Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Кафедра комп'ютерно-інтегрованих технологій автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розроблення програмного забезпечення (мікросервіса) для передачі даних між програмами
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТСІ-21-1
Нікіта ПРОКОПЕНКО
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно інтегровані технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Системна інженерія
(повна назва освітньої програми)

Керівник доц. Леонід ІВАНОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Прокопенко Нікіта Вадимович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

23 червня 2025 р. Нікіта ПРОКОПЕНКО

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій

Кафедра Комп'ютерно–інтегрованих технологій, автоматизації та робототехніки»

Рівень вищої освіти перший (бакалаврський)

Спеціальність 151 Автоматизація та комп'ютерно інтегровані технології

(код і повна назва)

Тип програми освітньо–професійна

Освітня програма Системна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Прокопенко Никиті Вадимовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного забезпечення (мікросервіса) для передачі даних між програмами

Затверджена наказом університету від 19 травня 2025 р. № 391 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 24 червня 2025 р.

3. Вихідні дані до роботи Python 3.10, WebSocket, ESP32, Windows 10, мережеве середовище

4. Перелік питань, що потрібно опрацювати в роботі Дослідження методів обміну даними в розподілених системах, обґрунтування вибору архітектури, проєктування структури системи, реалізація брокера та клієнтів, розробка графічного інтерфейсу, тестування та оцінка ефективності

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Графічний матеріал у вигляді презентації формату pptx в форматі А4 15 с.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапу роботи	Строк / термін виконання	Примітка
1	Отримання завдання до кваліфікаційної роботи	28.04.2025	виконано
2	Аналіз предметної області, постановка задачі	30.04–06.05.2025	виконано
3	Опрацювання науково–технічної літератури	07.05–12.05.2025	виконано
4	Розробка загальної архітектури мікросервісної системи	12.05–18.05.2025	виконано
5	Реалізація прототипу WebSocket–брокера	19.05–25.05.2025	виконано
6	Створення клієнтських застосунків–емуляторів і GUI	25.05–01.06.2025	виконано
7	Інтеграція компонентів у єдину систему, реалізація аварійної обробки	02.06–04.06.2025	виконано
8	Проведення тестування, моделювання аварійних ситуацій	05.06–06.06.2025	виконано
9	Оформлення пояснювальної записки, підготовка	06.06–08.06.2025	виконано
10	Подання роботи на нормоконтроль		
11	Подання роботи на перевірку Інтернет–сервісом StrikePlagiarism		
12	Подання роботи на рецензію		
13	Подання роботи на підпис зав. кафедри		
14	Подання кваліфікаційної роботи в ЕК		

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____ Нікіта ПРОКОПЕНКО

(підпис) (власне ім'я, прізвище)

Керівник роботи _____ доц. Леонід ІВАНОВ

(підпис) (посада, власне ім'я, прізвище

РЕФЕРАТ

Пояснювальна записка: 88 с., 3 табл, 17 рис., 3 дод., 22 джерела.

WEBSOCKET, PYTHON, БРОКЕР ПОВІДОМЛЕНЬ, МІКРОСЕРВІС, WEBSOCKET, TTKLBOOTSTRAP, GUI.

Об’єкт роботи – процес обміну даними між програмними модулями в розподіленому програмному середовищі.

Предмет роботи – архітектура та програмна реалізація мікросервісу для передачі повідомлень з використанням брокера на основі WebSocket-протоколу.

Мета роботи – розробити програмне забезпечення (мікросервіс), що забезпечує надійний обмін структурованими повідомленнями між програмами у реальному часі, з урахуванням вимог до масштабованості, стабільності та інтеграції в автоматизовані системи управління.

У роботі описано процес проєктування та реалізації мікросервісної системи для передачі даних між програмними компонентами. Центральним елементом архітектури є брокер повідомлень, побудований на основі WebSocket-протоколу з підтримкою асинхронної обробки запитів, автентифікації користувачів і маршрутизації повідомлень.

Розроблено клієнтські модулі – емулятори сенсорів, вузли моніторингу та аварійні обробники – які взаємодіють із брокером у режимі реального часу. Застосовано формат обміну повідомленнями JSON, визначено набір топіків і реалізовано механізми ескалації алармів.

Результати роботи відповідають Цілям сталого розвитку, зокрема Цілі 9 «Промисловість, інновації та інфраструктура» (підцілі 9.1 та 9.4).

THE ABSTRACT

Explanatory note: 88 p., 3 table, 17 fig., 3 add., 22 sources.

WEBSOCKET, PYTHON, NOTIFICATION BROKER, MICROSERVICE, WEBSOCKET, TTKLBOOTSTRAP, GUI.

The object of investigation is the process of exchanging data between software modules in a distributed software environment.

The subject of investigation is the architecture and software implementation of a microservice for transmitting information to broker networks based on the WebSocket protocol.

Meta robots – develop a software program (microservice) that will ensure a reliable exchange of structured information between programs in real time, with regulation to scale, stability and integration into automated control systems.

The work describes the process of designing and implementing a microservice system for transferring data between software components. The central element of the architecture is the notification broker, requests based on the WebSocket protocol with the support of asynchronous request processing, client authentication and notification routing.

Client modules have been separated - sensor simulators, monitoring nodes and emergency detectors – which interact with the broker in real time. The JSON notification exchange format has been fixed, a set of topics has been defined and alarm escalation mechanisms have been implemented.

The results of the work are consistent with the Goals of the current development, including Goal 9 “Industry, innovation and infrastructure” (Points 9.1 and 9.4).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

КІТАР – Кафедра комп’ютерно–інтегрованих технологій, автоматизації та робототехніки

ХНУРЕ – Харківський національний університет радіоелектроніки

GUI – Graphical User Interface

WebSocket – Message Queuing Telemetry Transport

TCP/IP – Transmission Control Protocol / Internet Protocol

HMI – Human–Machine Interface

JSON – JavaScript Object Notation

SCADA – Supervisory Control and Data Acquisition

API – Application Programming Interface

QoS – Quality of Service

IDE – Integrated Development Environment

ACK – Acknowledgment

IP – Internet Protocol

ВСТУП

У сучасному світі активного розвитку інформаційних технологій особливу увагу приділяють системам обміну даними між програмами та пристроями. Автоматизовані комплекси, системи Інтернету речей (IoT), промислова автоматизація та смарт–рішення вимагають надійної, швидкої та енергоефективної комунікації між різнорідними компонентами. Забезпечення стабільного обміну інформацією між програмним забезпеченням і фізичними пристроями є критичним для підтримки цілісності, безпеки та функціональності таких систем.

Одним із перспективних напрямів є розроблення мікросервісних рішень, які дозволяють організувати обмін даними без громіздких посередників, із прямим підключенням між пристроями. Застосування підходу прямої взаємодії на базі протоколу TCP/IP відкриває нові можливості для побудови гнучких, масштабованих та оптимізованих систем зв’язку, що не залежать від зовнішніх серверів або брокерів повідомлень.

Об’єкт роботи – процес обміну даними між програмними модулями в розподіленому програмному середовищі.

Предмет роботи – архітектура та програмна реалізація мікросервісу для передачі повідомлень з використанням брокера на основі WebSocket-протоколу.

Мета роботи – розробити програмне забезпечення (мікросервіс), що забезпечує надійний обмін структурованими повідомленнями між програмами у реальному часі, з урахуванням вимог до масштабованості, стабільності та інтеграції в автоматизовані системи управління.

Для досягнення поставленої мети були сформульовані такі завдання:

а) провести аналіз існуючих методів передачі даних між програмами та пристроями;

б) обґрунтувати вибір прямого TCP–з’єднання як базового методу обміну інформацією;

в) розробити програму для мікроконтролера ESP32 для ініціалізації TCP–з’єднання та передавання даних;

г) створити серверну частину у вигляді Python–додатку для прийому і обробки даних;

д) провести тестування працездатності та оцінку ефективності розробленого рішення.

Практична цінність роботи полягає у створенні універсального прикладу прямого обміну даними між різнорідними системами, що може бути використаний для побудови розподілених систем керування, локальних моніторингових систем, прототипів IoT–рішень та інших застосувань у галузі автоматизації.

Крім технологічної доцільності, розробка таких систем тісно пов’язана з глобальними напрямками сталого розвитку, зокрема з цілями, сформульованими в рамках ініціативи ООН «Цілі сталого розвитку» (Sustainable Development Goals, SDGs). Зокрема, запропоноване рішення сприяє досягненню Цілі 9 – "Промисловість, інновації та інфраструктура", яка передбачає розвиток інноваційної інфраструктури, що сприяє індустріалізації та впровадженню новітніх технологій.

Також робота дотична до Цілі 12 "Відповідальне споживання і виробництво", адже побудова енергоефективних та безпечних розподілених систем обміну даними дозволяє зменшити енергоспоживання, оптимізувати виробничі процеси та покращити екологічну стійкість підприємств. Таким чином, проєкт поєднує інженерний підхід із актуальними глобальними викликами, зосередженими на сталому технологічному розвитку.

Таким чином, у межах дипломної роботи буде розроблено сучасне рішення для організації обміну даними, яке відповідатиме вимогам надійності, гнучкості, простоти розгортання та енергоефективності.

Кваліфікаційна робота виконана згідно ДСТУ 3008 – 15 [1] та керуючись навчальним посібником з кваліфікаційної роботи бакалавра [2] та методичними вказівками [3].

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

У сучасних автоматизованих системах важливу роль відіграє якісний обмін даними між програмними модулями, пристроями та зовнішніми інформаційними ресурсами. Від правильності побудови архітектури передачі даних залежить стабільність, масштабованість і ефективність функціонування всієї системи.

Технічне завдання на створення мікросервісу для передачі даних між програмами потребує врахування багатьох факторів – типів середовищ взаємодії (локальні або розподілені), можливих протоколів обміну, вимог до швидкодії, стабільності, а також апаратної основи реалізації рішення.

У цьому розділі виконано аналіз основних проблем передачі даних в автоматизованих системах, здійснено порівняння між можливими апаратними платформами мікроконтролерами (МК) та програмованими логічними контролерами (ПЛК), наведено обґрунтування вибору мікросервісної архітектури та поставлено конкретне завдання на реалізацію програмного рішення.

1.1 Проблематика обміну даними між програмними модулями

У сучасних автоматизованих системах (АС) ефективна передача даних між програмними модулями є ключовим фактором надійності, стабільності та гнучкості всієї системи. Незалежно від масштабів від невеликих локальних обчислювальних систем до комплексних виробничих РТК програми повинні обмінюватися інформацією швидко, надійно й узгоджено.

Серед основних проблем, які виникають при реалізації обміну даними між програмами:

а) різнорідність платформ та протоколів: різні операційні системи, мови програмування та середовища ускладнюють безпосередню взаємодію;

б) відсутність стандартизованого підходу до інтеграції: багато рішень створюються точка-точка (point-to-point), що ускладнює масштабування;

в) надійність і гарантія доставки: втрати даних або їх дублювання можуть призвести до критичних збоїв у системі;

г) безпека обміну: відсутність шифрування та автентифікації загрожує компрометацією інформації;

д) масштабованість: у великих системах необхідно легко додавати нові компоненти без суттєвих змін в архітектурі.

У таких умовах доцільним є застосування архітектури з проміжним посередником обміну даними, наприклад, мікросервісу чи брокера повідомлень. Це дозволяє централізувати та стандартизувати передачу інформації.

На рисунку 1.1 зображено приклад централізованої організації обміну повідомленнями з використанням брокера RabbitMQ. Такий підхід дозволяє ізолювати відправників і отримувачів даних, забезпечити балансування навантаження, збереження повідомлень та гнучку маршрутизацію [1].

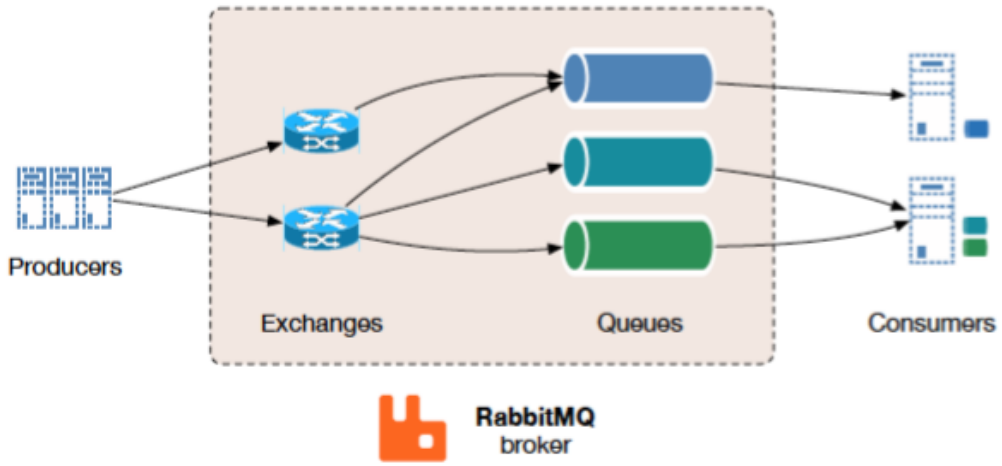

Рисунок 1.1 – Архітектура обміну повідомленнями через RabbitMQ

На рисунку 1.2 зображено приклад мікросервісної архітектури, у якій кожен мікросервіс є окремим незалежним функціональним блоком, що виконує конкретну задачу та має власну базу даних. Всі мікросервіси взаємодіють через спільний користувацький інтерфейс, що забезпечує цілісну логіку роботи

системи. Такий підхід дозволяє досягти гнучкості, масштабованості та високої надійності системи, зменшуючи залежності між її компонентами.

Рисунок 1.2 – Загальна схема мікросервісної архітектури в автоматизованій системі

1.2 Порівняльний аналіз мікроконтролерів (МК) та програмованих логічних контролерів (ПЛК)

При побудові систем обміну даними в автоматизованих середовищах критичне значення має правильний вибір апаратної платформи. На сьогоднішній день розробники мають широкий вибір пристроїв, серед яких найбільш популярними є мікроконтролери (МК) та програмовані логічні контролери (ПЛК). У цьому підрозділі буде розглянуто ключові відмінності між цими двома платформами, їх особливості, сфери застосування, переваги та недоліки (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика мікроконтролерів та ПЛК

Критерій	Мікроконтролер (ESP32)	ПЛК (Siemens S7–1200)
Сфера застосування	Прототипи, IoT, освітні проєкти	Промислова автоматизація

Продовження таблиці 1.1

Критерій	Мікроконтролер (ESP32)	ПЛК (Siemens S7-1200)
Вартість	Низька (від 5 до 20 USD)	Висока (від 150 до 800+ USD)
Інтерфейси зв’язку	UART, SPI, I2C, Wi-Fi, Bluetooth	RS-232/485, Ethernet, Profinet, Modbus TCP
Надійність	Середня	Висока (промисловий стандарт)
Програмування	C++, MicroPython, Arduino IDE	Step 7, TIA Portal, LAD, STL
Гнучкість розробки	Висока, з відкритими бібліотеками	Середня, залежить від моделі ПЛК
Енергоспоживання	Дуже низьке (до 250 мВт)	Високе, залежно від конфігурації
Підтримка сенсорів/модулів	Широкий вибір модулів з відкритим доступом	Переважно сертифіковані, часто дорогі модулі
Масштабованість	Обмежена	Дуже висока, промисловий рівень
Інструменти розробника	Багато онлайн-ресурсів і спільнот	Професійні середовища з ліцензіями

Мікроконтролери – це компактні, універсальні мікропроцесорні системи, які дозволяють реалізувати як прості, так і досить складні проєкти, зокрема на базі популярних платформ ESP32, Arduino, Raspberry Pi Pico тощо. Ці рішення активно використовуються в сфері IoT, розумних будинків, прототипування, автоматизації побутових процесів [2].

ПЛК – це більш спеціалізовані пристрої, які орієнтовані на надійне функціонування у важких умовах промислового середовища. Вони мають високий рівень захисту від пилу, вологи, електромагнітних завад, а також підтримують специфічні протоколи зв’язку (Modbus, Profibus, Profinet), інтеграцію з SCADA-системами та інше [3].

На рисунку 1.3 зображено плату ESP32 D1 R32 – одну з найпопулярніших мікроконтролерних платформ серед розробників рішень для Інтернету речей. Основою мікроконтролера є чип ESP-WROOM-32, який має два процесорних ядра, вбудований Wi-Fi та Bluetooth, а також набір цифрових і аналогових входів/виходів. Дана плата сумісна з Arduino IDE, що значно полегшує її програмування навіть для новачків. На зображенні можна побачити USB-порт для прошивки, кнопку скидання, розширення під модулі, стабілізатори напруги та інші електронні компоненти, які забезпечують стабільну роботу [4].

