Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

перший (бакалаврський)
(рівень вищої освіти)

Розроблення програмного модуля для отримання результатів перевірки датчиків автоматизованих систем за стандартними протоколами
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТАКІТ-21-2
Станіслав ДУБИНА
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Автоматизація та комп'ютерно-інтегровані технології
(повна назва освітньої програми)

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

Допускається до захисту
Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я , Дубина Станіслав Сергійович, 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 відповідне джерело.

" 10 " червня 2025 р. Станіслав ДУБИНА

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Автоматизація комп’ютерно-інтергрованих технологій
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАР _____
(підпис)
« 28 » квітня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дубина Станіславу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи розроблення програмного модуля для отримання результатів перевірки датчиків автоматизованих систем за стандартними протоколами
Затверджена наказом по університету від 19.05.2025 р. № 390 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 22.06.2025 р.
3. Вихідні дані до роботи: Тема кваліфікаційної роботи, технічне завдання, вимоги до функціональності модуля перевірки сенсорів; підтримка обміну даними з сенсорами за протоколом Modbus TCP; використання бібліотеки pymodbus для комунікації; очікувані діапазони вимірювань температури та вологості, задані в конфігураційному файлі; структура результатів перевірки (час, значення, статус); механізм генерації тестових даних; передбачене логування запитів та результатів перевірки; реалізація API через FastAPI; розгортання модуля в середовищі Docker.
4. Перелік питань, що потрібно опрацювати в роботі:
4.1 Аналіз структури АСУ
4.2 Роль сенсорів у формуванні первинної інформації
4.3 Класифікація протоколів обміну
4.4 Проєктування аріхтектури програмного модуля
4.5 Реалізація алгоритмів перевірки даних
4.6 Тестування працездатності модуля
4.7 Формування висновків щодо доцільності інтеграції рішення в реальні системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Демонстраційний графічний матеріал у форматі PowerPoint(*.pptx) – 12 ст. формату А4

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 етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Ознайомлення з темою, постановка мети, об'єкта та предмета дослідження	від 28.04.2025 до 22.06.2025	Виконано
2	Аналіз літературних джерел, вивчення структури АСУ, типів сенсорів і протоколів	від 28.04.2025 до 22.06.2025	Виконано
3	Розроблення алгоритму перевірки роботи сенсорів	від 28.04.2025 до 22.06.2025	Виконано
4	Реалізація програмного модуля з використанням FastAPI та pymodbus	від 28.04.2025 до 22.06.2025	Виконано
5	Проведення тестування, аналіз результатів	від 28.04.2025 до 22.06.2025	Виконано
6	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	від 28.04.2025 до 22.06.2025	Виконано
7	Оформлення пояснювальної записки	від 28.04.2025 до 22.06.2025	Виконано
8	Подання роботи на рецензію	від 28.04.2025 до 22.06.2025	Виконано
9	Подання роботи на підпис зав. кафедри	від 28.04.2025 до 22.06.2025	Виконано
11	Подання кваліфікаційної роботи в ЕК	від 28.04.2025 до 22.06.2025	Виконано

Дата видачі завдання 28.04.2025 р.

Здобувач _____ Станіслав ДУБИНА
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ доцент Леонід ІВАНОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 76 с., 1 табл., 7 рис., 2 додатки, 32 джерела.

ЛОКАЛЬНА МЕРЕЖА, ІНЕРЦІЙНІ ДАТЧИКИ, СТАНДАРТНІ ПРОТОКОЛИ, ПРОГРАМНИЙ МОДУЛЬ, ВЕРИФІКАЦІЯ ДАНИХ, FASTAPI, MODBUS TCP, СЕНСОРНІ СИСТЕМИ.

Об’єкт розробки – процес обміну інформацією між сенсорами та програмним забезпеченням в АСУ з метою отримання й перевірки показників.

Предмет розробки – програмні засоби збору та перевірки сенсорної інформації в автоматизованих системах на основі протоколу Modbus-TCP.

Мета роботи – розроблення програмного модуля, що дозволяє отримувати результати перевірки стану сенсорів, які функціонують у складі автоматизованих систем управління, із використанням стандартного протоколу Modbus-TCP.

У першому розділі розглянуто склад автоматизованих систем, принцип роботи сенсорних пристроїв, типи даних і комунікаційні протоколи.

