
    self, anomaly_code: str, ctx: dict,
) -> list[Diagnosis]:
    rule = self._match_rule(anomaly_code)
    if rule is None:
        return []
    out: list[Diagnosis] = []
    affects_keywords = rule.get("affects_components", [])
    if affects_keywords:
        for f in ctx.get("faults", []):
            if self._affects_match(f.get("affects"), affects_keywords):
                out.append(Diagnosis(
                    matched_code=anomaly_code,
                    kind="fault",
                    cause=f.get("cause"),
                    solution=f.get("solution"),
                    fault_iri=f.get("id"),
                    fault_symptom=f.get("symptom"),
                    affects_component=f.get("affects"),
                    severity=f.get("severity"),
                    confidence=0.7,
                ))

```

```

    ))

    wanted_codes = set(rule.get("error_codes", []))
    if wanted_codes:
        for ec in ctx.get("error_codes", []):
            if ec.get("code") in wanted_codes:
                out.append(Diagnosis(
                    matched_code=anomaly_code,
                    kind="error_code",
                    error_code=ec.get("code"),
                    error_description=ec.get("description"),
                    error_action=ec.get("action"),
                    severity=ec.get("severity"),
                    confidence=1.0,
                ))

    hint = rule.get("hint")
    if not out and hint:
        out.append(Diagnosis(
            matched_code=anomaly_code,
            kind="hint",
            hint=hint,
            confidence=0.3,
        ))
    return out

def enrich(self, anomaly_codes: list[str], device_id: str) -> list[Diagnosis]:

    if not anomaly_codes:
        return []
    ctx = self.get_context(device_id)
    if not ctx.get("faults") and not ctx.get("error_codes"):
        log.debug("No ontology context for %s — diagnoses will be hints only",
            device_id)

    all_diag: list[Diagnosis] = []
    for code in anomaly_codes:
        all_diag.extend(self._diagnose_one_code(code, ctx))

    seen: set[tuple[str, str]] = set()
    deduped: list[Diagnosis] = []
    for d in sorted(all_diag, key=lambda x: -x.confidence):
        key = (d.kind, d.fault_iri or d.error_code or d.hint or "")
        if key in seen:
            continue
        seen.add(key)
        deduped.append(d)