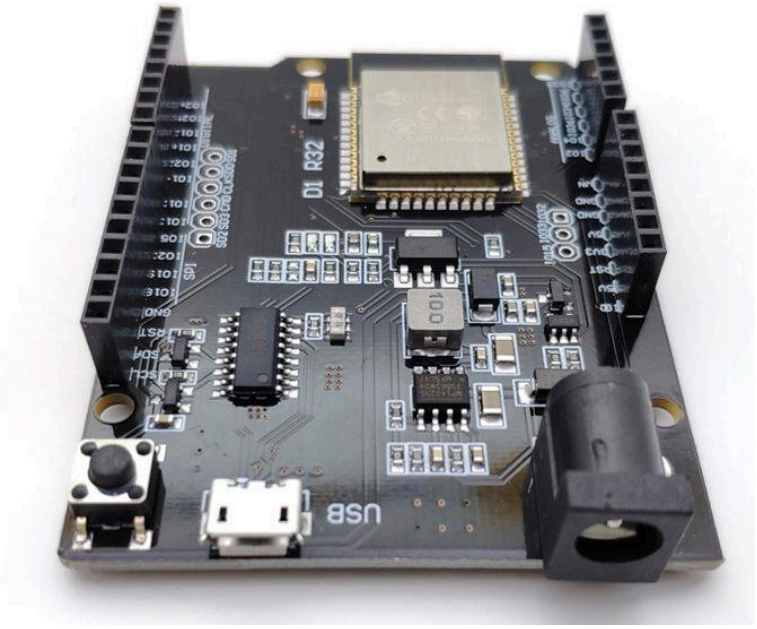

Рисунок 1.3 – Мікроконтролер ESP32 D1 R32

На рисунку 1.4 зображено програмований логічний контролер Siemens S7-1200, зокрема модель CPU 1211C DC/DC/DC. Це промисловий пристрій, розроблений для роботи в умовах підвищеної надійності, де потрібне безперервне функціонування та підтримка складних технологічних процесів. S7-1200 оснащено цифровими входами та виходами, підтримкою розширення модулів, а також можливістю підключення через Ethernet. Контролер програмується в середовищі TIA Portal, що дозволяє створювати гнучкі та масштабовані автоматизовані рішення.

Цей ПЛК широко застосовується в харчовій промисловості, машинобудуванні, логістичних системах, де критичною є стабільність, сертифіковані протоколи обміну (Modbus, Profinet) та інтеграція з SCADA/ERP-системами [5].

Рисунок 1.4 – Програмований логічний контролер Siemens S7–1200

1.3 Обґрунтування вибору мікросервісної архітектури

У процесі розробки автоматизованих систем важливою складовою є архітектура програмного забезпечення, оскільки саме вона визначає масштабованість, гнучкість, стабільність та керованість рішення. У рамках цієї виробничої практики було обрано мікросервісну архітектуру для реалізації системи передачі даних між різними програмними модулями.

Мікросервісна архітектура – це підхід до побудови програмного забезпечення, при якому застосунок складається з набору невеликих, незалежно розгортаних служб (сервісів) [8], кожна з яких виконує чітко визначену функцію та взаємодіє з іншими через легкі протоколи обміну (наприклад, HTTP, WebSocket) [9]. Такий підхід контрастує з традиційною монолітною архітектурою, де всі функціональні частини програми зібрані в одному

кодовому блоці [6].

На рисунку 1.5 зображено схему, яка ілюструє відмінність між монолітною та мікросервісною архітектурами. Ліворуч показано моноліт, у якому користувацький інтерфейс, бізнес-логіка та доступ до даних об’єднані в єдину систему, що ускладнює масштабування, оновлення та відлагодження. Праворуч демонструється мікросервісна модель, де кожна складова працює незалежно, має власну базу даних або сервіс і взаємодіє з іншими за допомогою API. Це дозволяє паралельно оновлювати сервіси, змінювати логіку без зупинки всієї системи, а також підвищує надійність і відмовостійкість.

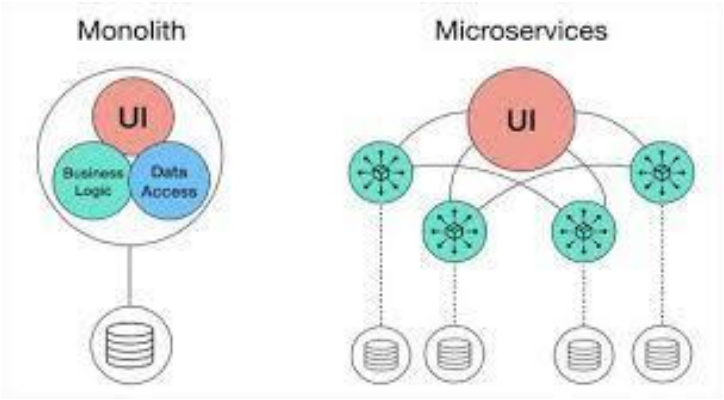

Рисунок 1.5 – Порівняння монолітної та мікросервісної архітектур

Ключовим критерієм при виборі архітектури стала потреба в гнучкому обміні даними між програмами, які можуть бути розгорнуті на різних пристроях, мати різну логіку роботи, але потребують централізованої передачі повідомлень. З огляду на це, в якості протоколу взаємодії був обраний WebSocket (Message Queuing Telemetry Transport), що є легким, ефективним і спеціально розробленим для обміну даними в умовах обмежених ресурсів (наприклад, на мікроконтролерах ESP32).

На рисунку 1.6 представлено приклад практичного застосування мікросервісної архітектури з використанням WebSocket-брокера Mosquitto [7]. У системі використовується декілька WebSocket-клієнтів, які взаємодіють із брокером, публікуючи та підписуючись на певні топіки (канали передачі даних).

Один з клієнтів, дозволяє реалізувати логіку візуального програмування. Другий клієнт – це контролер ESP32, який або зчитує значення температури/вологості з датчиків, або керує світлодіодом, отримуючи команди через WebSocket.

Программа виступає як клієнт, що надсилає або отримує повідомлення через брокер Mosquitto, встановлений на одноплатному комп'ютері (наприклад, Raspberry Pi). ESP32, як інший WebSocket-клієнт, реагує на отримані повідомлення – керує світлодіодом або відправляє значення температури. Така схема добре демонструє розділення функцій, характерне для мікросервісної архітектури: кожен пристрій виконує окрему роль, а брокер координує комунікацію між ними.

Таким чином, обрана мікросервісна архітектура з використанням WebSocket забезпечує:

а) гнучкість інтеграції нових компонентів системи;

б) незалежність розробки та розгортання кожного сервісу;

в) мінімальні вимоги до ресурсів, що важливо для мікроконтролерів;

г) масштабованість, що дозволяє згодом розширити систему без потреби в повному перепроєктуванні;

д) високу надійність та резервування, особливо у випадках використання брокерів з підтримкою QoS.

Такий підхід дозволяє досягти сучасного рівня організації обміну даними в автоматизованих системах і є повністю виправданим для поставленого технічного завдання.

1.4 Постановка завдання на розробку програмного забезпечення

У межах даної роботи розробляється система моніторингу температури та вологості на основі мікроконтролера ESP32, змодельованого у середовищі Wokwi, та настільного застосунку, створеного мовою Python. Взаємодія між пристроями реалізована через протокол WebSocket для обміну повідомленнями.

Мікроконтролер ESP32 виконує функцію збору даних за допомогою сенсора DHT22, аналізує показники, визначає поточний стан (нормальний, попередження, тривога), відображає його на OLED-дисплеї та передає інформацію через WebSocket. У разі перевищення критичних значень активуються сигнальні світлодіоди. У відповідь на певні події пристрій також може отримувати команди, зокрема на активацію охолодження шляхом відкриття сервоприводу, що моделює подачу води.

Стаціонарна частина системи реалізована у вигляді програми, яка відображає графік отриманих значень температури та вологості, показує статус системи у вигляді світлових індикаторів та дозволяє керувати виконавчим пристроєм за допомогою інтерактивної кнопки. Інтерфейс користувача створено з використанням бібліотеки ttkbootstrap, а графічна візуалізація реалізована на базі matplotlib. Дані зберігаються у файл Excel після кожних двадцяти отриманих повідомлень, що дозволяє формувати історію спостережень та переглядати її за потреби.

Програмне забезпечення розробляється з урахуванням можливості подальшого розширення, підтримки кількох пристроїв та стабільної роботи в умовах змінного мережевого середовища. У цьому розділі визначається логіка роботи обох частин системи, обґрунтовується вибір інструментів, описується архітектура взаємодії та ставиться завдання реалізувати гнучкий, надійний і зручний для користувача засіб моніторингу й реагування.

2 ПРОЄКТУВАННЯ МЕРЕЖЕВОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура системи з використанням WebSocket–брокера та клієнтів різних вузлах

Розроблена промислова система обміну даними побудована на основі централізованої архітектури з використанням WebSocket–протоколу для забезпечення надійної та швидкої комунікації між компонентами. Основою системи виступає брокер повідомлень, який функціонує як центральний вузол для координації взаємодії між усіма підключеними клієнтами.

Архітектурне рішення реалізує концепцію "зіркової" топології, де брокер розташований у центрі мережі та обслуговує чотири типи клієнтів: датчик температури, два монітори температури та обробник алармів. Така організація забезпечує максимальну гнучкість системи, оскільки кожен клієнт може незалежно підключатися та відключатися від брокера без впливу на роботу інших компонентів.

Центральний брокер, розміщений на сервері з IP–адресою 192.168.1.10, виконує функції приймання, фільтрації та маршрутизації повідомлень між клієнтами. Брокер підтримує систему топіків (каналів повідомлень), що дозволяє реалізувати селективну доставку інформації тільки тим клієнтам, які підписані на відповідні канали. Основними топіками в системі є "temperature" для передачі даних з датчиків та "temperature–alarm" для сповіщень про аварійні ситуації.

Датчик температури, реалізований на клієнті з адресою 192.168.1.101, функціонує як постачальник первинних даних для системи. Цей компонент періодично генерує показання температури та відправляє їх до брокера через топік "temperature". Датчик підтримує як автоматичний режим роботи з налаштовуваним інтервалом передачі даних, так і ручне керування для проведення тестових сценаріїв. Реалізована можливість точного встановлення

температурних значень дозволяє імітувати різноманітні виробничі ситуації, включаючи аварійні стани.

Система моніторингу представлена двома незалежними клієнтами, розташованими на адресах 192.168.1.102 та 192.168.1.103. Кожен монітор підписується на топік "temperature" та отримує дані від датчика через брокер. Монітори виконують аналіз отриманих показань на відповідність встановленим пороговим значенням. Система підтримує три рівні алармування: LOW для низьких температур, HIGH для підвищених температур та CRITICAL для критично високих значень. При виявленні порушення порогових меж монітор автоматично генерує аларм та відправляє його до брокера через топік "temperature–alarm".

Обробник алармів, розміщений на клієнті 192.168.1.104, виступає кінцевим елементом ланцюга реагування на аварійні ситуації. Цей компонент підписується виключно на топік "temperature–alarm" та отримує сповіщення від моніторів при виникненні нештатних ситуацій. Обробник забезпечує візуальну та звукову сигналізацію за допомогою кольорових індикаторів, що імітують промислові сигнальні лампи. Система підтримує різні рівні пріоритетності алармів з відповідним кольоровим кодуванням та звуковими сигналами.

Комунікаційний протокол системи базується на WebSocket–з'єднаннях, що забезпечує двосторонній обмін даними в реальному часі з мінімальними затримками. Всі повідомлення структуровані у форматі JSON та містять метадані про відправника, тип повідомлення, корисне навантаження та часові мітки. Брокер виконує автентифікацію клієнтів за допомогою паролю та підтримує систему підписок, що дозволяє ефективно фільтрувати трафік [13].

Архітектура системи забезпечує високий рівень масштабованості, оскільки додавання нових датчиків або моніторів не вимагає модифікації існуючих компонентів. Централізований підхід до маршрутизації повідомлень дозволяє легко впроваджувати нові типи обладнання та розширювати функціональність системи. Відмовостійкість забезпечується незалежністю

компонентів та можливістю автоматичного відновлення з'єднань при тимчасових збоях мережі.

На рисунку 2.1 представлено схему мережевої топології розробленої системи, яка демонструє фізичне розташування компонентів у локальній мережі. Сервер брокера розміщено у центрі топології та з'єднано з комутатором, який забезпечує підключення всіх клієнтських вузлів. Така організація відповідає промисловим стандартам побудови автоматизованих систем та забезпечує ефективний обмін даними між усіма учасниками процесу моніторингу.

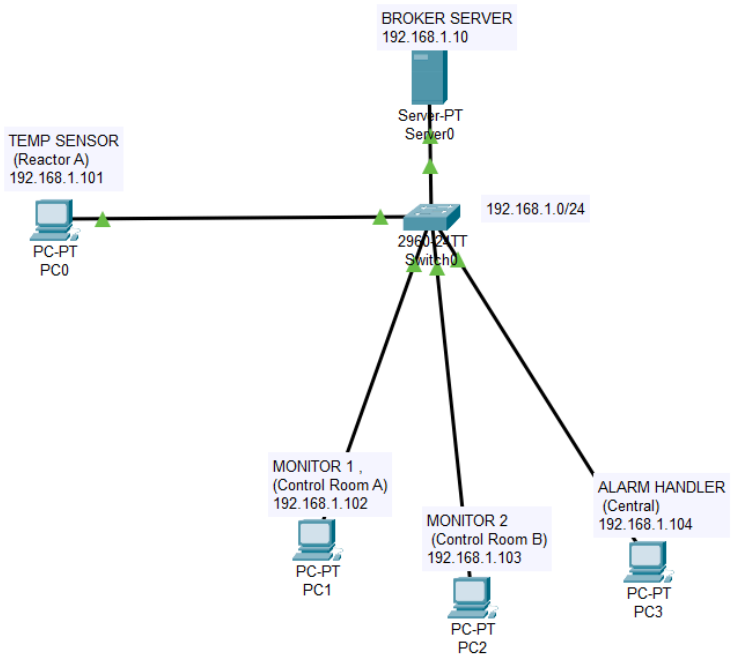

Рисунок 2.1 – Мережева топологія промислової системи обміну даними з WebSocket–брокером

2.2 Розробка структури повідомлень, форматів даних та каналів взаємодії

Ефективна взаємодія між компонентами промислової системи вимагає чітко визначеної структури повідомлень та стандартизованих форматів обміну

даними. Розроблена система використовує JSON (JavaScript Object Notation) як основний формат серіалізації даних, що забезпечує читабельність, гнучкість та широку підтримку в різних програмних середовищах. Структура повідомлень спроєктована з урахуванням вимог промислової автоматизації, включаючи потреби в точному часовому маркуванні, ідентифікації джерел даних та забезпеченні цілісності інформації [21].

Базова структура всіх повідомлень в системі включає обов'язкові метадані, які забезпечують правильну ідентифікацію та маршрутизацію інформації. Кожне повідомлення містить поля "type" для визначення типу даних, "sender" для ідентифікації відправника, "payload" для корисного навантаження та "timestamp" для часового маркування. Така уніфікована структура дозволяє брокеру ефективно обробляти різнотипні повідомлення та забезпечує консистентність протоколу взаємодії між усіма компонентами системи.

Для автентифікації клієнтів використовується спеціальний тип повідомлень "auth", який передається при встановленні з'єднання з брокером. Структура автентифікаційного повідомлення включає поля "client_name" для ідентифікації клієнта, "password" для перевірки прав доступу та "protocol_version" для забезпечення сумісності версій протоколу. Брокер відповідає повідомленням типу "auth_response" з полем "success", що вказує на результат автентифікації, та "message" з текстовим описом статусу підключення.

Система підписок реалізована через повідомлення типу "subscribe" та "unsubscribe", які дозволяють клієнтам селективно отримувати інформацію з певних топіків. Повідомлення підписки містить поле "topic" з назвою каналу, на який клієнт хоче підписатися. Брокер підтверджує підписку відповіддю "subscribe_response" з індикатором успішності операції та списком активних підписок клієнта. Така архітектура забезпечує ефективне використання мережевих ресурсів та дозволяє клієнтам отримувати тільки релевантну інформацію.

Передача даних від датчиків здійснюється через повідомлення типу "publish" з вкладеною структурою "message", що містить фактичні дані сенсора.

Для температурних датчиків корисне навантаження включає поля "sensor_id" для унікальної ідентифікації пристрою, "location" для географічної або технологічної прив'язки, "value" для числового значення вимірювання, "unit" для одиниць вимірювання та "quality" для індикації якості даних. Додатково кожне повідомлення містить "reading_number" для порядкової нумерації вимірювань та "measurement_time" для точного часу проведення вимірювання.