У другому розділі виконано аналіз вимог до програмного забезпечення, визначено недоліки існуючих рішень, обґрунтовано необхідність створення нового модуля та обрано його структуру.

У третьому розділі розроблено алгоритми перевірки датчиків, реалізовано архітектуру модуля на базі FastAPI, pymodbus та асинхронної обробки запитів.

У результаті розроблена програмна система, що забезпечує перевірку значень сенсорів через HTTP-інтерфейс з виведенням результатів у JSON-форматі, підтримує роботу з реальними пристроями та емуляторами, та готова до інтеграції у промислові IT-середовища

ABSTRACT

Explanatory note: 76 pages, 1 table, 7 figures, 2 appendices, 32 sources.

USER INTERFACE, INFORMATION SYSTEM, SOFTWARE MODULE, SENSOR VERIFICATION, STANDARD PROTOCOLS, MODBUS TCP, FASTAPI, INDUSTRIAL AUTOMATION.

Object of development – the process of exchanging information between sensors and software in automated control systems (ACS) for the purpose of receiving and verifying indicators.

Subject of development – software tools for collecting and verifying sensor data in automated systems based on the Modbus-TCP protocol.

Purpose of the work – development of a software module that enables obtaining the results of sensor status verification, which operate as part of automated control systems, using the standard Modbus-TCP protocol.

The first section reviews the structure of automation systems, the functioning of sensors, types of data, and communication protocols.

The second section analyzes software requirements, highlights existing solutions' limitations, justifies the need for a new module, and selects its structure.

The third section presents the development of algorithms and the implementation of the module architecture using FastAPI, pymodbus, and asynchronous processing.

As a result, a software system was developed that performs real-time sensor validation via HTTP, delivers structured JSON responses, supports both physical and simulated devices, and is ready for integration into industrial IT infrastructure.

ПЕРЕЛІК СКОРОЧЕНЬ

АСУ – автоматизована система управління;

ПЗ – програмний засіб;

БП – бізнес-процеси;

BPM – управління бізнес-процесами (Business Process Management);

ІС – інформаційна система;

API – прикладний програмний інтерфейс (Application Programming Interface);

JSON – JavaScript Object Notation;

HTTP – Hypertext Transfer Protocol;

TCP – Transmission Control Protocol;

RTU – Remote Terminal Unit;

Modbus – промисловий комунікаційний протокол;

YAML – Yet Another Markup Language;

SCADA – Supervisory Control and Data Acquisition;

Інтерфейс – користувацький або мережевий інтерфейс модуля.

ЗМІСТ

ВСТУП

Автоматизовані системи управління (АСУ) стали невід'ємною складовою технологічних процесів у промисловості, енергетиці, транспорті, медицині та багатьох інших сферах. Їх ефективне функціонування безпосередньо залежить від якості отримуваної інформації про стан об'єктів керування, яка забезпечується за допомогою датчиків. Збір, обробка та перевірка достовірності даних, що надходять з датчиків, є критично важливою складовою загальної структури АСУ.

У структурі будь-якої автоматизованої системи саме датчики виконують роль первинних джерел даних. Вони здійснюють перетворення фізичних величин (температури, тиску, вологості, рівня, напруги, струму тощо) у електричні сигнали або цифрові значення, які надалі підлягають обробці й аналізу. Достовірність цих вимірювань є запорукою правильності функціонування системи загалом, оскільки на основі цих даних формуються керуючі впливи.Невірні, зашумлені або застарілі значення з датчиків можуть призвести до порушення стабільності роботи системи, помилкових рішень, виходу обладнання з ладу або навіть до надзвичайних ситуацій. Саме тому питання валідації, перевірки та оцінки стану сенсорних пристроїв є предметом особливої уваги як на стадії проєктування АСУ, так і під час її експлуатації.