    return deduped[: self.max_diagnoses]
@staticmethod
def format_for_log(diag: Diagnosis) -> str:
    if diag.kind == "error_code":
        sev = f" sev={diag.severity}" if diag.severity else ""
        return f"[{diag.error_code}?{sev}] {diag.error_description or ''} → {diag.error_action or ''}"
    if diag.kind == "fault":
        sev = f" sev={diag.severity}" if diag.severity else ""
        return f"[fault:{diag.affects_component or '?'}{sev}] {diag.cause or ''} → {diag.solution or ''}"
    return f"[hint] {diag.hint or ''}"

```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

СИСТЕМА ДЛЯ АНАЛІЗУ СТАНУ ОБЛАДНАННЯ ЛАБОРАТОРІЇ КІБЕРЕНЕРГЕТИЧНИХ СИСТЕМ ЗАСОБАМИ ГРАНИЧНИХ ОБЧИСЛЕНЬ

виконала: студентка групи ТВ-21

Безклинська Валерія Вікторівна

керівник: асистент Голець Владислав Олександрович

2026

Актуальність теми

Сучасні кіберенергетичні системи потребують постійного моніторингу та швидкого реагування на відхилення у роботі. Централізовані системи мають затримки передачі даних та залежать від стабільної мережі. Використання технології граничних обчислень дозволяє виконувати локальну обробку безпосередньо поблизу джерела, що забезпечує швидшу реакцію.

Метою: аналіз стану обладнання кіберенергетичної лабораторії засобами граничних обчислень.

Об'єкт дослідження: процес аналізу параметрів роботи обладнання кіберенергетичних систем.

Предметом дослідження: є методи локальної обробки даних, виявлення аномалій та програмні засоби моніторингу стану обладнання із використанням технології Edge Computing, MQTT протоколу.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 2

Постановка задачі

Формулювання задачі: Необхідно розробити систему для аналізу стану обладнання лабораторії кіберенергетичних систем засобами граничних обчислень (Edge Computing), яка забезпечуватиме локальну обробку даних, виявлення аномалій та відображення результатів аналізу у режимі реального часу.

Перелік задач, які потрібно було розв'язати для досягнення цілі:

- провести аналіз предметної області кіберенергетичних систем;
- дослідити сучасні системи моніторингу та технологію Edge Computing;
- проаналізувати принципи роботи теплових насосів Mitsubishi Electric;
- спроектувати архітектуру програмної системи;
- реалізувати edge-analyzer для локальної обробки даних на Raspberry Pi;