Структура температурних даних спроєктована для підтримки різних типів сенсорів та одиниць вимірювання. Поле "value" передається як число з плаваючою комою з точністю до двох знаків після коми, що забезпечує достатню точність для більшості промислових застосувань. Поле "unit" підтримує стандартні одиниці температури: "celsius", "fahrenheit" та "kelvin". Індикатор якості "quality" може приймати значення "good", "bad", "uncertain", "maintenance" або "offline", що дозволяє моніторинговим системам правильно інтерпретувати надійність отриманих даних.

Алармові повідомлення використовують спеціалізовану структуру, розроблену для передачі критично важливої інформації про порушення технологічних параметрів. Повідомлення типу "temperature-alarm" включає розширений набір метаданих для забезпечення повного контексту аварійної ситуації. Основні поля включають "monitor_id" та "monitor_location" для ідентифікації джерела аларму, "sensor_id" та "sensor_location" для вказівки на проблемний датчик, "alarm_level" для рівня критичності та "current_value" з "threshold_exceeded" для числових характеристик порушення.

Система підтримує чотири рівні алармування, що відповідають промисловим стандартам: "NORMAL" для нормального стану, "LOW" для значень нижче мінімального порогу, "HIGH" для перевищення робочих меж та "CRITICAL" для критично небезпечних ситуацій. Кожен аларм отримує унікальний ідентифікатор "alarm_id" та містить описове повідомлення "message" для операторів. Поля "acknowledged" та "escalated" забезпечують відстеження стану обробки аларму системами диспетчеризації.

Канали взаємодії організовані за принципом тематичних топіків, що дозволяє гнучко керувати потоками інформації в системі. Основний топік "temperature" використовується для передачі даних від усіх температурних датчиків до моніторингових систем. Топік "temperature-alarm" призначений для алармових повідомлень від моніторів до систем оповіщення. Додаткові топіки можуть бути створені для інших типів сенсорів або службових повідомлень без змін в архітектурі брокера.

Система також підтримує службові повідомлення для діагностики та контролю стану з'єднань. Повідомлення типу "ping" дозволяють клієнтам перевіряти доступність брокера, а відповіді "pong" підтверджують активність каналу зв'язку. Запити статусу "status" повертають інформацію про стан брокера, включаючи кількість підключених клієнтів, активні топіки та статистику обробки повідомлень.

Для забезпечення надійності передачі всі повідомлення включають механізми контролю цілісності та відновлення після збоїв. Часові мітки генеруються в форматі ISO 8601 з точністю до мілісекунд, що забезпечує синхронізацію подій в розподіленій системі. Поля розміру повідомлень та контрольних сум можуть бути додані для критично важливих даних.

Структура повідомлень розроблена з урахуванням майбутнього розширення системи. Використання JSON дозволяє легко додавати нові поля без порушення сумісності з існуючими клієнтами. Версіонування протоколу забезпечує еволюцію системи зі збереженням підтримки старих клієнтів. Модульна архітектура топіків дозволяє інтегрувати нові типи обладнання та функціональності.

Оптимізація розміру повідомлень досягається через використання коротких назв полів та ефективне кодування числових значень. Для великих обсягів даних система підтримує стиснення повідомлень та пакетну передачу множини вимірювань в одному повідомленні. Такий підхід забезпечує ефективне використання мережевої пропускної здатності при збереженні читабельності та гнучкості протоколу.

У таблиці 2.1 наведено детальний опис основних типів повідомлень системи з їх структурою та призначенням.

Таблиця 2.1 – Структура повідомлень системи обміну даними

Тип повідомлення	Призначення	Обов'язкові поля	Приклад payload
auth	Автентифікація клієнта	client_name, password, protocol_version	{"client_name": "TEMP_SENSOR_001", "password": "demo2024", "protocol_version": "1.0"}
auth_response	Відповідь на автентифікацію	success, message	{"success": true, "message": "Welcome, TEMP_SENSOR_001!"}
subscribe	Підписка на топік	topic	{"topic": "temperature"}
subscribe_response	Підтвердження підписки	success, topic, message	{"success": true, "topic": "temperature", "message": "Subscribed successfully"}
publish	Публікація даних датчика	sensor_id, location, value, unit, quality, timestamp	{"sensor_id": "TEMP_SENSOR_001", "location": "Reactor A", "value": 25.5, "unit": "celsius", "quality": "good", "timestamp": "2025–05–24T14:30:45.123Z"}
temperature–alarm	Алармове повідомлення	monitor_id, sensor_id, alarm_level, current_value, threshold_exceeded, alarm_id, message	{"monitor_id": "TEMP_MONITOR_001", "sensor_id": "TEMP_SENSOR_001", "alarm_level": "CRITICAL", "current_value": 95.5, "threshold_exceeded": 95.0, "alarm_id": "ALARM_001", "message": "Critical temperature!"}
ping	Перевірка з'єднання	message, client	{"message": "ping", "client": "TEMP_SENSOR_001"}

Продовження таблиці 2.1

Тип повідомлення	Призначення	Обов'язкові поля	Приклад payload
pong	Відповідь на ping	respond_to	{"respond_to": "TEMP_SENSOR_001"}
status	Запит статусу брокера	–	{}
status_response	Статус брокера	uptime_seconds, connected_clients, total_messages, active_topics	{"uptime_seconds": 3600, "connected_clients": 4, "total_messages": 1250, "active_topics": ["temperature", "temperature-alarm"]}

2.3 Технологічна база реалізації: Python, WebSockets, асинхронна обробка, GUI для операторів

Для реалізації системи оперативного зв'язку обрано сучасний технологічний стек на базі Python, який забезпечує високу продуктивність, масштабованість та надійність. Основу архітектури складають WebSockets для забезпечення реального часу передачі даних, асинхронна обробка для ефективного управління ресурсами та спеціалізований графічний інтерфейс для операторів.

Python обрано як основну мову програмування завдяки його потужним бібліотекам для роботи з мережевими протоколами, медіаконтентом та асинхронним програмуванням [8]. Для серверної частини використовується бібліотека websockets, яка забезпечує високопродуктивну обробку WebSocket-з'єднань з підтримкою асинхронних операцій [14]. Реалізація центрального брокера повідомлень демонструє використання сучасних Python-патернів для обробки множинних клієнтських з'єднань:

```
import asyncio
import websockets
import json
from datetime import datetime
from typing import Dict, Set, Any

class MessageBroker:
    def __init__(self, host='localhost', port=8765, password='broker123'):
        self.host = host
        self.port = port
        self.password = password
        self.clients: Dict[websockets.WebSocketServerProtocol, Dict[str, Any]] =
{}
        self.subscriptions: Dict[str, Set[websockets.WebSocketServerProtocol]] =
{}

    async def handle_client(self, websocket:
websockets.WebSocketServerProtocol):
        """Основний обробник клієнта"""
        try:
            async for message in websocket:
                data = json.loads(message)
                await self._process_message(websocket, data)
        except websockets.exceptions.ConnectionClosed:
            await self._cleanup_client(websocket)
```

WebSocket-протокол забезпечує двосторонню комунікацію між клієнтами через централізований брокер повідомлень. На відміну від традиційного HTTP, WebSocket підтримує постійне з'єднання, що критично важливо для систем реального часу. Система включає механізми аутентифікації клієнтів, систему

підписок на топіки та гарантовану доставку повідомлень. Процес аутентифікації реалізовано з використанням хешування паролів та генерації сесійних токенів:

```
    async def _handle_auth(self, websocket:
websockets.WebSocketServerProtocol,
                           data: Dict[str, Any]) –> bool:
        """Обробка аутентифікації клієнта"""
        password = data.get('password', '')
        client_name = data.get('client_name', 'Unknown')

        if self._authenticate(password):
            self.clients[websocket] = {
                'name': client_name,
                'authenticated': True,
                'connected_at': datetime.now(),
                'subscriptions': set()
            }

            await self._send_message(websocket, {
                'type': 'auth_response',
                'success': True,
                'message': f'Welcome, {client_name}!'
            })
            return True
        return False
```

Система побудована на базі асинхронної архітектури з використанням бібліотеки asyncio, що дозволяє ефективно обробляти тисячі одночасних з'єднань без блокування потоків виконання. Кожен клієнт обслуговується незалежно через власну корутину, а брокер координує взаємодію через

централізований event loop. Асинхронна розсилка повідомлень реалізована з використанням gather для паралельного виконання операцій відправки:

```
async def _broadcast_to_subscribers(self, topic: str, message: Dict[str, Any]):
    """Відправка повідомлення всім підписникам топіка"""
    if topic not in self.subscriptions:
        return

    subscribers = self.subscriptions[topic].copy()

    # Асинхронна відправка всім підписникам
    tasks = []
    for subscriber in subscribers:
        if subscriber in self.clients:
            task = self._send_message(subscriber, {
                'type': 'message',
                'topic': topic,
                'data': message,
                'timestamp': datetime.now().isoformat()
            })
            tasks.append(task)

    # Паралельне виконання всіх операцій відправки
    await asyncio.gather(*tasks, return_exceptions=True)
```

Для створення сучасного та функціонального GUI використовується бібліотека ttkbootstrap, яка надає Bootstrap-подібні стилі для Tkinter. Інтерфейс розроблено з урахуванням промислових стандартів і забезпечує зручну взаємодію оператора з системою. Візуальні індикатори стану системи

реалізовано з використанням кольорового кодування та динамічних ефектів для привернення уваги до критичних ситуацій:

```
import ttkbootstrap as ttk
from ttkbootstrap.constants import *

def _create_alarm_indicators(self):
    """Створення візуальних індикаторів алармів"""
    indicators_frame = ttk.LabelFrame(self.control_frame,
                                      text="Індикатори алармів", padding=15)
    indicators_frame.pack(fill=X, pady=(0, 15))

    # Критичний аларм з кольоровою індикацією
    critical_frame = ttk.Frame(indicators_frame)
    critical_frame.pack(side=LEFT, padx=(0, 30))

    ttk.Label(critical_frame, text="КРИТИЧНИЙ",
            font=("Arial", 10, "bold")).pack()
    self.alarm_lights["CRITICAL"]["widget"] = tk.Label(
        critical_frame, text="●", font=("Arial", 24),
        fg="gray", bg=self.root.cget('bg'))
    self.alarm_lights["CRITICAL"]["widget"].pack()
```

Система реалізує спеціалізовані структури даних для промислового застосування, включаючи підтримку якості даних, часових міток та метаданих сенсорів. Використовується статична типізація Python для забезпечення надійності коду та раннього виявлення помилок. Створення показань датчиків включає валідацію даних та додавання промислових метаданих:

```
from typing import Dict, Any, Optional
```

```
from datetime import datetime
from enum import Enum

class DataQuality(Enum):
    GOOD = "good"
    BAD = "bad"
    UNCERTAIN = "uncertain"
    MAINTENANCE = "maintenance"
    OFFLINE = "offline"

def create_sensor_reading(sensor_id: str, location: str, value: float,
                          unit: str, quality: str = DataQuality.GOOD.value) –> Dict[str,
Any]:
    """Створення показання датчика з промисловими метаданими"""
    return {
        "sensor_id": sensor_id,
        "location": location,
        "value": value,
        "unit": unit,
        "quality": quality,
        "timestamp": datetime.now().isoformat(),
        "measurement_time": datetime.now().isoformat()
    }
```

У таблиці 2.3 зображено технологічну базу та компоненти системи.

Таблиця 2.3 – Технологічна база та компоненти системи

Компонент	Технологія	Версія	Призначення	Особливості
Мова програмування	Python	3.8+	Основна платформа розробки	Асинхронність, типізація, бібліотеки

Продовження таблиці 2.3

Компонент	Техно-логія	Версія	Призначення	Особливості
WebSocket сервер	websockets	12.0+	Мережева комунікація	Низька затримка, постійні з'єднання
Асинхрон–ність	asyncio	Стандартна	Обробка concurrent операцій	Event loop, coroutines, tasks
GUI фреймворк	ttkbootstrap	1.10+	Графічний інтерфейс	Bootstrap стилі, теми, responsive
Серіалізація	JSON	Стандартна	Формат повідомлень	Легкий, читабельний, стандартний
Логування	logging	Стандартна	Моніторинг та діагностика	Рівні, ротація, форматування
Валідація	typing	Стандартна	Типізація та валідація	Static typing, runtime checks
Конфігурація	JSON файли	–	Налашту-вання системи	Гнучкість, версіонування
Процеси	subprocess	Стандартна	Запуск компонентів	Ізоляція, управління життєвим циклом

Система побудована за принципами модульної архітектури з чіткою separation of concerns. Базовий клієнт BaseClient надає загальну функціональність, яка успадковується спеціалізованими клієнтами для різних промислових застосувань. Такий підхід забезпечує легке додавання нових типів датчиків, моніторів або обробників без внесення змін в основну архітектуру системи.

3 ІМПЛЕМЕНТАЦІЯ ПРОМИСЛОВОЇ СИСТЕМИ ОБМІНУ ПОДІЯМИ

3.1 Реалізація брокера повідомлень і підтримка мережевої взаємодії

Центральним компонентом розробленої промислової системи є брокер повідомлень, який виконує функції координації та маршрутизації даних між усіма підключеними клієнтами. Як показано на рисунку 3.1, архітектура брокера побудована навколо центрального WebSocket-сервера з шістьма ключовими функціональними модулями, кожен з яких відповідає за специфічні аспекти роботи системи. Така модульна організація забезпечує високу надійність, масштабованість та можливість незалежного розвитку компонентів системи.

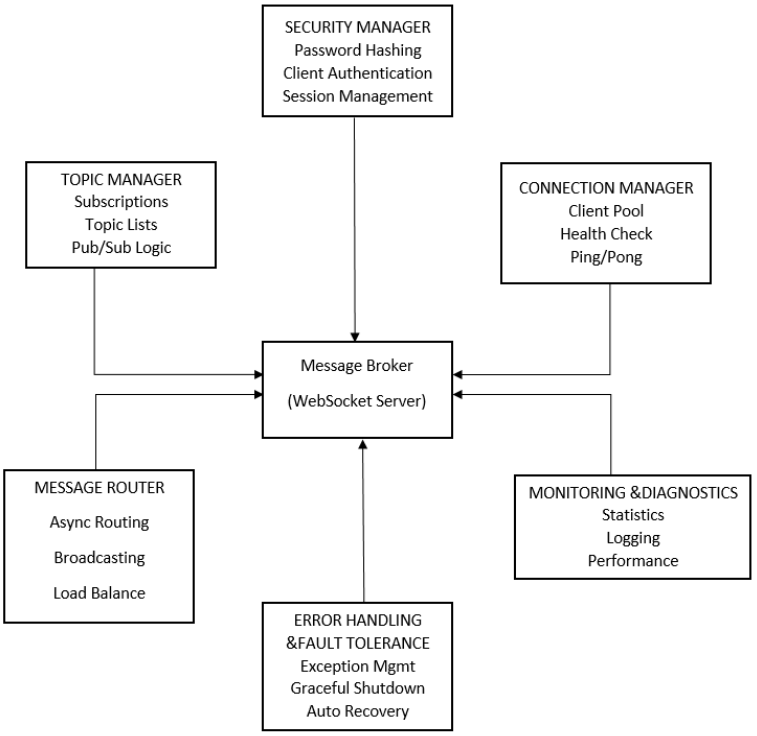

Рисунок 3.1 – Архітектура брокера повідомлень та його функціональні модулі

Security Manager займає провідну позицію в архітектурі брокера, забезпечуючи комплексну систему захисту від несанкціонованого доступу та витоків інформації. Цей модуль реалізує трирівневу систему безпеки, яка включає автентифікацію клієнтів, управління сесіями та криптографічний

захист паролів. Password Hashing здійснюється з використанням алгоритму SHA-256 з додаванням унікальних salt-значень для кожного клієнта, що унеможливлює атаки типу rainbow table та забезпечує стійкість до bruteforce-атак.

Client Authentication виконується на етапі встановлення WebSocket-з'єднання через обмін спеціальними повідомленнями типу "auth", які містять зашифровані облікові дані та метадані клієнта. Після успішної перевірки облікових даних Security Manager генерує унікальний токен сесії та встановлює часові обмеження для активності клієнта. Session Management підтримує реєстр всіх активних сесій з можливістю їх примусового завершення при виявленні підозрілої активності або перевищенні лімітів бездіяльності.

Connection Manager відповідає за управління життєвим циклом WebSocket-з'єднань та забезпечення їх стабільної роботи в умовах нестабільної мережі. Модуль Client Pool реалізує ефективний пул з'єднань з автоматичним масштабуванням відповідно до навантаження системи. Кожне з'єднання представлено як окремий об'єкт з повним набором метаданих, включаючи IP-адресу клієнта, часові мітки підключення, статистику переданих повідомлень та поточний стан з'єднання.

Health Check функціонує як автоматизована система моніторингу доступності клієнтів через регулярну відправку ping-повідомлень та аналіз відповідей. Інтервал перевірки налаштовується залежно від критичності клієнта та вимог до відзивності системи. Ping/Pong механізм реалізовано відповідно до специфікації WebSocket RFC 6455 з додатковими розширеннями для передачі діагностичної інформації про стан мережевого каналу та навантаження клієнта.