Однією з важливих тенденцій останніх років стало впровадження стандартних протоколів комунікації для забезпечення уніфікованої передачі даних між різними компонентами системи. Серед таких протоколів найбільш розповсюдженими є Modbus, MQTT, OPC UA, BACnet, які дозволяють реалізовувати як прямий обмін даними, так і забезпечувати побудову динамічних конфігурованих систем. Зокрема, протокол Modbus-TCP є широко використовуваним стандартом у промисловості завдяки простоті реалізації, відкритості та підтримці великою кількістю пристроїв. У практиці побудови АСУ часто виникає потреба в оперативній перевірці працездатності

датчиків – як під час введення системи в експлуатацію, так і в процесі планової діагностики або аварійного аналізу. Враховуючи складність інженерних систем, кількість точок вимірювання та критичність окремих параметрів, перевірка кожного сенсора вручну або за допомогою сторонніх інструментів є ресурсомісткою та неефективною. Це зумовлює необхідність створення спеціалізованих програмних модулів, здатних автоматизувати процес перевірки коректності роботи сенсорів у контексті заданої конфігурації. Водночас, сучасне програмне забезпечення для діагностики стану датчиків часто є закритим, прив’язаним до конкретного обладнання або має громіздкий інтерфейс, що ускладнює його використання в умовах, коли потрібна легка, універсальна й інтегрована система з можливістю масштабування. Зазначене визначає актуальність задачі створення власного інтерпретованого модулю, який працює з відкритими протоколами обміну, реалізує валідацію вимірювань у реальному часі та забезпечує прозорість у конфігурації.

Метою цієї роботи є розроблення програмного модуля, що дозволяє отримувати результати перевірки стану сенсорів, які функціонують у складі автоматизованих систем управління, із використанням стандартного протоколу Modbus-TCP. Розв’язання цього завдання передбачає побудову гнучкої програмної архітектури, яка включає адаптери для протоколів, API-інтерфейс, емулятор сенсорного пристрою, конфігураційне управління, а також тестовий механізм.

Наукова новизна роботи полягає в реалізації адаптивного модулю, що використовує асинхронну архітектуру обробки запитів у зв’язці з синхронними клієнтами протоколів, а також у впровадженні системи перевірки сенсорів, здатної працювати в умовах повної відсутності фізичного обладнання за рахунок симуляції відповідей.

Практична значущість полягає в можливості оперативного використання розробленого модуля для моніторингу стану сенсорів у системах управління будь-якої

складності. Крім того, завдяки відкритій структурі, модуль може бути інтегрований у більші інформаційно-керуючі комплекси, а також доповнений підтримкою інших протоколів без зміни основної логіки.

Об’єктом розробки виступає процес обміну інформацією між сенсорами та програмним забезпеченням в АСУ з метою отримання й перевірки показників.

Предметом розробки є програмні засоби збору та перевірки сенсорної інформації в автоматизованих системах на основі протоколу Modbus-TCP.

Методи розробки включають методи об’єктно-орієнтованого програмування, асинхронні підходи обробки запитів у мережевих середовищах, аналіз структур протоколів промислового зв’язку, симуляцію процесів збору даних, методи тестування функціонального та інтеграційного рівня, а також використання конфігураційних файлів для гнучкого управління структурою системи.

У роботі застосовано мову програмування Python, що дозволила поєднати простоту реалізації з потужними можливостями мережевих і протокольних бібліотек. Зокрема, використано FastAPI для побудови асинхронного веб-інтерфейсу, бібліотеку pymodbus для взаємодії з Modbus-пристроями, Pydantic для моделювання результатів, а також утиліти для тестування, логування та роботи з симуляцією пристроїв.

Структура кваліфікаційної роботи охоплює чотири основних розділи. У першому розділі розглядається структура АСУ та роль сенсорної інформації. Другий розділ присвячено аналізу вимог до програмного забезпечення і недоліків існуючих рішень. Третій розділ містить опис розробленого програмного модуля, його архітектури та логіки роботи. Четвертий розділ подає експериментальні результати перевірки працездатності модуля. Завершують роботу висновки щодо ефективності реалізованого рішення.

Основними завданнями, які розв’язуються в рамках дослідження, є аналіз особливостей сучасних автоматизованих систем управління та ролі сенсорів у структурі таких систем; систематизація типів інформації, яку збирають датчики, і підходів до її обробки; огляд актуальних протоколів для обміну даними між сенсорами і системами збору даних; виявлення недоліків існуючого програмного забезпечення для перевірки сенсорів та обґрунтування необхідності створення нового модуля; побудова алгоритмів перевірки працездатності сенсорів із врахуванням очікуваних діапазонів значень; створення конфігурованої системи із зовнішнім YAML-конфігом для підключення різних сенсорів; розроблення локального емулятора для тестування роботи системи без фізичного підключення обладнання; реалізація REST API для віддаленого отримання результатів перевірки; проведення експериментального тестування працездатності системи; формування висновків щодо ефективності запропонованого підходу та можливих напрямів удосконалення.