- розробити веб-інтерфейс;
- реалізувати побудову графіків параметрів системи у режимі реального часу;
- проаналізувати результати роботи системи та оформити документацію.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 3

Аналіз предметної області

Існуючі системи моніторингу:

1. SCADA-системи
2. Хмарні IoT-платформи
3. Централізовані системи аналізу
4. Системи телеметричного моніторингу



Основні недоліки їх:

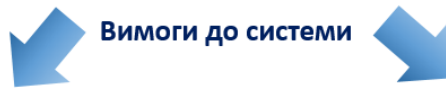
- Високі затримки передачі даних
- Залежність від мережі та серверів
- Велике навантаження на хмару
- Відсутність локальної edge-аналітики
- Недостатній семантичний аналіз даних
- Обмежене виявлення аномалій у реальному часі



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 4

Проектування системи



Вимоги до системи

Функціональні вимоги

- Локальний edge-аналіз даних
- Виявлення аномалій
- Rule-based перевірки
- ML-аналіз стану системи
- Інтеграція онтології
- Відображення дашборду у реальному часі
- Перегляд історії подій та графіків
- Формування та обробка тривог

Нефункціональні вимоги

- Робота в режимі реального часу
- Низькі затримки обробки
- Автономність edge-пристрою
- Масштабованість системи
- Надійність передачі даних
- Простота розширення системи
- Підтримка мережевої взаємодії
- Зручний веб-інтерфейс

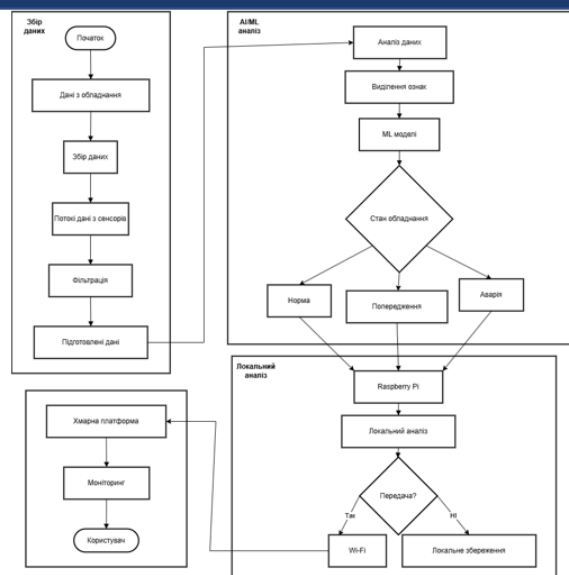


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 5

Проектування системи

Алгоритм функціонування всіх компонентів системи EnergyLab

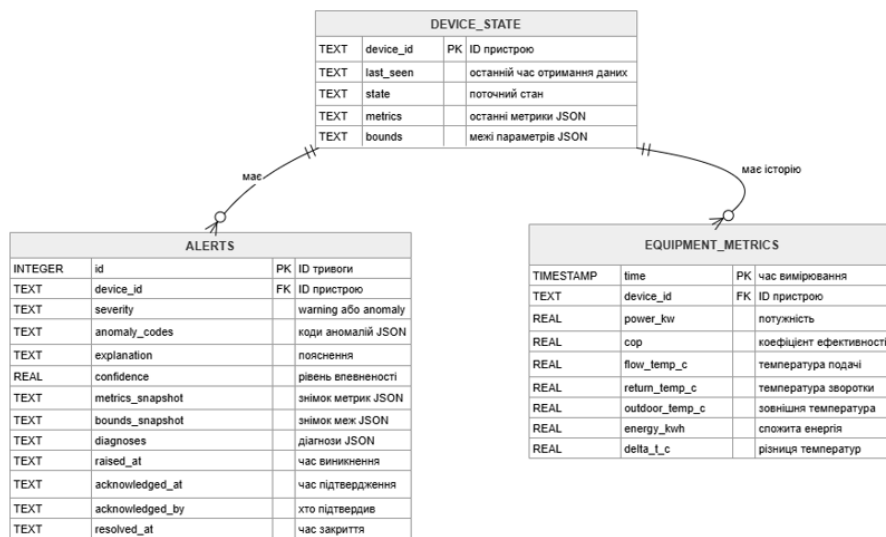


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 6

Проектування системи

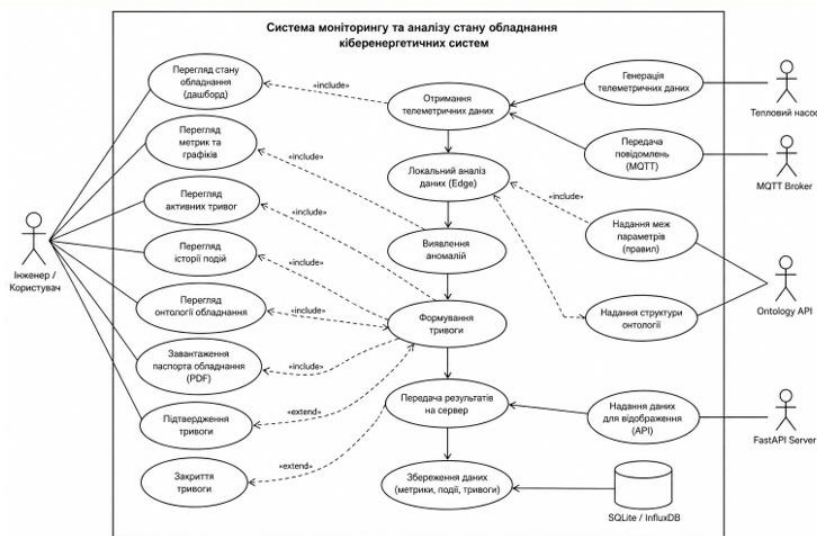
Модель бази даних



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 7

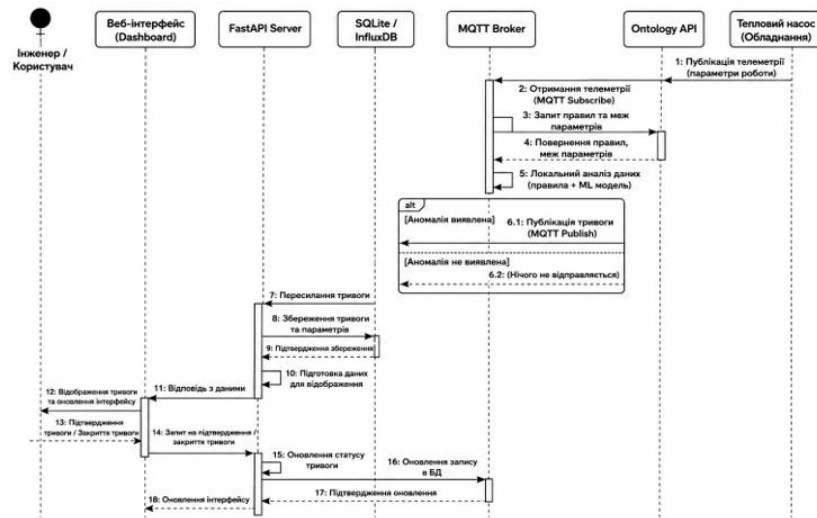
Проектування системи – ДІАГРАМА ПРЕЦЕДЕНТІВ



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 8

Проектування системи – ДІАГРАМА ПОСЛІДОВНОСТЕЙ



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 9

Реалізація системи

Опис процесу розробки

- Реалізовано edge-analyzer на Raspberry Pi
- MQTT використовується для отримання телеметрії
- Виконано rule-based та ML-аналіз
- FastAPI реалізує backend та REST API
- SQLite та InfluxDB використовуються для збереження даних
- Web dashboard забезпечує моніторинг обладнання