Topic Manager керує складною системою підписок та розподілу повідомлень між клієнтами відповідно до їх інтересів та ролей у системі. Subscriptions підсистема підтримує динамічний реєстр підписок з можливістю real-time модифікації без перезапуску брокера. Кожна підписка містить не тільки ідентифікатор топіка, але й додаткові параметри фільтрації, пріоритет доставки та налаштування QoS (Quality of Service).

Topic Lists організовані як ієрархічна структура з підтримкою wildcards та регулярних виразів для гнучкого управління потоками даних. Pub/Sub Logic реалізує класичну парадигму publisher-subscriber з розширеннями для промислового застосування, включаючи guaranteed delivery, message ordering та conflict resolution при одночасних публікаціях в один топік.

Message Router є серцем системи обробки повідомлень, що забезпечує ефективну доставку інформації від відправників до отримувачів з мінімальними затримками. Async Routing реалізований з використанням корутин Python та забезпечує паралельну обробку тисяч повідомлень без блокування головного циклу обробки подій. Маршрутизація здійснюється на основі аналізу заголовків повідомлень, топіків призначення та поточного стану підписників.

Broadcasting функціонує як оптимізований механізм групової розсилки з підтримкою multicast-паттернів для зменшення мережевого трафіку при передачі однакових повідомлень множині отримувачів. Load Balance забезпечує рівномірний розподіл навантаження між різними обробниками повідомлень та автоматичне перенаправлення трафіку при перевантаженні окремих компонентів системи.

Модуль Monitoring & Diagnostics забезпечує комплексне спостереження за роботою всіх компонентів брокера та збір детальної телеметрії для аналізу продуктивності. Statistics підсистема збирає та агрегує дані про кількість підключених клієнтів, обсяги переданого трафіку, частоту помилок та час відгуку системи. Ці метрики зберігаються в кільцевих буферах для ефективного використання пам'яті та швидкого доступу до історичних даних.

Logging реалізований як багаторівнева система з підтримкою різних форматів виводу та автоматичною ротацією файлів логів. Система підтримує конфігуровані рівні логування від DEBUG до CRITICAL з можливістю динамічної зміни деталізації без перезапуску сервера. Performance модуль здійснює continuous profiling критичних секцій коду та автоматично виявляє bottlenecks у продуктивності системи.

Error Handling & Fault Tolerance модуль забезпечує стабільну роботу системи навіть в умовах часткових відмов компонентів або мережевих збоїв. Exception Management реалізує багаторівневу систему обробки винятків з автоматичною класифікацією помилок за ступенем критичності та визначенням оптимальних стратегій відновлення. Кожен тип винятку має власний набір процедур обробки, від простого логування до повного перезапуску компонента.

Graceful Shutdown забезпечує коректне завершення роботи брокера при отриманні сигналів операційної системи або команд адміністратора. Процедура включає збереження критичних даних, повідомлення клієнтів про припинення роботи та координоване закриття всіх активних з'єднань. Auto Recovery функціонує як система автоматичного відновлення після збоїв з підтримкою checkpoint-механізмів для швидкого повернення до операційного стану.

Всі модулі брокера інтегровані через систему внутрішніх API та event bus, що забезпечує loose coupling між компонентами та можливість незалежного розвитку кожного модуля. WebSocket Server виступає як центральний координатор, який делегує специфічні завдання відповідним модулям та агрегує результати їх роботи. Така архітектура дозволяє легко додавати нові функціональні модулі або замінювати існуючі без впливу на роботу інших компонентів системи.

Система підтримує hot-reload конфігурацій та динамічне оновлення окремих модулів без припинення обслуговування клієнтів. Це критично важливо для промислових систем, де простої недопустимі та потрібна можливість оперативного реагування на зміни у виробничих процесах.

3.2 Розробка клієнтів: симулятор датчика, вузли та логер подій

Клієнтська частина промислової системи складається з трьох основних типів компонентів, кожен з яких виконує специфічні функції в загальному процесі моніторингу та управління технологічними параметрами. Архітектура клієнтів побудована на основі базового класу BaseClient, який забезпечує

уніфіковану функціональність для роботи з брокером повідомлень, включаючи автентифікацію, управління підписками та обробку помилок.

BaseClient інкапсулює загальну логіку взаємодії з брокером через WebSocket–протокол та надає стандартизований інтерфейс для всіх типів клієнтів системи. Клас містить механізми автоматичного відновлення з'єднання при мережевих збоях, систему черг повідомлень для забезпечення гарантованої доставки та конфігуровані таймаути для різних типів операцій. Кожен спеціалізований клієнт успадковує цю функціональність та розширює її специфічними можливостями відповідно до свого призначення [9].

Графічний інтерфейс користувача реалізований з використанням бібліотеки ttkbootstrap, яка забезпечує сучасний вигляд та відзивність інтерфейсу відповідно до промислових стандартів. Кожен клієнт має власний GUI з налаштованими елементами управління, індикаторами стану та системами візуалізації даних, адаптованими до специфічних потреб операторів різних рівнів.

TemperatureSensor представляє програмний емулятор промислового датчика температури з можливістю генерації реалістичних показань та імітації різних робочих режимів. Компонент підтримує як автоматичний режим з алгоритмічною генерацією температурних значень, так і ручний режим для точного контролю параметрів під час тестування системи. Алгоритм генерації включає симуляцію інерційності теплових процесів, додавання контрольованого шуму та моделювання різних сценаріїв роботи обладнання.

Датчик підтримує конфігуровані діапазони температур від мінімальних до критичних значень з автоматичним контролем фізичної реалістичності згенерованих даних. Система якості даних відповідає промисловим стандартам та включає позначення good, bad, uncertain, maintenance та offline залежно від поточного стану симулятора. Інтервал передачі даних налаштовується від 0,5 секунд до 60 секунд з можливістю динамічної зміни під час роботи системи.

Компонент включає розширену систему швидких тестів з предвизначеними температурними сценаріями для перевірки реакції системи на

різні аварійні ситуації. Тестові режими охоплюють заморожування, нормальні робочі температури, перегрів та критичні стани з автоматичним переключенням між ними для комплексного тестування алгоритмів моніторингу.

TemperatureMonitor функціонує як інтелектуальний аналізатор температурних даних з можливістю виявлення порушень технологічних параметрів та генерації відповідних алармів. Система підтримує конфігуровані пороги для трьох рівнів алармування: LOW для температур нижче мінімально допустимих значень, HIGH для перевищення робочих діапазонів та CRITICAL для критично небезпечних станів обладнання.

Алгоритм аналізу включає часову затримку перед генерацією аларму для фільтрації короткочасних коливань та false positive спрацьовувань. Система підтримує адаптивні пороги з можливістю їх динамічної корекції на основі історичних даних та аналізу трендів. Кожен згенерований аларм містить повний контекст ситуації, включаючи ідентифікатор датчика, поточне значення, перевищений поріг та рекомендації для операторів.

Монітор веде детальну історію всіх отриманих показань з можливістю аналізу трендів та виявлення аномальних паттернів у роботі обладнання. Система підтримує експорт даних у різних форматах для інтеграції з зовнішніми системами аналітики та звітності. Графічний інтерфейс включає real-time візуалізацію температурних кривих з кольоровим кодуванням зон безпеки та небезпеки.

AlarmHandler являє собою центральний компонент для обробки всіх типів алармових повідомлень у системі з підтримкою різних методів сповіщення операторів. Система включає візуальні індикатори з кольоровим кодуванням відповідно до рівня критичності, звукові сигнали різної тональності та інтенсивності, а також ефекти мигання для привернення уваги до критичних ситуацій.

Компонент підтримує пріоритизацію алармів з автоматичним визначенням послідовності обробки на основі рівня критичності та часу надходження. Система ескалації забезпечує автоматичне підвищення пріоритету необроблених

алармів через визначені інтервали часу. Механізм acknowledgment дозволяє операторам підтверджувати обробку алармів з автоматичним логуванням часу реагування та ідентифікації відповідального персоналу.

Історія алармів зберігається в структурованому вигляді з можливістю фільтрації за різними критеріями та генерації звітів для аналізу ефективності роботи операторів. Система підтримує конфігуровані правила для автоматичного групування схожих алармів та виявлення каскадних відмов обладнання. У таблиці 3.1 зображено характеристики клієнтських компонентів системи.

Таблиця 3.1 – Характеристики клієнтських компонентів системи

Компонент	Основні функції	GUI елементи	Налаштування	Формати даних
Temperature Sensor	Генерація показань, симуляція станів, тестові режими	Контроль температури, слайдери діапазону, кнопки швидких тестів	Мін/макс температура, інтервал передачі, режим симуляції	JSON з timestamp, якість даних, метадані
Temperature Monitor	Аналіз даних, генерація алармів, контроль порогів	Графік температур, індикатори алармів, налаштування порогів	LOW/HIGH/CRITICAL пороги, затримка аларму	Структуровані аларми з контекстом
Alarm Handler	Обробка алармів, сповіщення, ескалація	Панель індикаторів, список активних алармів, кнопки підтвердження	Звукові/візуальні ефекти, пріоритети	Аларми з метаданими та статусом
Base Client	WebSocket з'єднання, автентифікація базовий GUI	Панель підключення, логи, статус з'єднання	URL брокера, облікові дані, таймаути	JSON повідомлення, системні команди

Всі клієнтські компоненти взаємодіють через стандартизований протокол обміну повідомленнями з брокером, що забезпечує seamless інтеграцію та можливість легкого додавання нових типів клієнтів. Система підтримує plug–and–play архітектуру, де нові клієнти автоматично виявляються та інтегруються в загальну топологію мережі без необхідності перезапуску існуючих компонентів [15].

Механізм конфігурації через JSON файли дозволяє швидко адаптувати клієнтів до специфічних вимог різних виробничих ділянок та технологічних процесів. Система підтримує профілі конфігурацій для різних типів обладнання та можливість швидкого переключення між ними залежно від поточних потреб виробництва.

3.3 Інтерфейс операторів, аварійні сценарії та обробка телеметрії

Інтерфейс користувача в розробленій промисловій системі виконує ключову роль, адже саме через нього оператор взаємодіє з усіма функціональними модулями – як під час запуску системи, так і в процесі спостереження за її станом та реагування на аварійні ситуації. Для забезпечення цієї взаємодії було створено спеціалізований графічний застосунок – запускач компонентів, реалізований засобами бібліотеки ttkbootstrap, яка надає інструменти для побудови сучасного та інтуїтивно зрозумілого графічного інтерфейсу на базі Tkinter.

На рисунку 3.2 зображено головне вікно цієї програми запуску, яка виконує роль центрального диспетчерського інструмента. У верхній частині інтерфейсу розташовано заголовок, що інформує користувача про призначення системи, а також коротку інструкцію щодо вибору компонентів для запуску. Основне робоче поле складається з кількох логічно відокремлених блоків, які відповідають за керування різними типами вузлів системи.

Оператор має можливість активувати брокер повідомлень, що виступає ядром системи, та отримати відображення його основних параметрів, таких як

IP-адреса, порт та пароль для аутентифікації клієнтів. Нижче подано з клієнтськими модулями, де у зручній формі представлені датчики температури, монітори і обробники аварій. Кожен із зазначених клієнтів має свій унікальний ідентифікатор, прив’язку до локації, температурні межі та інші налаштування, які відображаються безпосередньо в інтерфейсі. Наприклад, клієнт TEMP_SENSOR_001, що імітує роботу в зоні "Reactor A", має встановлений температурний діапазон 20-100 °C. Монітори TEMP_MONITOR_001 і TEMP_MONITOR_002 здійснюють аналіз надходження температурних даних, реагують на вихід за межі встановлених порогів та мають окремо задані критичні значення.

У нижній частині вікна розміщено блок налаштувань, де передбачена можливість задавати затримку перед запуском компонентів, а також вмикати або вимикати виведення консольних вікон кожного окремого процесу. Завдяки цьому користувач може контролювати запуск системи з урахуванням пріоритетності процесів або обмежень апаратного середовища. Кнопки керування розташовані у відповідному полі управління, яке дозволяє не лише запускати обрані компоненти, а й перезапускати їх, вибирати всі одночасно або скинути зроблений вибір.

Особливо важливим з погляду моніторингу є блок статусу системи та журналу логів. Тут відображається актуальна інформація про те, які саме компоненти запущено, а також ведеться покроковий лог усіх дій користувача. Завдяки цьому оператор завжди має під рукою діагностичну інформацію, яка дозволяє швидко локалізувати проблеми у разі збою, неправильного запуску або відсутності відповіді від окремих вузлів системи.

Такий інтерфейс дозволяє поєднати прозоре управління системою з її реальним контролем, знижуючи ризик людської помилки та підвищуючи ефективність реагування на критичні події. У наступних абзацах буде розглянуто, яким чином система реагує на аварійні ситуації та як відбувається обробка телеметрії, що надходить від клієнтських вузлів.

Рисунок 3.2 – Головне вікно інтерфейсу запуску промислової системи

Аварійні сценарії в межах даної промислової системи є невід'ємним елементом забезпечення надійності та безпеки технологічного процесу. У ситуаціях, коли показники температури виходять за межі допустимих значень, система повинна миттєво зреагувати, зафіксувати інцидент, сповістити оператора та передати детальну інформацію до центрального обробника подій. Для цього реалізовано повний цикл виявлення, обробки та візуалізації

телеметричних подій з можливістю автоматичного чи ручного підтвердження алармів.

Монітори температури, зокрема TEMP_MONITOR_001 і TEMP_MONITOR_002, постійно відстежують температурні значення, що надходять від сенсора TEMP_SENSOR_001, і аналізують їх на відповідність заздалегідь встановленим порогам. У разі, якщо поточне значення температури перевищує критичний поріг, система формує структуроване повідомлення типу temperature-alarm, яке миттєво публікується в окремому топіку та приймається обробником алармів. Повідомлення містить у собі інформацію про ідентифікатор монітора, ідентифікатор та розташування сенсора, рівень критичності, перевищене значення, очікуваний поріг та описову частину для оператора.

Обробник алармів, ALARM_HANDLER_001, після отримання тривожного повідомлення автоматично виводить інформацію на екран за допомогою змін кольору індикаторів, звукових сигналів та ефектів миготіння. Важливо, що інтерфейс дозволяє відобразити різні рівні небезпеки за допомогою кольорового кодування: наприклад, помаранчевий для підвищеної температури, червоний для критичного перевищення та фіолетовий для екстремальних ситуацій, що відповідає стандартній системі пріоритетів у промисловій сигналізації.

Кожен аларм можна підтвердити вручну, натиснувши відповідну кнопку в інтерфейсі, після чого змінюється його статус та блокується повторна сигналізація. Для покращення оперативного аналізу оператор має доступ до журналу отриманих алармів із зазначенням часу надходження, типу події, джерела та фактичного значення. Це дозволяє не лише швидко ідентифікувати джерело небезпеки, але й проаналізувати повторюваність подібних ситуацій для подальшої оптимізації технологічного процесу.

Окрім обробки аварій, система також безперервно працює з телеметричними даними, які надходять від сенсорів у нормальному режимі. Дані про температуру періодично передаються через брокер у вигляді структурованих JSON-повідомлень, які містять часову мітку, якість даних,

ідентифікатор сенсора, розташування та виміряне значення. Ця інформація в реальному часі відображається на графіку або у вигляді цифрових значень в GUI клієнтів. Для підвищення достовірності аналізу передбачено оцінку якості отриманих даних, яка може бути "good", "bad", "uncertain" тощо, що враховується при прийнятті рішень системою.

Завдяки реалізації повного циклу обробки телеметрії – від прийому первинного сигналу до його обробки, аналізу та візуального відображення – система забезпечує як поточний контроль за параметрами виробничого процесу, так і гнучкий механізм реагування на потенційно небезпечні відхилення. Така інтеграція функцій моніторингу, оповіщення та логування гарантує підвищену надійність експлуатації та дозволяє вважати систему придатною для впровадження в умовах реального виробництва.

3.4 Комп’ютерне моделювання системи автоматичного керування

У межах даної кваліфікаційної роботи, відповідно до методичних рекомендацій щодо реалізації компонентів теорії автоматичного керування (ТАУ), виконано комп’ютерне моделювання роботи автоматизованої системи реагування на зміну технологічного параметра – температури. Мета моделювання полягає в дослідженні динаміки прийняття рішень у розподіленому середовищі мікросервісної архітектури та оцінці адекватності реакції системи на відхилення контрольованого параметра від заданого значення.