Результати роботи можуть бути віднесені до Цілі сталого розвитку 9 "Промисловість, інновації та інфраструктура", зокрема до завдання 9.5, яке передбачає сприяння інноваціям і розвитку промислових ІТ-рішень шляхом впровадження модульного діагностичного програмного забезпечення у сфері автоматизованих систем.

1 АНАЛІЗ СУЧАСНОГО РОЗВИТКУ АВТОМАТИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

1.1 Склад АСУ та принцип її роботи

Автоматизовані системи управління (АСУ) становлять основу сучасних технічних і виробничих процесів, забезпечуючи безперервне управління технологічними об’єктами та процесами з мінімальним втручанням людини. Ефективне функціонування таких систем ґрунтується на правильному проєктуванні їхньої архітектури, адекватному виборі апаратного та програмного забезпечення, а також на повній відповідності взаємодіючих компонентів функціональним і технічним вимогам. У структурному плані будь-яка автоматизована система складається з трьох рівнів, кожен з яких виконує специфічну функцію в контурі управління: рівень збору інформації, рівень обробки та прийняття рішень, а також рівень візуалізації та адміністрування. Основу нижнього рівня формують технічні засоби збору даних, а саме датчики та виконавчі механізми, що розташовані безпосередньо на об’єкті керування. Датчики здійснюють вимірювання фізичних параметрів, таких як температура, тиск, рівень, витрати, швидкість тощо, перетворюючи їх у сигнали, які придатні для подальшої обробки. Виконавчі механізми, своєю чергою, реагують на керуючі сигнали, змінюючи стан об’єкта у відповідності до логіки керування. Таким чином, нижній рівень забезпечує фізичний зв’язок системи з реальним середовищем.

Наступним є рівень логічного управління, де відбувається перетворення сирих даних у прийнятні для аналізу значення, обчислення алгоритмів керування та формування керуючих впливів. Цей рівень реалізується за допомогою контролерів, таких як програмовані логічні контролери (PLC), вбудовані системи управління або комп’ютери реального часу. Контролери приймають вхідні сигнали з польового рівня,

інтерпретують їх згідно з попередньо запрограмованими алгоритмами, порівнюють з установленими порогами та видають сигнали на виконавчі пристрої. У багатьох випадках саме на цьому рівні відбувається попередня фільтрація даних, розрахунок похідних характеристик, виявлення аномалій або реалізація регуляторів, таких як пропорційно-інтегрально-диференціальні (ПІД) схеми. Контролери виконують свій цикл обробки з високою частотою, що дозволяє оперативно реагувати на зміну параметрів процесу.

Верхній рівень автоматизованої системи виконує функції моніторингу, аналізу, архівації даних, формування звітів, а також забезпечення інтерфейсу між системою та обслуговуючим персоналом. Тут використовуються SCADA-системи, сервери збору даних, операторські панелі, а також спеціалізоване програмне забезпечення, яке забезпечує візуалізацію технологічного процесу, відображення поточних та історичних даних, управління подіями та сигналами тривоги. Цей рівень не лише фіксує стан об’єкта в реальному часі, а й дозволяє приймати стратегічні рішення, налаштовувати параметри системи, планувати обслуговування та оптимізовувати ресурси.

Принцип роботи АСУ базується на циклічній взаємодії між зазначеними рівнями. Датчики зчитують інформацію з об’єкта керування, після чого ці дані передаються через відповідні інтерфейси до контролера, який виконує аналіз поточного стану. Якщо параметри виходять за межі допустимих значень або змінюються відповідно до встановленого алгоритму, система генерує відповідні сигнали керування, що передаються на виконавчі пристрої. Ці пристрої, у свою чергу, змінюють параметри технологічного процесу, забезпечуючи його стабілізацію або адаптацію до нових умов. Дані про кожну дію та зміну зберігаються у відповідних масивах пам’яті, архівах або базах даних, з можливістю подальшого аналізу.