```

def on_connect(client, _, _):
    log.info(msg: "MQTT connected - subscribing %s", *args: TOPIC_METRICS_WILDCARD)
    client.subscribe(TOPIC_METRICS_WILDCARD, qos=0)

if bounds.get("min_cop") is not None and cop is not None and cop < bounds["min_cop"]:
    anomalies.append(f"cop_below_nominal({cop:.2f}<{bounds['min_cop']:.2f})")

ml_anomaly, mse, ratio = DETECTOR.score(metrics)
ml_explanation = f"ML mse={mse:.4f} ratio={ratio:.2f}* threshold"
    
```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 10

Реалізація системи

Апаратна частина реалізації

Для побудови було використано :

- мікрокомп'ютер Raspberry Pi (версія 1.3 Wifi Zero - модель без вбудованого модуля Wi-Fi);
- карта пам'яті microSD обсягом 32 ГБ;
- адаптер живлення (micro-USB);
- USB OTG-кабель,;
- зовнішній Wi-Fi модуль (USB);
- ноутбук (персональний комп'ютер).

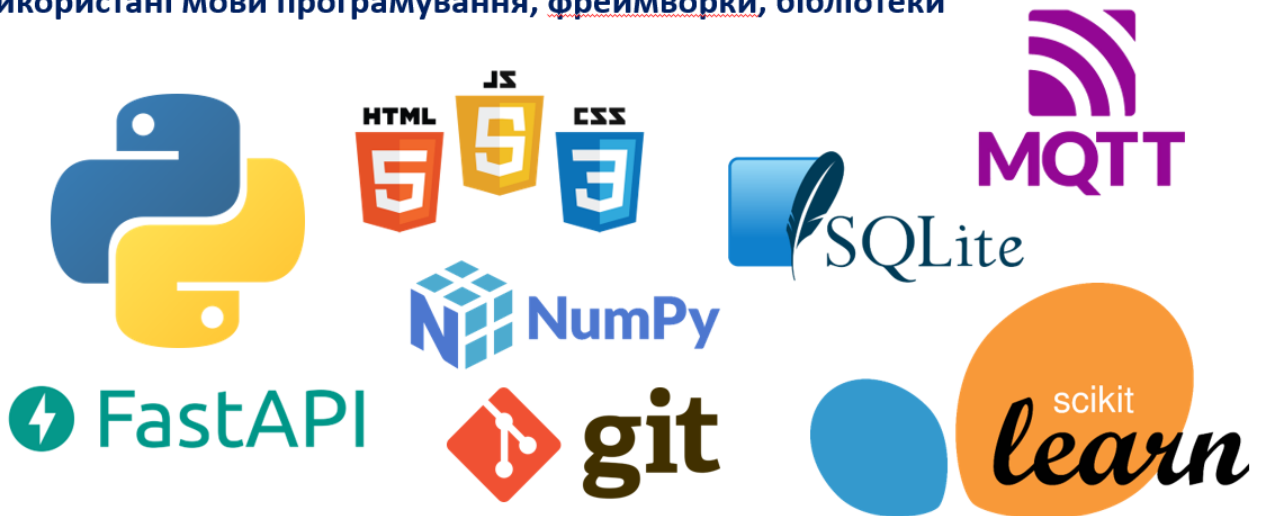


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 11

Реалізація системи

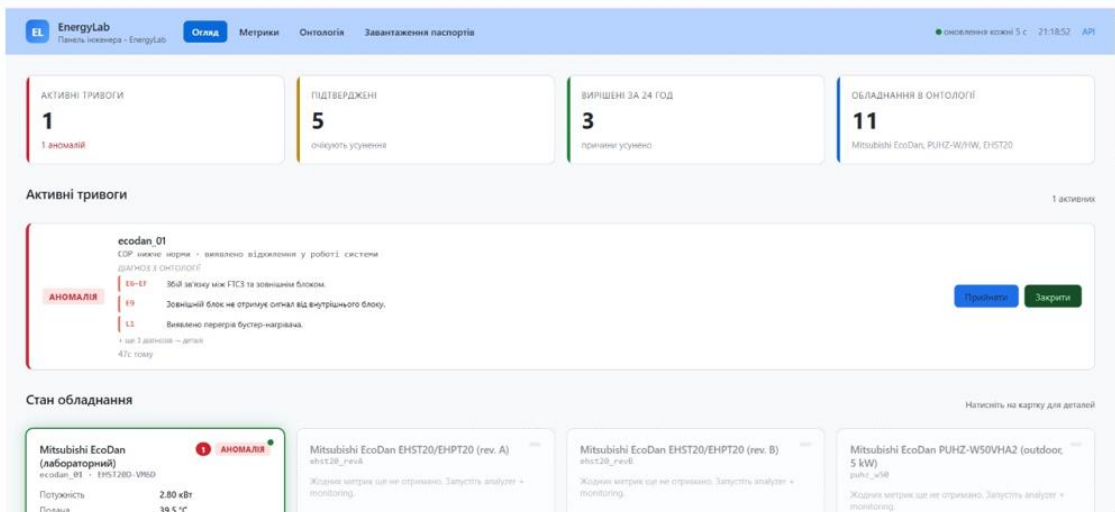
Використані мови програмування, фреймворки, бібліотеки



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 12

Реалізація системи



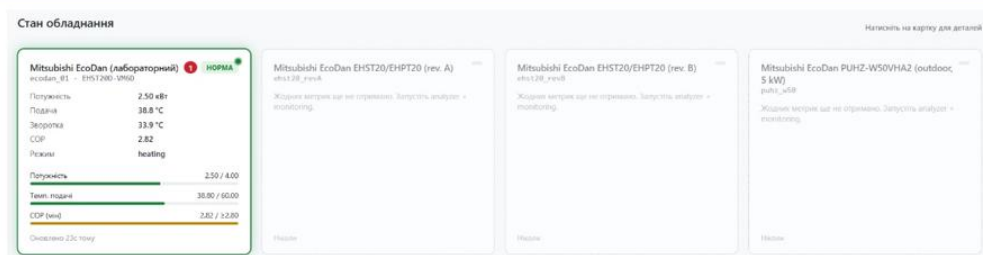
Головна сторінка [дашборду EnergyLab](#)



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 13

Реалізація системи



Блок зі станом обладнання

Історія подій

18.05.2026, 21:18:02 АНОМАЛІЯ - ecodan_01 - ВИРІШЕНО
COP нижче норми - виявлено відхилення у роботі системи
ДІАГНОЗ З ОНТОЛОГІЇ
E9-EF Збій зв'язку між FTC3 та зовнішнім блоком.
E9 Зовнішній блок не отримав сигнал від внутрішнього блоку.
L1 Виявлено перегрів бустер-нагрівача.
+ ще 3 деталі - деталі

Блок з історією подій



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 14

Тестування системи

Методи тестування

- Модульне тестування (Unit Testing)

Засоби тестування:

- Pytest
- Mock-об'єкти
- Тестові дані

Результат:

- Кожен файл має мінімальний набір unit-тестів