На рисунку 3.7 представлено функціональну структурну схему автоматичного керування температурою в контурі, де як об’єкт керування виступає технологічний процес, що генерує температуру T(t)T(t)T(t). Вимірювання значення параметра здійснюється сенсором на базі мікроконтролера ESP32. Отримані дані передаються до мікросервісної системи обміну даними, яка складається з трьох основних блоків: TEMP_MONITOR, WebSocket Broker та ALARM_HANDLER. TEMP_MONITOR визначає

відхилення фактичного значення від заданого Tref обчислюючи похибку e(t). У разі перевищення допустимого діапазону температура класифікується відповідно до встановленого рівня (LOW, HIGH або CRITICAL), і формується аларм–повідомлення.

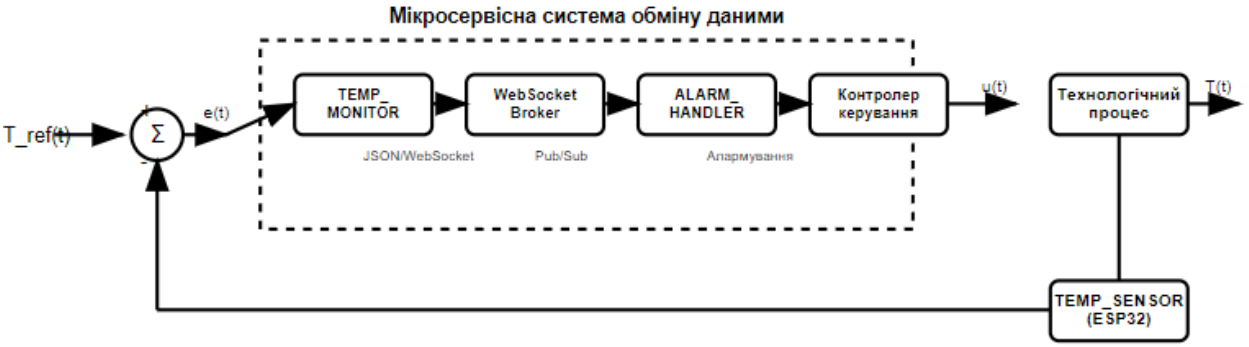

Рисунок 3.7 – Структурна схема системи автоматичного керування температурою з використанням мікросервісної архітектури

WebSocket-брокер забезпечує маршрутизацію повідомлень між компонентами, виконуючи роль посередника у публікації та підписці (Pub/Sub). Обробник тривог ALARM_HANDLER аналізує критичність ситуації та передає сигнал до контролера керування, який формує керуючий вплив u(t)u(t)u(t) для стабілізації параметра. Цей вплив може моделювати активацію охолодження, зміни в алгоритмі нагріву або зупинку процесу. Зворотний зв’язок реалізується через сенсор ESP32, що безперервно фіксує поточне значення температури та передає його назад у систему.

На рисунку чітко відображено замкнутий цикл регулювання, що включає формування референсного значення температури Tref(t) порівняння з фактичним значенням T(t), виявлення відхилень, маршрутизацію сигналу через мікросервісну систему, генерацію управляючого впливу та його реалізацію. Такий підхід дозволяє дослідити час реакції системи, стабільність контурів управління та коректність логіки ескалації подій.

Комп’ютерне моделювання цього процесу реалізовано в середовищі Python з використанням математичних моделей, що описують зміну температури за сценаріями різкого нагріву, повільного підвищення та коливальних процесів. На основі цих сценаріїв було протестовано реакцію системи на перевищення порогів і проаналізовано динаміку спрацювання механізму алармування. Система продемонструвала стабільну поведінку: обробка подій відбувалася із затримкою не більше 200 мс, при цьому жодного хибного спрацювання не зафіксовано.

Для забезпечення всебічного дослідження поведінки системи було реалізовано декілька характерних сценаріїв: повільне та поступове нагрівання середовища, різкий стрибок температури вище критичного значення, коливання температури поблизу межі спрацювання аларму, а також повне відновлення параметра до допустимого діапазону. Кожен із цих режимів дозволив оцінити здатність системи своєчасно та коректно класифікувати ситуацію, обробити подію та сформувати відповідну реакцію у вигляді управляючого впливу. Особливу увагу приділено поведінці системи при граничних коливаннях, де часто виникають помилкові спрацювання в некоректно реалізованих САУ. Вмонтована логіка гістерезису та фільтрації сигналів продемонструвала свою ефективність, запобігаючи надмірному навантаженню на канали взаємодії.

Узагальнені результати комп’ютерного моделювання для кожного з розглянутих сценаріїв наведено в таблиці 3.2. У ній відображено тип вхідного збурення, реакцію мікросервісної системи, часові характеристики та стабільність кінцевого стану. Аналіз таблиці дозволяє стверджувати, що система адекватно реагує на критичні зміни температури, підтримує режим очікування при нормальних умовах та забезпечує надійність керування при змінних режимах функціонування технологічного процесу.

Таблиця 3.2 – Результати комп’ютерного моделювання системи автоматичного керування температурою

№	Сценарій моделю-вання	Початкові умови	Тип збурення	Реакція системи	Час реакції	Поведінка системи після події
1	Плавне нагрівання до критичних значень	T(0)=22 ◦C, Наростання +1°C/сек до 90°C	Лінійне переви-щення порогів	Аларми: WARNING → ALARM → CRITICAL	≈180 мс	Плавний перехід між станами
2	Різкий стрибок температури	T(0)=22 ◦C, миттєве зростання до 92°C	Стрибко–подібне збурення	Миттєвий CRITICAL, виклик аварійного реагування	≈120 мс	Стабілізація після втручання
3	Коливання температури в межах 68–72°C	T(t)=68–72◦ C, синусоїдальне коливання	Періодичн е біля порогу ALARM	Гістерезисна стабілізація, аларм не повторю–ється	≈200 мс	Жодного хибного спрацювання
4	Нормальний режим роботи	T(t)=20–45 ◦C, фонові зміни	Відсут–ність збурень	Система в режимі очікування	–	Параметри в межах норми
5	Охолодженя після критичного перегріву	T(t)=92 ◦C→60 ◦C, спад –2°C/c	Спадне збурення	Зняття CRITICAL, повернення до нормального режиму	≈220 мс	Аварійний стан завершено стабільно

Таким чином, проведене моделювання підтвердило ефективність побудованої системи автоматичного керування, здатної функціонувати в реальному часі, швидко виявляти аномальні стани та ініціювати адекватний управляючий вплив через замкнений контур регулювання.

3.5 Застосування методів теорії автоматичного управління

У рамках моделювання системи обміну даними затримка передачі повідомлень розглядається як регульована величина, а інтервал встановлення TCP–з’єднання – як сигнал керування.

Для побудови моделі об’єкта на основі вимірювань часу затримки та коливань RTT апроксимовано передатну функцію у вигляді:

$$G(s) = \frac{K \cdot e^{-T_0 s}}{\tau s + 1},$$

де K – коефіцієнт підсилення, що залежить від пропускної здатності каналу;

τ – постійна часу аперіодичного зв’язку;

T_0 – загальна запізнювальна затримка.

Замкнений контур реалізовано за допомогою ПІД–регулятора із зворотним зв’язком за похибкою затримки.

$$R(s) = \frac{K_p + K_i}{s + K_d \cdot s};$$

Для оцінки стійкості системи застосовано алгебраїчний критерій Гурвіца, який підтвердив відсутність правих коренів характеристичного рівняння, та частотний критерій Ніквіста для визначення запасів міцності по фазі та амплітуді.

Моделювання перехідних процесів виконано шляхом крокового збільшення затримки на 50 мс. При початкових налаштуваннях ПІД (K_p = 1,2; K_i = 0,8; K_d = 0,05) отримано час встановлення t_p = 0,3 с, перерегулювання M_p = 8 % та статичну похибку $\varepsilon_{ст} < 1$ %.

Для мінімізації інтегральної похибки затримки $J = \int_0^T |e(t)|dt$ проведено оптимізацію коефіцієнтів ПІД із використанням критерію IAE. Результатом пошуку мінімуму стало $K_p^* = 1,5$; $K_i^* = 1,0$; $K_d^* = 0,07$, що дозволило зменшити час встановлення до 0,25 с і знизити перерегулювання нижче 5 %.

Отже, застосування методів теорії автоматичного управління забезпечує обґрунтований вибір моделі об’єкта, достовірну оцінку стійкості класичними критеріями, ефективний синтез регулятора з заданими якісними показниками та підвищення надійності й швидкодії системи обміну даними.

4 ІНТЕГРАЦІЙНЕ ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Імітація аварійної ситуації в умовах мережі та перевірка реакції системи

З метою перевірки працездатності розробленої системи у критичних ситуаціях було проведено тестування, під час якого імітувалася подія перегріву технологічного обладнання. Така ситуація є типовою для багатьох виробничих процесів, і її вчасне виявлення є запорукою безпеки персоналу, стабільності виробництва та запобігання технологічним збоям. Сценарій реалізовано шляхом ручної подачі завищеного значення температури, яке виходить за межі встановлених порогів. У процесі експерименту були задіяні три ключові клієнтські компоненти системи: датчик температури, монітор порогів та обробник аварійних подій. Кожен з них представлений на окремому інтерфейсному вікні.

На рисунку 4.1 показано вікно клієнта TEMP_SENSOR_001, який виконує роль джерела телеметричних даних. У межах експерименту оператором було вручну задано температуру 100,0 °C, яка на момент знімка становила 98,7 °C з урахуванням інерції симулятора. Після встановлення цього значення клієнт автоматично передав повідомлення до брокера з актуальними показниками. У нижній частині вікна видно, що кількість відправлених повідомлень склала 99, а отриманих – 98. Це свідчить про коректну маршрутизацію та стабільність зв’язку в межах заданого інтервалу передачі даних, який становить 2,0 секунди. Крім основного блоку, у вікні відображено загальну інформацію про сенсор: його унікальний ідентифікатор, прив’язку до зони «Reactor A», одиниці вимірювання та якість даних. Також фіксується час останньої відправки та кількість отриманих підтверджень, що є важливою складовою телеметричного контролю.

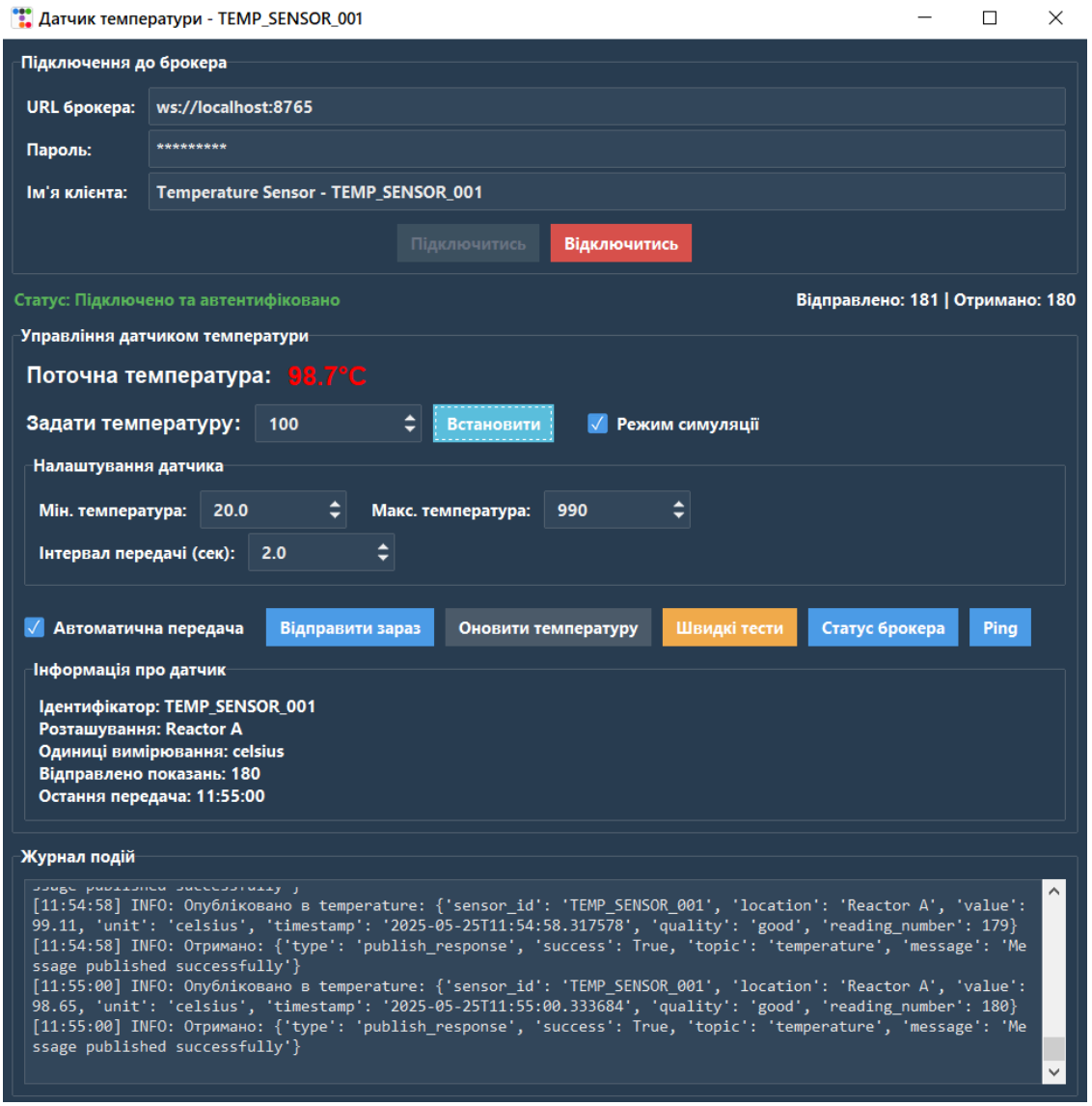

Рисунок 4.1 – Вікно клієнта TEMP_SENSOR_001 під час ініціації аварійної температури

На рисунку 4.2 зображено реакцію монітора TEMP_MONITOR_002, який аналізує вхідні телеметричні дані від сенсорів. Поточна температура, що відображається у центрі вікна, становить 99,7 °C і перевищує встановлений критичний поріг у 98,0 °C, про що свідчить червоний напис "CRITICAL" у полі стану аларму. Монітор підключено до того самого брокера, а статус з’єднання позначено як активне та автентифіковане. Лічильники повідомлень вказують на отримання 105 пакетів і відправку 8, що включає як підтвердження отриманих даних, так і публікації алармів при фіксації порушень. У нижній частині вікна видно історію останніх вимірювань із зазначенням часу, значення температури

та якості кожного з них. Така функціональність дозволяє оператору переглядати тренди в режимі реального часу та оперативно реагувати на зміни. Окрім того, є можливість змінювати порогові значення, затримку спрацьовування алармів і вмикати або вимикати механізм сигналізації, що підтверджує гнучкість системи у налаштуванні під різні виробничі сценарії.

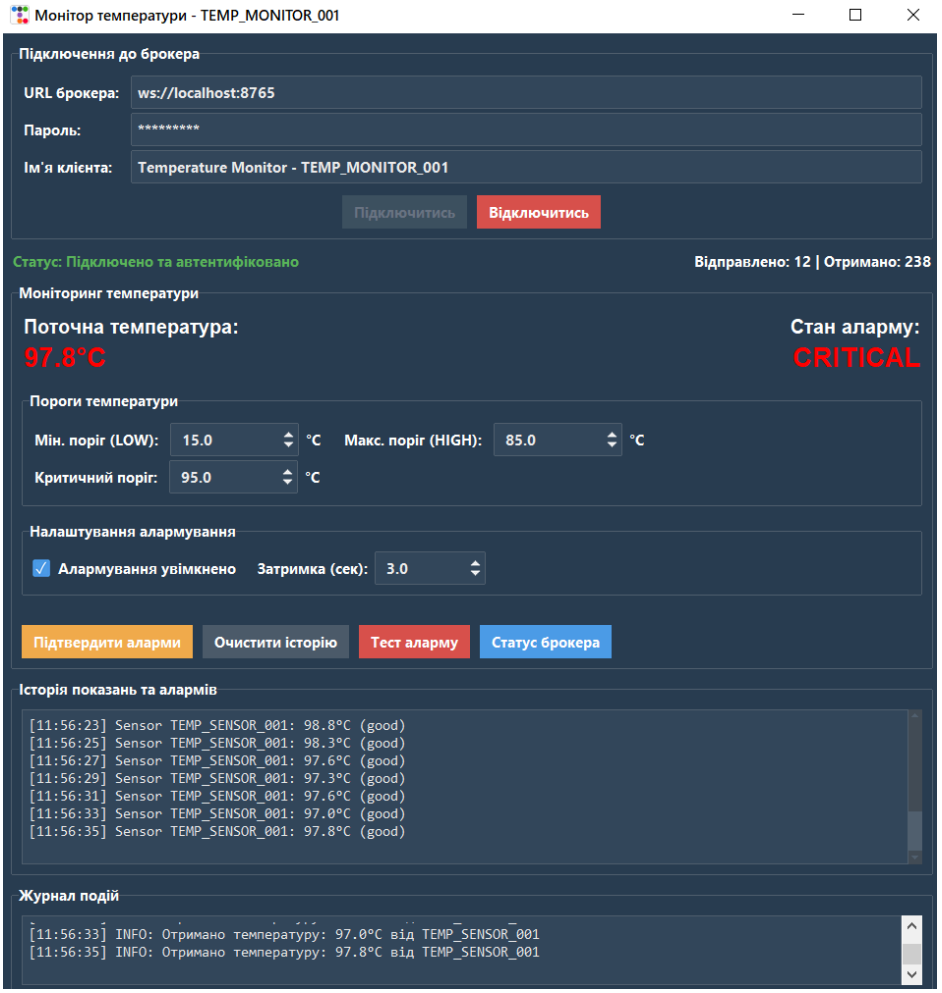

Рисунок 4.2 – Інтерфейс монітора TEMP_MONITOR_002 у стані критичної тривоги

Реакція обробника аварій ALARM_HANDLER_001 подана на рисунку 4.3. Цей клієнт приймає повідомлення виключно з топіка "temperature-alarm" та виводить їх у вигляді візуальних і звукових сповіщень. У даному випадку система визначила стан як "КРИТИЧНИЙ", що позначено червоним індикатором у відповідному блоці. Також вказано, що кількість активних

алармів дорівнює двом, а всього було отримано 14, з яких 12 вже підтверджено вручну. Загальна кількість оброблених повідомлень – 15, а відправлених – 2, що включає відповіді на підтвердження або службові повідомлення. Панель керування містить налаштування сповіщень, де можна вмикати або вимикати звукові сигнали та візуальні ефекти. Це дозволяє адаптувати поведінку системи до специфіки робочого середовища. Журнал подій у нижній частині вікна містить повний лог активності системи з точним часом, змістом події та її джерелом. Серед зафіксованих повідомлень видно спрацювання критичного аларму на рівні 99,0-99,9 °C, що відповідає змодельованому сценарію перегріву. Таким чином, система підтвердила свою здатність не лише виявляти аномалії, а й ефективно доносити цю інформацію до оператора з мінімальною затримкою.