Одним із ключових чинників стабільної роботи АСУ є коректна реалізація зв’язку між її складовими. Для цього використовуються стандартизовані протоколи обміну даними, які визначають формат, швидкість, тип і правила взаємодії між пристроями. Найбільш поширеними серед них є Modbus, OPC, MQTT, PROFIBUS, BACnet, CANopen та інші. Наприклад, протокол Modbus дозволяє організовувати комунікацію між головним пристроєм (master) та підлеглими (slave), передаючи числові значення в регістри, які потім зчитуються або змінюються згідно з запитами. Подібна стандартизація значно полегшує інтеграцію нових пристроїв у систему, а також забезпечує надійність і безперервність комунікації.

Варто зазначити, що сучасні АСУ дедалі частіше використовують цифрові технології, хмарні обчислення та елементи штучного інтелекту для підвищення гнучкості та точності управління. Це вимагає побудови адаптивних, масштабованих систем, здатних працювати в умовах неповної або зашумленої інформації. У зв’язку з цим зростає роль верифікації вхідних даних, а також постійного моніторингу стану сенсорних вузлів. Будь-яке відхилення в роботі датчика може мати суттєвий вплив на процес управління, особливо в критичних або безперервних виробничих середовищах. Неправильна або відсутня інформація з одного вузла здатна спричинити спотворення всієї логіки керування, що робить завдання перевірки та контролю сенсорів одним із найпріоритетніших у сучасних системах.

Окрему увагу слід приділити особливостям програмного забезпечення, яке реалізує функції збору, перевірки та обробки інформації. Такий модуль повинен мати гнучку архітектуру, здатність до роботи з різними протоколами, підтримку асинхронної обробки запитів, а також можливість масштабування під різні конфігурації об’єкта. Важливими є також модульність, конфігурованість через зовнішні файли, підтримка стандартів валідації, логування та інтеграції з системами вищого рівня.

У контексті технічної реалізації, до складу типової АСУ (рис. 1.1) входять як апаратні, так і програмні компоненти. Апаратні пристрої включають датчики, виконавчі механізми, інтерфейсні плати, контролери, операторські панелі, сервери. Програмні засоби охоплюють системи збору й аналізу даних, SCADA-програми, конфігураційні утиліти, засоби для побудови користувацьких інтерфейсів та історичних баз даних. Високий рівень інтеграції між цими елементами досягається завдяки використанню єдиної інформаційної моделі, уніфікованих протоколів обміну, а також чіткої структури взаємодії між підсистемами.

Рисунок 1.1 – Архітектура автоматизованої системи управління (АСУ)

Узагальнюючи, автоматизована система управління є складною взаємопов’язаною структурою, що забезпечує збирання, аналіз і використання інформації для прийняття оперативних рішень, які забезпечують ефективне функціонування технічних об’єктів. Її успішна реалізація базується на чітко визначених функціях кожного рівня, злагодженій роботі апаратних і програмних компонентів, стандартизованій комунікації та наявності механізмів перевірки достовірності вхідних даних. Усі ці елементи разом забезпечують високий рівень автоматизації, адаптивність до змін, а також зменшення людського фактора, що є основою надійності сучасних АСУ.

1.2 Порядок отримання інформації та її види

Функціонування автоматизованих систем управління значною мірою залежить від якості, точності та актуальності інформації, яка надходить від об’єкта керування. Саме інформація формує підґрунтя для прийняття технічних і логічних рішень, що реалізуються в межах усіх рівнів автоматизації. Отримання цієї інформації передбачає проходження декількох послідовних етапів, починаючи з фізичного вимірювання параметра об’єкта і завершуючи його представленням у вигляді уніфікованих структур даних, придатних для аналізу, зберігання і використання у керуючих алгоритмах[1].

Початковим джерелом будь-якої інформації в АСУ є фізичні параметри об’єкта керування. Це можуть бути температурні характеристики середовища, тиск у трубопроводі, рівень заповнення резервуара, положення рухомого вузла, швидкість обертання, сила струму, напруга, вологість, освітленість, хімічний склад речовини тощо. Ці параметри не мають цифрового подання в природному вигляді, а отже, потребують перетворення у форму, доступну для аналізу. Це перетворення здійснюється сенсорами або датчиками, що реагують на зміну фізичних або хімічних

властивостей середовища, формуючи при цьому електричні сигнали, оптичні імпульси, частотні коливання або інші види вихідних величин.