```
rootdir: D:\KPI\diploma\code_new\Energy-System
collected 6 items
```

```
analyzer\tests\test_pi_zero_analyzer.py .....
```

```
[100%]
```

```
=====
6 passed in 0.84s =====
=====
```

```
(code_new) PS D:\KPI\diploma\code_new\Energy-System> █
```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 15

Впровадження та супровід

Процес розгортання

- Docker Compose для запуску сервісів
- Raspberry Pi як edge-вузол
- MQTT для передачі телеметрії
- FastAPI backend та dashboard

README

Підготовка системи:

```
sudo apt update
sudo apt install -y git python3-venv libopenblas0
```

Розгортання:

```
git clone https://github.com/elQeR/Energy-System.git
cd Energy-System
MQTT_BROKER=192.168.1.10 \
ORTOLOGY_API=http://192.168.1.10:5002 \
ALERTS_API=http://192.168.1.10:5003 \
DEVICE_ID=ecodan_01 \
./scripts/pi-zero-bootstrap.sh
```



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 16

Висновки

Досягнення мети та виконання завдань

- Розроблено edge-систему моніторингу та аналізу стану обладнання кіберенергетичної лабораторії.
- Реалізовано локальний аналіз на Raspberry Pi.
- Поєднано ML, rule-based та онтологічний підходи.
- Створено веб-інтерфейс моніторингу та систему тривоги.

Практична та наукова цінність

- Локальний аналіз у режимі реального часу.
- Зменшення навантаження на мережу та сервер.
- Використання lightweight ML для edge-пристроїв.

Перспективи розвитку

- Можливість передбачити майбутні збої в системі
- Підтримка нових типів обладнання.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

№ 17



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"