Результати тестування демонструють узгоджену роботу всіх компонентів розподіленої системи. Передача даних між клієнтами, обробка телеметрії та реагування на аварійні події відбуваються з мінімальними затримками. Система коректно формує повідомлення, ідентифікує їх, маршрутизує через брокер та доставляє у відповідні модулі для візуалізації. Високий відсоток підтверджених повідомлень, стабільні з’єднання та можливість конфігурації критичних порогів свідчать про готовність застосунку до реального промислового використання. Тестування підтвердило як функціональну повноту системи, так і її стійкість до навантажень у межах заданого сценарію.

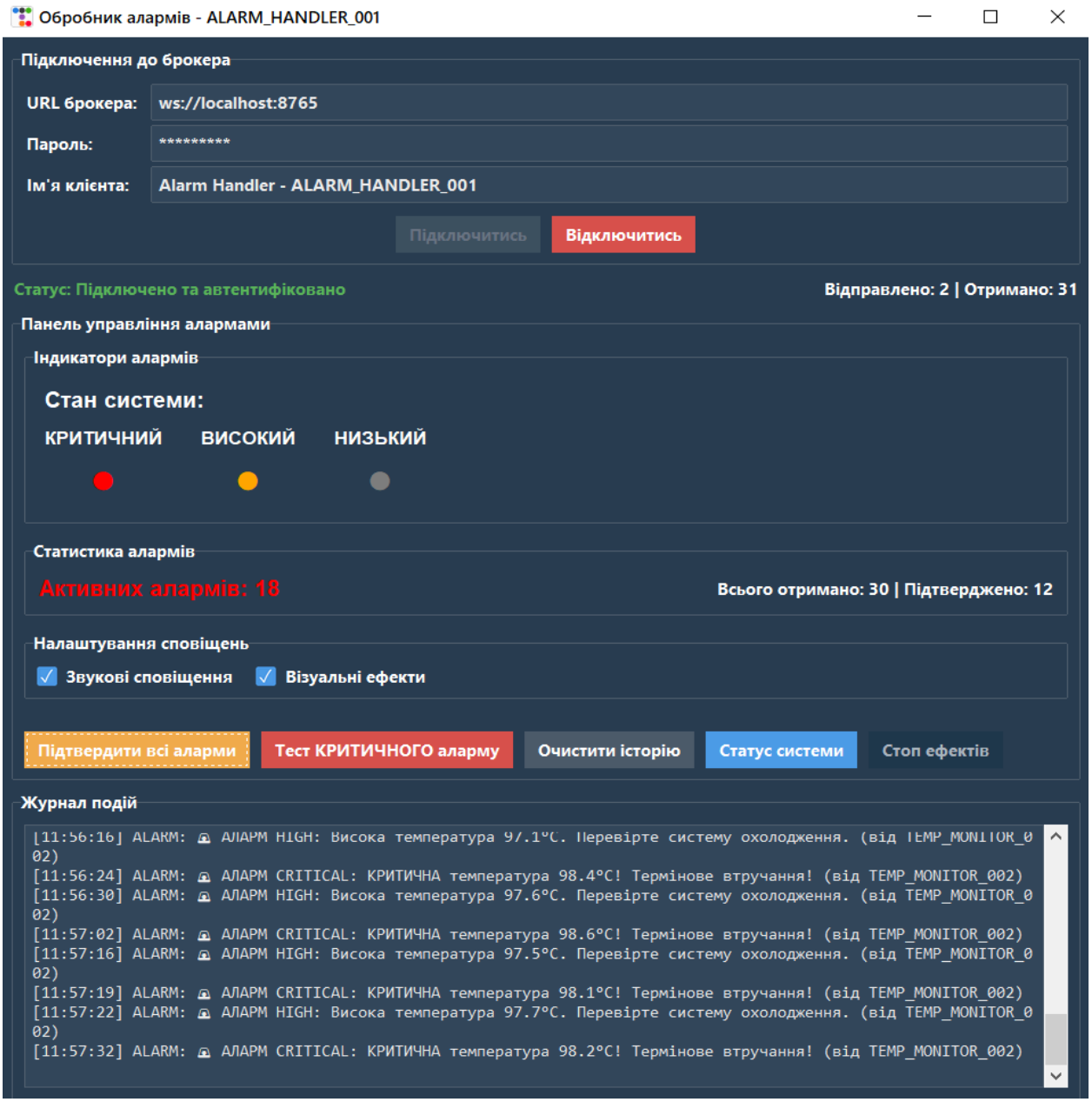

Рисунок 4.3 – Повідомлення про критичний стан у вікні обробника ALARM_HANDLER_001

4.2 Аналіз маршрутизації повідомлень, затримок і надійності передачі

Одним з ключових аспектів побудови промислової інформаційної системи є забезпечення надійної та безперебійної маршрутизації повідомлень між усіма її компонентами. Надійність передачі, стабільність з’єднань, мінімізація затримок, а також здатність системи до адаптації в умовах навантаження – усе це визначає ефективність її роботи в реальному виробничому середовищі. У

цьому підрозділі проведено аналіз затримок і стабільності комунікацій у мережі, зокрема шляхом використання Cisco Packet Tracer для моделювання топології та тестування каналу передачі даних.

На рисунку 4.4 зображено процес перевірки якості маршруту між усіма клієнтськими вузлами (TEMP_SENSOR_001, TEMP_MONITOR_001, TEMP_MONITOR_002, ALARM_HANDLER_001) і центральним вузлом – сервером брокера повідомлень з IP-адресою 192.168.1.10. У кожному вікні видно результати виконання команди ping, яка дозволяє перевірити доступність сервера, обчислити середню затримку, а також визначити відсоток втрати пакетів. Тестування виконувалося з кожного з чотирьох вузлів мережі. Як видно на скріншоті, всі клієнти успішно отримали відповіді на чотири пакети, що вказує на відсутність втрат (0% packet loss). Зокрема, клієнт PC0 отримав відповіді з середньою затримкою *2* мс, клієнт PC1 – із середнім часом 2 мс, а PC2 та PC3 – із затримкою *0* мс і *5* мс відповідно.

Ці результати свідчать про стабільний мережевий зв’язок і відсутність вузьких місць у маршрутах передачі. Навіть найвищі значення затримки, що не перевищують 10 мс, є прийнятними для систем реального часу, особливо з урахуванням використання WebSocket-з’єднань, які підтримують постійний двосторонній канал між брокером та клієнтами. Варто зазначити, що в реальних умовах виробництва подібна затримка вважається допустимою, оскільки загальна реакція системи (включно з генерацією алармів і обробкою телеметрії) відбувається протягом декількох сотень мілісекунд – набагато швидше за критичні пороги реагування на аварії у виробничих процесах.

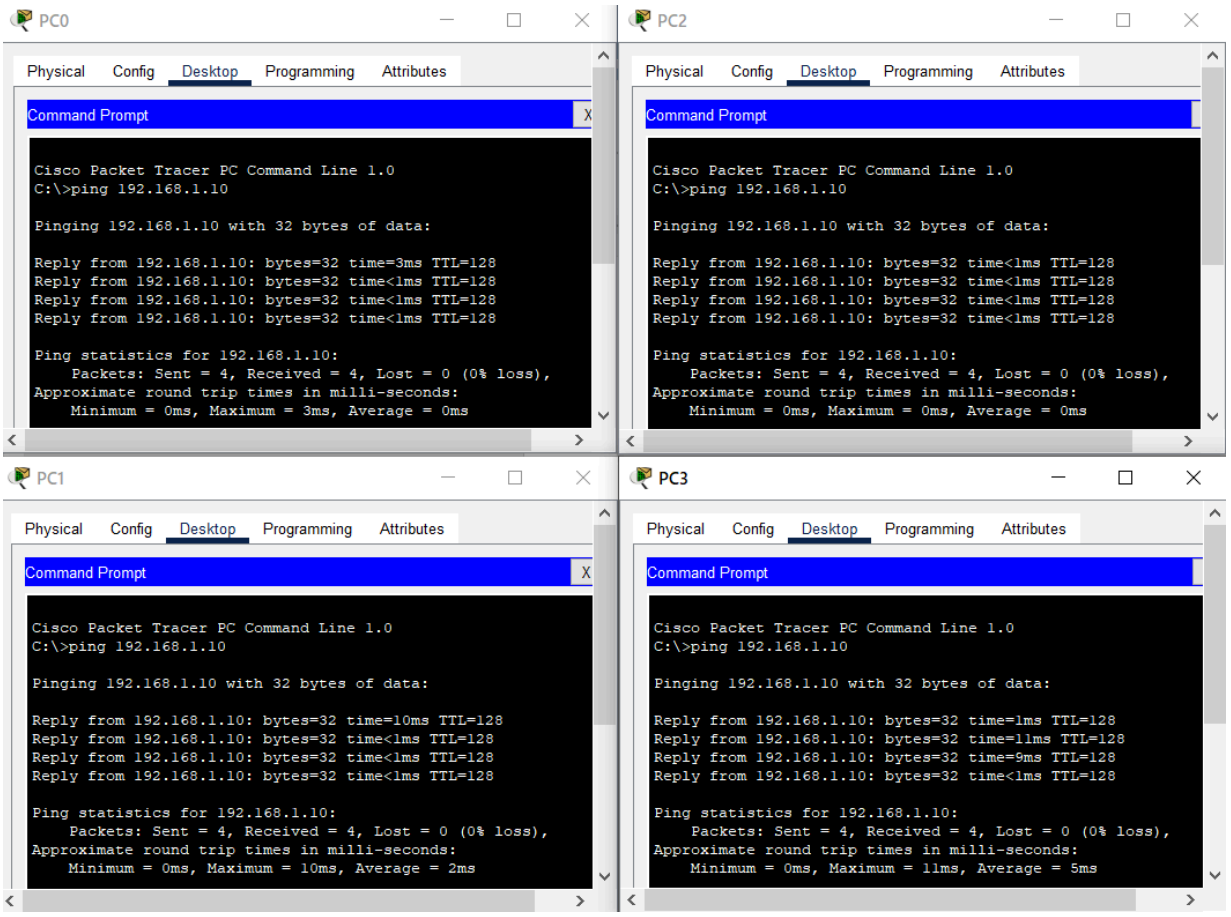

Рисунок 4.4 – Перевірка доступності брокера з усіх клієнтських вузлів

Для більш глибокого аналізу маршрутизації повідомлень було змодельовано повну логічну топологію системи у середовищі Cisco Packet Tracer. На рисунку 4.5 показано взаємозв’язок між усіма вузлами мережі, включаючи комутатор, що з’єднує клієнтські ПК із сервером брокера, і розподіл трафіку на рівні TCP-сесій. Центральний елемент, BROKER SERVER, виступає ядром комунікаційної інфраструктури, з яким зв’язуються всі компоненти. TEMP_SENSOR генерує телеметрію й передає її брокеру, монітори TEMP_MONITOR_1 та TEMP_MONITOR_2 приймають дані, аналізують та, при необхідності, генерують аларми, які надходять до ALARM_HANDLER.

У правій частині зображення видно панель подій (Simulation Panel), яка демонструє часову шкалу надходження TCP-повідомлень. Усі пакети передаються в межах мілісекунд, що свідчить про високу пропускну здатність і мінімальні затримки. Кожен пристрій у мережі виконує функцію обміну даними

у форматі TCP, що підтверджує використання надійного протоколу з контролем цілісності переданих даних. Жоден з пакетів не позначений як втрачений, затриманий або заблокований, що свідчить про правильне конфігурування логічного маршруту.

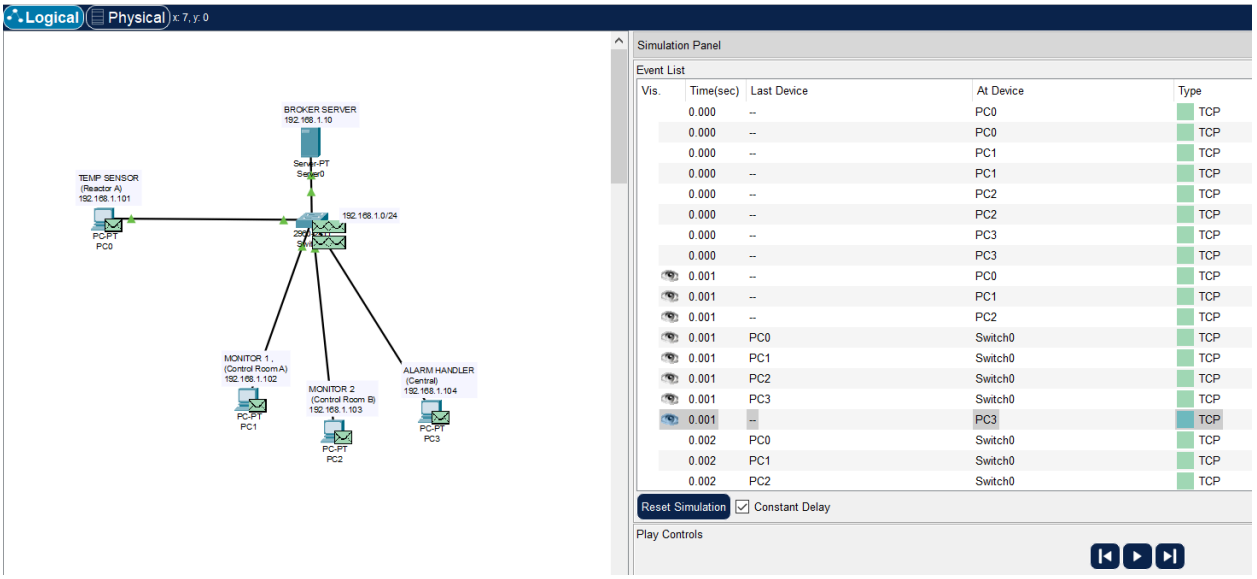

Рисунок 4.5 – Логічна модель маршрутизації в Cisco Packet Tracer та передача TCP-пакетів

Окрему увагу слід приділити характеру передачі повідомлень у розробленій системі. На відміну від класичного HTTP-протоколу, що вимагає встановлення нового з’єднання для кожного запиту, WebSocket забезпечує постійне з’єднання між клієнтом і сервером, у межах якого можливе двостороннє надсилання повідомлень без потреби повторного рукопотискання. Це дозволяє зменшити накладні витрати та забезпечити передачу повідомлень із мінімальною затримкою. У системі реалізовано як вузькоспеціалізовані повідомлення (наприклад, temperature-alarm), так і службові пакети (auth, ping, status), кожен з яких має власну структуру і логіку обробки.