Сигнали, сформовані на виході сенсора, можуть мати аналогову або цифрову природу. У разі аналогового виходу мова йде про безперервний електричний сигнал, амплітуда або частота якого пропорційна значенню вимірюваної величини. Для цифрових сенсорів характерна побудова вихідного сигналу у формі кодованих пакетів або імпульсних послідовностей, що вже містять оброблену інформацію. Залежно від типу сенсора і вимог до точності, може бути обраний один із підходів. У разі аналогових сигналів обов'язковим є використання аналого-цифрових перетворювачів, які дискретизують сигнал у часі, квантують його за рівнем і переводять у цифрове представлення.Подальшим етапом у порядку отримання інформації є передавання сигналу від сенсора до пристрою обробки, найчастіше – до контролера або модульного пристрою збору даних. Це передавання може здійснюватися як дротовими, так і бездротовими каналами, з використанням відповідних інтерфейсів. У промислових умовах найчастіше використовуються дротові інтерфейси типу RS-232, RS-485, Ethernet, CAN або специфічні польові шини. Для бездротового зв'язку застосовуються технології Wi-Fi, Bluetooth, LoRaWAN або Zigbee, з урахуванням відстані, енергоспоживання та умов експлуатації. Окрім фізичного каналу зв'язку, важливим є також комунікаційний протокол, що забезпечує структурування, адресацію, верифікацію та інтерпретацію даних. Такі протоколи, як Modbus, OPC UA, MQTT, BACnet, забезпечують єдиний формат обміну та дозволяють поєднувати пристрої різних виробників у єдину систему.

Отримавши цифрове значення вимірюваного параметра, контролер виконує початкову обробку – перевірку на допустимість, усереднення, фільтрацію, порівняння з граничними значеннями або нормалізацію. У разі використання складніших систем можуть застосовуватись методи математичного згладжування, логіки нечіткого виводу

або адаптивної корекції даних. Усі ці процедури спрямовані на те, щоб отримана інформація відповідала вимогам до достовірності, точності, повторюваності й стабільності.Після обробки на рівні контролера інформація може передаватись далі – на верхній рівень системи. Це передавання здійснюється через промислову мережу або IP-мережу на сервер збору даних, SCADA-систему або хмарну платформу. Тут дані зберігаються у базах, аналізуються для виявлення тенденцій, формують архіви, звіти та використовуються для побудови систем візуалізації. Таким чином, інформація, що пройшла шлях від фізичного об’єкта до оператора або адміністратора, може бути подана у вигляді графіків, таблиць, логів подій, індикаторів або текстових повідомлень. Таке представлення значно полегшує сприйняття і прийняття рішень, особливо у багаторівневих та географічно розподілених системах.

Щодо класифікації інформації, яка використовується в АСУ, її можна поділити за декількома критеріями [2]. За походженням інформація поділяється на первинну, тобто безпосередньо зчитану з сенсорів, та похідну, що виникає внаслідок обчислень або логічних операцій. За структурою розрізняють дискретну (булеву), яка відображає наявність або відсутність певного стану, та аналогову, що має безперервний характер і потребує числового аналізу. За способом подання вона може бути текстовою, числовою, графічною або комбінованою. За актуальністю інформація може бути оперативною – що відображає поточний стан системи, або архівною – що фіксує її історію. Усі ці види інформації мають різне значення для різних функцій системи: керування, контролю, аналізу, оптимізації, обслуговування.

У системах реального часу особливо важливою є здатність обробляти і передавати інформацію з мінімальною затримкою. Це досягається за рахунок застосування пріоритетних каналів, буферизації, синхронізації годинників, а також розподілу навантаження між окремими модулями. У випадках критичних для безпеки систем використовуються резервування каналів, дублювання сенсорів, а також

додаткові засоби перевірки вхідних даних, зокрема через крос-перевірку або математичні моделі об’єкта.

Окремим аспектом порядку отримання інформації є перевірка її достовірності. Це включає як апаратну діагностику стану ліній зв’язку, так і програмну перевірку коректності отриманих значень. У програмних модулях можуть реалізовуватись функції визначення некоректних або вихідних за межі значень, розпізнавання аномалій, виявлення нестабільної поведінки або втрати зв’язку. Такі функції є невід’ємною частиною підсистем контролю та забезпечення надійності. У більш складних системах додатково впроваджуються алгоритми штучного інтелекту або машинного навчання для прогнозування виходу сенсора з ладу або виявлення нестандартних режимів роботи.