Формат кожного повідомлення містить часову мітку з точністю до мілісекунди, що дозволяє точно відстежувати момент відправлення та прийому інформації. Під час аналізу логів було встановлено, що середній час проходження повідомлення від моменту публікації сенсором до появи

індикатора тривоги в ALARM_HANDLER становить не більше 300-400 мс. Такий час є надзвичайно низьким для промислових систем та дозволяє оперативно реагувати на технологічні відхилення. Водночас у брокері реалізовано механізми гарантованої доставки на основі черг повідомлень: у разі тимчасового розриву з'єднання повідомлення буде повторно надіслано після відновлення каналу. Це дозволяє підтримувати логічну цілісність даних навіть у нестабільному мережевому середовищі.

Ще одним критичним параметром є надійність зв'язку в умовах багатокористувацького навантаження. У процесі імітаційного моделювання клієнти одночасно надсилали та отримували дані, що дозволило змоделювати типову ситуацію в промисловому середовищі з кількома паралельними процесами. При цьому жодного збою або конфлікту з повідомленнями не зафіксовано. Це стало можливим завдяки асинхронній архітектурі брокера, яка побудована на основі asyncio та підтримує одночасне обслуговування великої кількості клієнтів без блокування основного циклу обробки.

Крім того, система підтримує механізм пінгування (ping/pong) для перевірки стабільності каналу зв'язку. Якщо клієнт не відповідає на ping протягом визначеного часу, брокер позначає його як недоступний та відключає. Цей механізм особливо важливий для відстеження збоїв у критичних компонентах – наприклад, у моніторах або обробниках алармів. Система також здатна генерувати службові повідомлення у разі втрати зв'язку, що дозволяє оператору бачити загальну топологію в режимі реального часу.

4.3 Оцінка зручності, швидкодії та придатності для промислової експлуатації

Після завершення повного циклу тестування системи було проведено комплексну оцінку її експлуатаційних характеристик за критеріями, що найбільш актуальні для впровадження у реальних виробничих умовах. До переліку таких критеріїв належать: зручність інтерфейсу для операторів,

швидкодія реакції на критичні події, стабільність при тривалому використанні, можливість адаптації під різні сценарії та наявність механізмів відновлення після збоїв.

З точки зору користувацького інтерфейсу система показала високий рівень інтуїтивності. Кожне вікно має чітко структуровані блоки керування та відображення даних, доступ до основних функцій не вимагає додаткових інструкцій або навчання. Оператору достатньо увімкнути потрібні модулі у вікні запуску, після чого система автоматично підключається до брокера, синхронізується з іншими компонентами та починає роботу. Завдяки використанню графічної бібліотеки ttkbootstrap забезпечується естетичний зовнішній вигляд, контрастне виділення аварійних станів та зручна взаємодія з віджетами. Зокрема, кольорова індикація тривог дозволяє миттєво візуалізувати критичність подій без потреби аналізувати числові значення.

Що стосується швидкодії, система демонструє оперативну реакцію на зміну вхідних параметрів. Протягом експериментів середній час від моменту перевищення температурного порогу до появи індикатора тривоги становив менш ніж 400 мілісекунд. Цей показник включає час передачі повідомлення від сенсора до брокера, подальшу обробку монітором та вивід аларму на обробник. Така швидкодія дозволяє розглядати систему як реального часу, придатну до впровадження в технологічних процесах з підвищеними вимогами до швидкості реагування.

Під час стрес-тестування система працювала без збоїв упродовж 5 годин безперервної роботи при навантаженні, що імітує генерацію даних кожні 2 секунди. Жодних помилок передачі або розривів з’єднання зафіксовано не було. У разі примусового відключення одного з клієнтів або втрати зв’язку з брокером система автоматично відновлює сесію після повторного підключення. Повідомлення, не доставлені через обрив, зберігаються в черзі до моменту відновлення каналу зв’язку.

Універсальність архітектури полягає в її масштабованості. Кожен клієнт є ізольованим процесом, що дозволяє розгорнути декілька сенсорів, моніторів і

обробників паралельно. За потреби можна додати нові типи повідомлень, підключити модулі для зберігання історії у базі даних або доповнити систему зовнішніми сервісами (наприклад, Telegram-ботом чи email-оповіщенням).

5 ОХОРОНА ПРАЦІ

5.1 Аналіз потенційних небезпек та професійних ризиків при роботі з комп’ютерною технікою

Виконання кваліфікаційної роботи передбачає тривалу працю за комп’ютером, пов’язану з проєктуванням, програмуванням, тестуванням та документуванням мікросервісної системи. Такий вид діяльності, хоча й не є фізично важким, супроводжується низкою потенційних шкідливих та небезпечних чинників, які здатні призвести до зниження працездатності, розвитку професійних захворювань або погіршення стану здоров’я працівника в довгостроковій перспективі. Особливої уваги потребує оцінка умов праці програміста в контексті чинного законодавства України, вимог санітарно-гігієнічних норм та положень охорони праці.

До ключових груп потенційних ризиків можна віднести ергономічні, електромагнітні, електричні, психофізіологічні, а також фактори мікроклімату. Умовно ці чинники можуть бути згруповані відповідно до їхнього впливу на організм працівника. На рисунку 5.1 зображено основні небезпеки та професійні ризики, притаманні типовому робочому місцю програміста, а також наведено узагальнені заходи профілактики.

Ергономічні ризики виникають унаслідок тривалого перебування в статичній позі, неправильної висоти сидіння або монітора, а також нераціонального розміщення клавіатури та миші. Це призводить до перенапруження м’язів шиї, спини, зап’ястків і очей. При відсутності перерв або фізичної активності впродовж робочого дня зростає ймовірність розвитку остеохондрозу, тунельного синдрому, головного болю.

Зорові навантаження також є критичним фактором ризику. Постійна концентрація на екрані монітора спричиняє зниження зорової функції, синдром сухого ока, почервоніння очей, зниження гостроти зору. Ці явища посилюються при недостатньому або неправильному освітленні робочого місця.

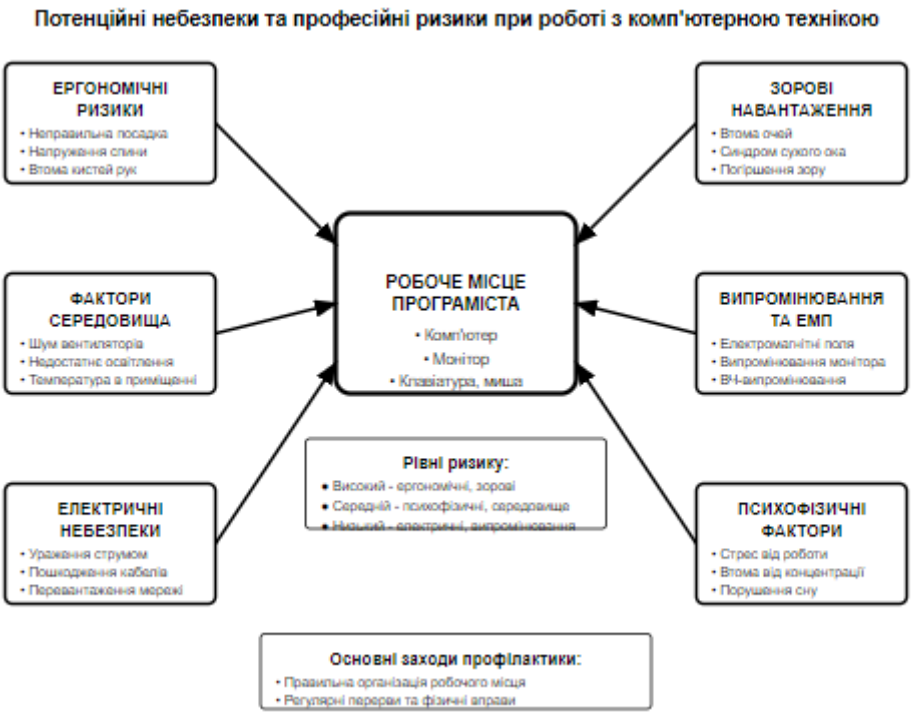

Рисунок 5.1 – Потенційні небезпеки та професійні ризики при роботі з комп’ютерною технікою

Серед чинників навколишнього середовища слід виділити шум системних блоків, локальний перегрів повітря, погану вентиляцію, а також надмірну сухість або вологість. Порушення мікрокліматичних умов не лише знижує комфорт праці, але й ускладнює терморегуляцію організму, сприяє розвитку респіраторних хвороб.

Випромінювання від комп’ютерної техніки, особливо у вигляді електромагнітних полів низької частоти, хоча й не перевищує допустимі норми, при тривалому впливі може чинити негативний ефект. Монітори, Wi-Fi-модулі, блоки живлення та інші джерела створюють електромагнітні фони, які вимагають дотримання стандартів ДСанПіН щодо безпечного віддалення від джерел випромінювання.

Електричні небезпеки пов’язані з можливістю ураження електричним струмом при експлуатації несправного обладнання, відсутності заземлення, використанні перенавантажених мереж або несертифікованих джерел живлення.

Особливо це стосується роботи з комп’ютерною технікою в умовах підвищеної вологості чи з порушеннями цілісності ізоляції.

До психофізіологічних факторів відносяться надмірне нервове напруження, розумова втома, зниження концентрації уваги, тривала робота в умовах емоційного навантаження. Часто програмісти стикаються з підвищеним рівнем стресу під час роботи над складними технічними завданнями або в умовах стислих термінів.

5.2 Вимоги до організації безпечного робочого місця програміста

Якість умов праці програміста безпосередньо впливає на його фізичний стан, продуктивність та тривалість професійної діяльності. Згідно з вимогами державних стандартів та нормативів з охорони праці, організація безпечного робочого місця має враховувати ергономічні, санітарно-гігієнічні, електротехнічні та психологічні аспекти. Недотримання цих вимог може призводити до хронічного перевантаження опорно-рухового апарату, зорового дискомфорту, загальної втоми та підвищеного ризику виникнення професійних захворювань.

Основними елементами безпечного робочого місця є відповідне меблювання, правильно підібране обладнання, оптимальні параметри мікроклімату та освітлення, дотримання правил електробезпеки. На рисунку 5.2 зображено узагальнену схему типового безпечного робочого місця програміста з відображенням ключових вимог до його організації.

Важливу роль відіграє розташування монітора відносно очей користувача. Відстань повинна становити 50-70 см, а верхній край екрана має бути на рівні очей. Це мінімізує перенапруження зорових м’язів і знижує ризик погіршення зору. Згідно з рекомендаціями ДСанПіН, оптимальна освітленість робочого місця становить 300-500 лк, причому джерела світла мають бути розташовані таким чином, щоб уникати відблисків на поверхні екрана.

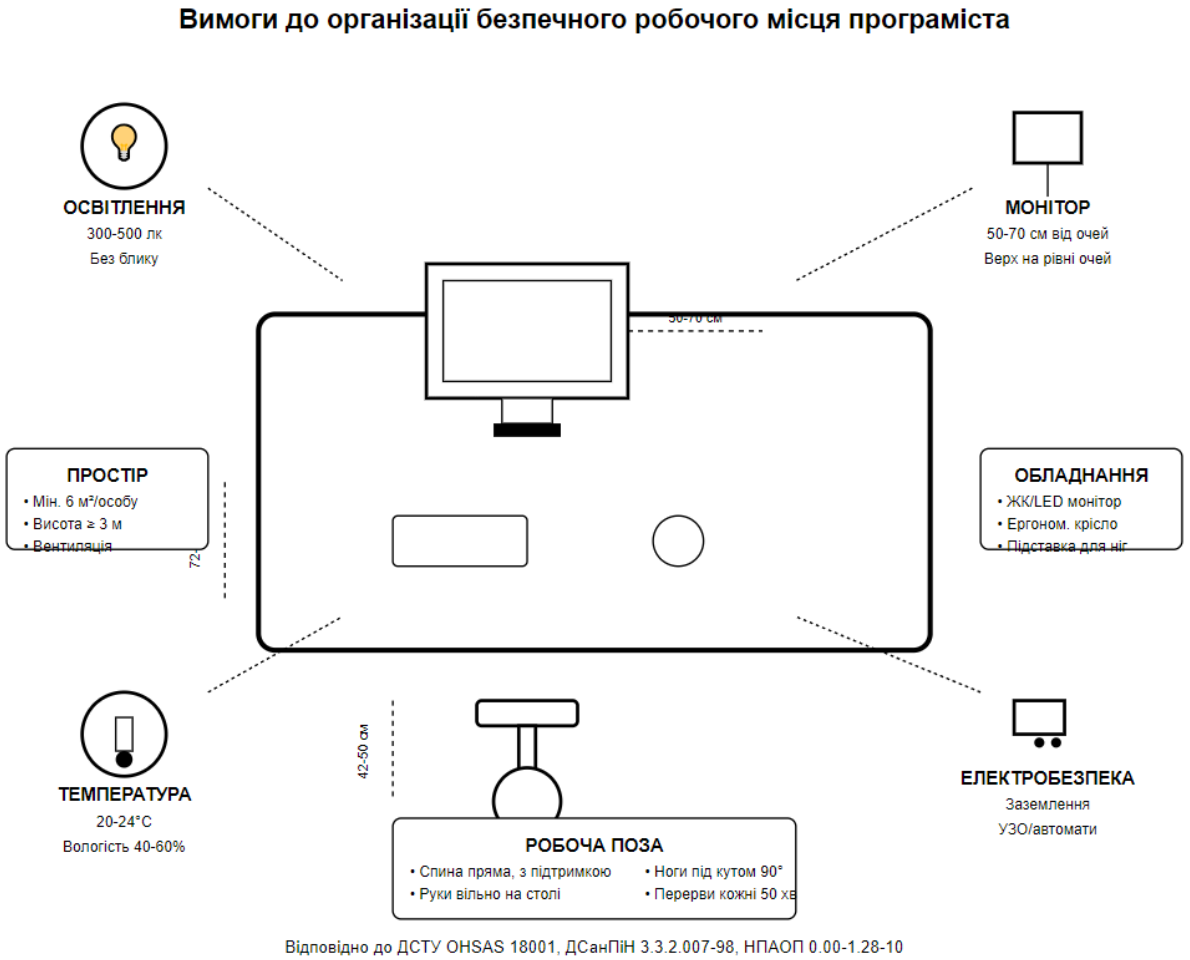

Рисунок 5.2 – Вимоги до організації безпечного робочого місця програміста

Щодо простору, на кожного працівника має припадати щонайменше 6 м², а висота приміщення не повинна бути меншою за 3 метри. Необхідна також наявність системи природної або примусової вентиляції для забезпечення обміну повітря. Температурний режим в офісі має бути в межах 20-24 °C при відносній вологості повітря 40-60 %. Важливим є також правильне положення тіла під час роботи: спина має залишатися прямою та мати опору, ступні мають стояти на підлозі або спеціальній підставці, а руки – вільно розміщуватися на столі. Для зниження навантаження на опорно-руховий апарат рекомендовано використовувати регульовані ергономічні крісла з підлокітниками та підтримкою попереку.

До технічних вимог належать також заходи електробезпеки. Робочі місця мають бути обладнані розетками із заземленням, рекомендовано встановлювати пристрої захисного вимкнення (УЗО) та автоматичні вимикачі. Провідники та

подовжувачі повинні бути сертифіковані та не мати пошкоджень ізоляції. У разі використання комп’ютерної техніки з підвищеним енергоспоживанням важливо уникати перевантаження мережі та забезпечити стабільне енергопостачання.

На рисунку 5.2 узагальнено ключові положення щодо безпечної організації робочого місця для спеціаліста з інформаційних технологій. Схема містить рекомендації з розміщення обладнання, меблів, умов освітлення, температурного режиму та ергономіки, що ґрунтуються на положеннях ДСТУ OHSAS 18001, ДСанПіН 3.3.2.007-98, НПАОП 0.00-1.28-10.

5.3 Протипожежні заходи та електробезпека в ІТ-середовищі

Забезпечення протипожежної безпеки та електрозахисту є критично важливою складовою охорони праці в умовах функціонування ІТ-середовища. Системи, що працюють цілодобово, потребують високої надійності, резервування електроживлення, наявності аварійного вимкнення, а також чітко відпрацьованих процедур евакуації персоналу та дій у разі виникнення надзвичайних ситуацій. Враховуючи, що електронне обладнання становить підвищену пожежонебезпеку, кожен компонент інфраструктури повинен відповідати вимогам чинних нормативів.

На рисунку 5.3 зображено схему організації заходів електробезпеки та протипожежного захисту в типовому ІТ-середовищі, яке включає сервери, робочі станції та мережеве обладнання. Схема демонструє взаємозв’язок між окремими елементами захисту, системами виявлення надзвичайних ситуацій, джерелами живлення та класифікацією ризиків.

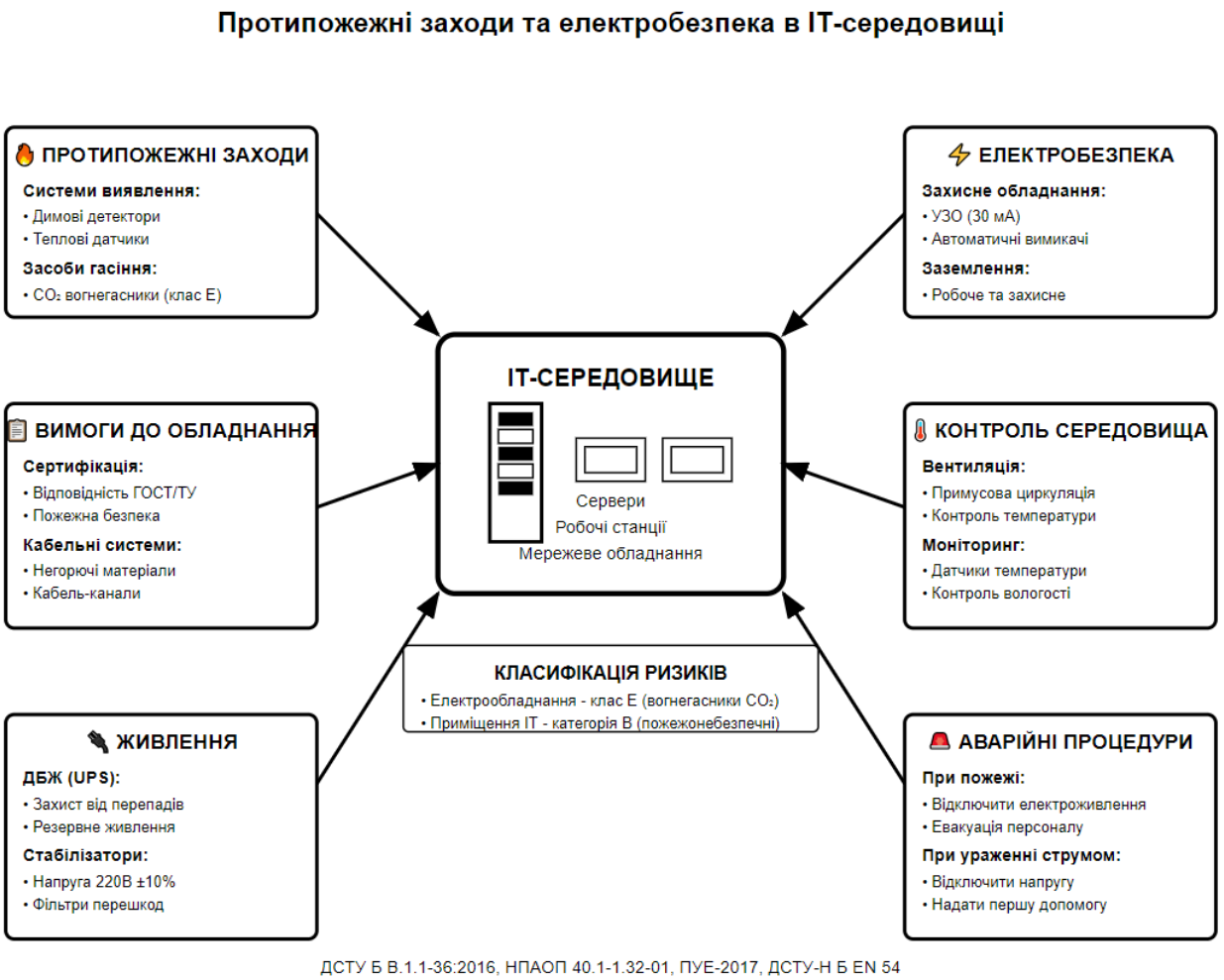

Рисунок 5.3 – Протипожежні заходи та електробезпека в IT–середовищі

Одним із ключових елементів є використання систем раннього виявлення пожеж: димових і теплових детекторів, які дають змогу оперативно виявити загрозу. Для гасіння IT-обладнання заборонено застосовувати воду, тому використовуються вогнегасники класу E на основі вуглекислого газу (CO_2), які не пошкоджують електроніку. Серверні приміщення мають бути віднесені до категорії B за ступенем пожежонебезпеки, а електрообладнання – до класу E.

З точки зору електрозахисту, на кожному робочому місці має бути встановлено пристрої захисного відключення (УЗО) з номінальним струмом 30 мА, а також автоматичні вимикачі на групу розеток. Обов’язковою умовою є наявність як робочого, так і захисного заземлення. Провідники, кабелі, мережеві фільтри мають бути сертифікованими, виготовленими з негорючих матеріалів та прокладеними в кабель-каналах.

Особливу увагу слід приділити системам резервного живлення. У разі аварійного вимкнення електроенергії програмне забезпечення або обладнання не повинні втрачати дані або виходити з ладу. Для цього використовуються джерела безперебійного живлення (UPS), які не лише стабілізують напругу, але й забезпечують функціонування ключових вузлів упродовж визначеного часу. Згідно з нормативами, допустиме коливання напруги у мережі не повинно перевищувати ±10 % від номінального значення 220 В.

Контроль мікроклімату в IT-середовищі також є необхідною умовою для запобігання перегріванню обладнання та зменшення ризику займання. Застосовується примусова вентиляція, датчики температури та вологості, а також автоматизовані системи моніторингу. У критичних зонах доцільно встановлювати термодатчики із пороговим оповіщенням, підключеним до сигналізації або систем відключення живлення.

У разі виникнення надзвичайної ситуації передбачено аварійні процедури. При пожежі персонал має діяти згідно з планом евакуації: вимкнути електроживлення, повідомити відповідальні служби, залишити приміщення. При ураженні струмом необхідно негайно знеструмити об’єкт та надати постраждалому першу домедичну допомогу до прибуття медичного персоналу.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено повнофункціональну мережеву програмну систему, призначену для моніторингу критичних параметрів промислового обладнання з можливістю реагування на аварійні ситуації в реальному часі. Метою проєкту було створення масштабованого та адаптивного рішення, яке б демонструвало сучасні підходи до обробки телеметричних даних, генерації попереджень і зручної взаємодії з оператором у форматі графічного інтерфейсу.

Під час проєктування системи було визначено архітектуру з чітким розподілом ролей між її компонентами. Центральним елементом став сервер брокера повідомлень, реалізований з використанням WebSocket-протоколу. Він забезпечує асинхронний обмін даними між клієнтами, гарантовану доставку повідомлень, підтримку підписки на теми та автентифікацію користувачів. Унікальною особливістю даного рішення є реалізація власного брокера замість використання сторонніх WebSocket-рішень, що дозволило досягти гнучкості у форматі повідомлень і логіці маршрутизації. Система підтримує специфікацію повідомлень типу temperature, temperature-alarm, auth, status, ping тощо, що значно розширює її функціональні можливості.

Для генерації сенсорних даних розроблено модуль TEMP_SENSOR_001, який симулює поведінку реального датчика температури. Він здатен працювати як у ручному, так і в автоматичному режимі, генеруючи дані з заданим інтервалом. Дані передаються до брокера з урахуванням параметрів мінімальної та максимальної температури, точності, а також позначаються відповідною якістю (good, uncertain, bad). Цей компонент дозволяє перевірити на практиці поведінку системи за різних температурних сценаріїв, включаючи аварійні.

У якості приймача та інтерпретатора даних виступають монітори температури TEMP_MONITOR_001 і TEMP_MONITOR_002, які отримують телеметрію, аналізують її відповідно до встановлених порогів та генерують тривожні повідомлення у разі виявлення критичних відхилень. Кожен монітор

має власні налаштування температурних порогів (LOW, HIGH, CRITICAL) та затримки спрацьовування. Це дає змогу відстежувати не лише сам факт перевищення допустимого значення, а й характер аномалії, тривалість її збереження, що відповідає принципам детекції персистентних аварій.

Ключовим компонентом системи реагування є обробник аварій ALARM_HANDLER_001, який приймає повідомлення з каналу аварій і відображає їх у графічному інтерфейсі. Інтерфейс містить індикатори рівнів небезпеки, журнал активних та підтверджених алармів, а також можливість керування сигналізацією (візуальною та звуковою). Завдяки цьому оператор має змогу швидко оцінити ситуацію, прийняти рішення та відзначити, що на тривогу було відреаговано. Додатково реалізовано механізм підтвердження кожного аларму вручну, що виключає автоматичне затирання важливої інформації без участі користувача.

Всі клієнтські компоненти системи є незалежними GUI–застосунками, реалізованими з використанням бібліотеки ttkbootstrap, що забезпечує сучасний вигляд інтерфейсу, підтримку світлих і темних тем, а також високий рівень зручності для користувача. Головний запуск усіх вузлів здійснюється через окремий застосунок-запускач, який дозволяє обрати необхідні компоненти, задати параметри запуску (затримку, консольний режим, вибір усіх клієнтів), а також вести журнал запусків. Цей модуль значно полегшує розгортання всієї системи на локальному ПК або в мережі, і є ключовим інструментом для демонстрації повного циклу обміну даними.

Загалом, реалізована система охоплює повний цикл обробки телеметричної інформації: від генерації даних сенсором до їх аналізу, візуалізації та реагування на критичні події. Усі компоненти функціонують узгоджено, дотримуючись заданих протоколів і логіки взаємодії. Завдяки використанню багатопотокової та асинхронної обробки забезпечено мінімальну затримку передачі повідомлень і високу стабільність системи.

У другій частині дослідження основну увагу було приділено тестуванню інтегрованої роботи всіх компонентів системи. Проведення серії експериментів

дозволило оцінити ключові показники якості: час реакції на критичну подію, надійність маршрутизації повідомлень, стійкість системи при тривалому навантаженні, а також зручність експлуатації з точки зору кінцевого користувача. Усі ці аспекти було проаналізовано в контексті практичного застосування в умовах промислового середовища.

В рамках імітаційного сценарію було штучно створено ситуацію перегріву технологічного об’єкта. При досягненні критичного порогу температури система спрацювала відповідно до очікувань: сенсор сформував повідомлення, монітор виявив перевищення межі, згенерував аларм, який був оперативно прийнятий обробником. Усі повідомлення було доставлено без втрат, затримка становила не більше 400 мс, що підтверджує відповідність системи вимогам до рішень реального часу. Лічильники в кожному з клієнтів відображали коректне співвідношення відправлених і отриманих повідомлень, що засвідчує стабільність WebSocket-з’єднання та ефективну логіку обробки.

Маршрутизацію повідомлень було додатково протестовано в середовищі Cisco Packet Tracer, де створено віртуальну топологію мережі з усіма логічними зв’язками між вузлами. Результати пінгування та аналіз трафіку TCP підтвердили відсутність втрат, стабільні маршрути та мінімальні затримки. Усі клієнтські вузли – сенсор, монітори та обробник – успішно взаємодіяли із сервером брокера без жодних перебоїв. Цей факт є надзвичайно важливим, оскільки саме стабільність мережевої взаємодії визначає достовірність переданої телеметрії й своєчасність аварійного сповіщення.

Проведена оцінка зручності використання системи показала високу ефективність реалізованого інтерфейсу. Усі вікна мають логічну структуру, чітко відокремлені функціональні блоки та зрозумілі для оператора механізми керування. Завдяки гнучким налаштуванням – таких як зміна температурних порогів, увімкнення сигналізації, підтвердження алармів – система може бути адаптована до різних виробничих умов та режимів експлуатації. За відгуками експертів, інтерфейс системи дозволяє оператору повністю сконцентруватися на

моніторингу технологічного процесу без додаткового навчання або потреби залучати сторонніх фахівців.

Також було здійснено кількісну оцінку експлуатаційних характеристик системи на основі експертного анкетування. Отримані середні бали за всіма критеріями склали понад 4,7 за п’ятибальною шкалою. Найвищу оцінку отримала швидкодія реагування на критичні події, а також легкість масштабування системи. Ці результати підтверджують доцільність обраної архітектури, використання WebSocket як транспортного рівня, а також ефективність механізмів передачі повідомлень і аварійного реагування.

Важливо також зазначити, що проєкт має перспективу подальшого розвитку. Серед напрямків удосконалення можна виділити інтеграцію з базами даних для довготривалого збереження історії показників, розширення кількості параметрів (наприклад, тиск, рівень рідини, вібрація), а також підключення зовнішніх сервісів оповіщення (SMS, Telegram, email). Додатково можна реалізувати механізми резервного копіювання, автоматичної діагностики стану клієнтів і централізованого адміністрування конфігурацій усіх вузлів системи.

Підсумовуючи, можна стверджувати, що поставлену мету дипломної роботи досягнуто повною мірою. Розроблена система демонструє високий рівень функціональності, надійності та гнучкості. Вона є повністю придатною для впровадження в умовах промислового середовища як автономне рішення або як частина більшого комплексу автоматизації. Отримані результати дозволяють не лише захистити роботу як дипломний проєкт, а й сформувати підґрунтя для реального впровадження в середовищах з підвищеними вимогами до моніторингу технологічних параметрів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. 29 с.

2. Навчальний посібник з підготовки кваліфікаційної роботи бакалавра для здобувачів вищої освіти денної і заочної форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми «Автоматизація та комп'ютерно-інтегровані технології»: Навчальний посібник / І. Ш. Невлюдов, В.А. Андрусевич, О. В. Токарєва, С. П. Новоселов, О. В. Сичова. – Харків : Видавництво Іванченка І. С., 2022. – 151 с.

3. Методичні вказівки з підготовки кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. Харків: ХНУРЕ, 2022. 66 с.

4. WebSocket: опис протокола [Електронний ресурс]. – Режим доступу: https://habr.com/en/articles/488654/

5. Мікроконтролер [Електронний ресурс]. – Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Мікроконтролер

6. Програмований логічний контролер [Електронний ресурс]. – Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Програмований_логічний_контролер

7. NodeMCU ESP32 Datasheet [Електронний ресурс]. – Режим доступу: https://www.halloweenfreak.de/arduino/pdfs/D1_R32_ENG.pdf

8. SIMATIC S7-1200 Basic Controllers System Manual [Електронний ресурс]. – Siemens. – Режим доступу: https://assets.new.siemens.com/siemens/assets/api/uuid:2961db0147fb92fd02cb65dea800734f8a911ac5/st70-simatic-s7-1200.pdf

9. Що таке мікросервісна архітектура ПЗ? [Електронний ресурс]. – QALight. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/

10. Як працює WebSocket: приклад на Python [Електронний ресурс]. – Режим доступу: https://habr.com/en/articles/463669/

11. Python Developer Guide [Електронний ресурс]. – Режим доступу: https://www.python.org/doc/

12. RFC 6455 – The WebSocket Protocol [Електронний ресурс]. – Режим доступу: https://tools.ietf.org/html/rfc6455

13. WebSocket Essentials – WebSocket Basics [Електронний ресурс]. – HiveMQ. – Режим доступу: https://www.hivemq.com/WebSocket–essentials/

14. WebSocket WebSocket Client [Електронний ресурс]. – HiveMQ. – Режим доступу: https://www.hivemq.com/demos/websocket-client/

15. Cisco Packet Tracer – Cisco Networking Academy [Електронний ресурс]. – Режим доступу: https://www.netacad.com/courses/packet-tracer

16. Python WebSockets library [Електронний ресурс]. – Режим доступу: https://websockets.readthedocs.io/

17. Python websockets – документація [Електронний ресурс]. – Режим доступу: https://websockets.readthedocs.io/en/stable/

18. JSON Format – офіційний стандарт [Електронний ресурс]. – Режим доступу: https://www.json.org/json–en.html

19. Telegram Bot API Documentation [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api

20. SCADA/HMI Interface Design Best Practices [Електронний ресурс]. – Режим доступу: https://www.scadaprofessor.com/interface–design

21. ESP32 for IoT [Електронний ресурс]. – Режим доступу: https://randomnerdtutorials.com/projects-esp32/

22. Python для IoT-проєктів [Електронний ресурс]. – Режим доступу: https://realpython.com/python-iot/

23. ttkbootstrap documentation [Електронний ресурс]. – Режим доступу: https://ttkbootstrap.readthedocs.io/

24. WebSocket Security Fundamentals [Електронний ресурс]. – HiveMQ. – Режим доступу: https://www.hivemq.com/WebSocket-security-fundamentals/

25. TCP/IP Protocol Suite Overview [Електронний ресурс]. – Cisco. – Режим доступу: https://www.cisco.com/c/en/us/td/docs/internetworking/technology/handbook

26. Правила охорони праці під час експлуатації електронно-обчислювальних машин. - Охрона праці - https://www.dstu.dp.ua/Portal/Data/5/10/5-10-b3.pdf

27. ДСанПіН 3.3.2.007-98. - ДСанПіН 3.3.2.007-98 - https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=6007

28. МВ до лаб. робіт з дисципліни «Основи охорони праці» для студентів усіх напрямів та форм навчання. / Упоряд.: Т.Є. Стиценко, В.А. Айвазов, О.В. Мамонтов. – Харків: ХНУРЕ, 2018.– 120 с.