Отже, порядок отримання інформації в автоматизованій системі є послідовністю логічно пов’язаних етапів – від перетворення фізичних величин до формування цифрових даних, їх обробки, перевірки та візуалізації. У цьому процесі беруть участь як апаратні, так і програмні засоби, а кожен етап має свої специфічні завдання, вимоги та можливості. Видобута інформація є основним ресурсом, на якому базується управління, а її якість визначає точність, стабільність і ефективність функціонування всієї системи.

1.3 Датчики – джерело отримання первинної інформації від об’єкту управління

Основу будь-якої автоматизованої системи управління становить процес збору даних про поточний стан об’єкта керування. Цей процес є вихідною точкою для функціонування всієї системи, оскільки саме на основі отриманої інформації будуються всі подальші дії, пов’язані з аналізом, прийняттям рішень, формуванням команд, регулюванням та оптимізацією. Саме тому ключова роль у структурі АСУ

належить сенсорам, або датчикам, які є первинними перетворювачами фізичних величин у сигнали, придатні для сприйняття й обробки технічними засобами [3].

Датчик у широкому розумінні – це пристрій, що реагує на зміну певної фізичної або хімічної характеристики середовища і формує на цій основі сигнал, який характеризує вимірюване значення. Такий сигнал може мати електричну, оптичну, частотну або іншу природу, залежно від конструкції сенсора і типу перетворення. Головною ознакою якісного датчика є його здатність до точного, стабільного і швидкого відображення змін у навколишньому середовищі. У межах автоматизованих систем датчики є обов'язковими компонентами польового рівня, що здійснюють безпосередній контакт з об'єктом контролю чи технологічним процесом.

Усі сенсори виконують загальну функцію – вимірювання значення певної величини. Проте їх поділяють за багатьма класифікаційними ознаками, зокрема за типом вимірюваної величини, принципом дії, видом вихідного сигналу, способом живлення, інтерфейсом передачі даних, точністю, умовами експлуатації та іншими характеристиками. За принципом дії датчики бувають резистивними, ємнісними, індуктивними, п'єзоелектричними, термоелектричними, оптичними, ультразвуковими, магнітними, механічними. Вибір типу сенсора для конкретного об'єкта залежить від необхідної точності, швидкодії, стійкості до зовнішніх факторів, вартості, а також технологічних обмежень. Залежно від типу фізичної величини, що підлягає вимірюванню, розрізняють температурні, тискові, рівневі, вібраційні, лінійні, кутові, вологісні, швидкісні, оптичні, хімічні, біологічні та багато інших видів сенсорів. Наприклад, температурні сенсори займають провідне місце серед польових пристроїв, оскільки температура є одним із найбільш часто контрольованих параметрів у промисловості, енергетиці, харчовій промисловості, медицині, лабораторних системах тощо. Серед найбільш поширених типів температурних сенсорів варто назвати терморезистори (наприклад, Pt100), термопари (типи J, K, T), а

також цифрові датчики температури, такі як DS18B20 або TMP36. Ці пристрої можуть мати аналоговий або цифровий вихід, різний діапазон вимірювання, спосіб монтажу, захист від вологи та пилу, а також тип інтерфейсу для підключення до системи.

У системах, де необхідно вимірювати тиск у трубопроводах, резервуарах, гідросистемах або газових контурах, використовуються тискові сенсори. Прикладом є Honeywell 26PC (рис. 1.2) або WIKA S-20, які забезпечують точне вимірювання надлишкового або абсолютного тиску з високою стабільністю та захистом від перенавантаження.

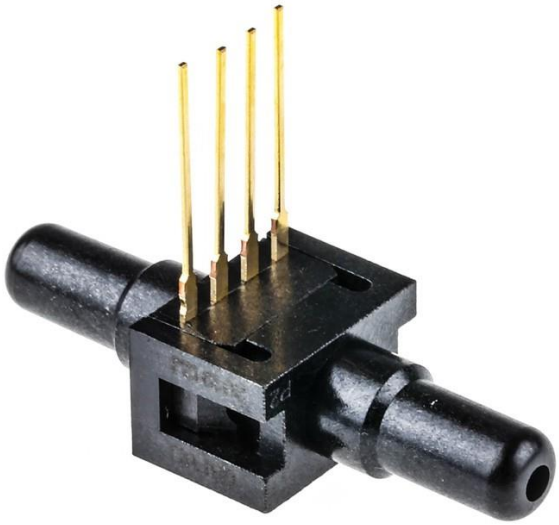

Рисунок 1.2 – Датчик тиску Honeywell 26PC

Такі сенсори можуть мати вихід у вигляді аналогового сигналу 4–20 мА, що є промисловим стандартом, або підтримувати протоколи Modbus RTU або HART для цифрового зв’язку [4]. Залежно від конструкції, тискові датчики можуть бути виконані у мембранному, п’єзорезистивному або ємнісному виконанні.У разі контролю рівня рідини або сипких матеріалів в резервуарах застосовуються ультразвукові, гідростатичні, ємнісні або лазерні сенсори рівня. Вони дають змогу безконтактно або

із зануренням визначити рівень заповнення, запобігти переливам, аварійним зупинкам або порушенням технології. Серед популярних моделей – Siemens SITRANS або Vega PLICS. Подібні сенсори зазвичай мають цифровий вихід і можуть підключатися до контролера через стандартні шини, наприклад, RS-485 або PROFIBUS.

Окрім класичних фізичних сенсорів, у сучасних автоматизованих системах використовуються сенсори складніших характеристик, зокрема багатофункціональні MEMS-датчики (Micro-Electro-Mechanical Systems), що об’єднують у собі кілька сенсорних елементів. До таких належать датчики прискорення, гіроскопи, магнітометри, які часто застосовуються в мобільних, портативних або розподілених системах. Вони здатні фіксувати положення, зміну орієнтації, тривимірні коливання тощо, що особливо важливо в умовах мобільного об’єкта, змінного середовища або складної кінематики.

Наприклад, сенсори типу MPU-6050 (рис. 1.3) широко використовуються в системах стабілізації, робототехніці, моніторингу активності об’єктів.Окрему категорію становлять сенсори з вбудованими протоколами передачі даних, які мають можливість безпосередньо взаємодіяти з центральним модулем без додаткових перетворювачів. Такі пристрої зазвичай називають інтелектуальними датчиками. Вони містять в собі не лише елемент вимірювання, але й мікроконтролер, модуль обробки сигналу, пам’ять та інтерфейс зв’язку. Такі датчики можуть мати вихід у вигляді JSON, працювати через Ethernet, Wi-Fi або NB-IoT, підтримувати протоколи REST, MQTT або CoAP, що значно спрощує інтеграцію в сучасні цифрові системи. Крім того, деякі з них можуть самостійно виконувати функції калібрування, автодіагностики, зберігання історії вимірювань, реагування на аномалії або зміни у конфігурації середовища.

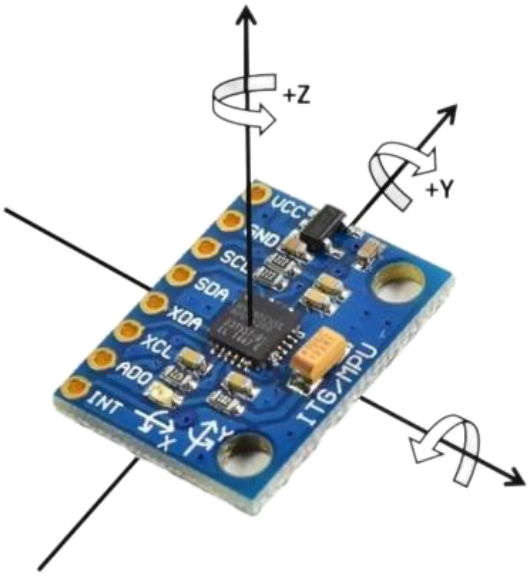

Рисунок 1.3 – Інтегрований сенсор прискорення та гіроскопа MPU-6050

Ключовими характеристиками, які визначають якість сенсора для використання в АСУ, є точність, похибка, стабільність, час відгуку, робочий температурний діапазон, механічна міцність, захист від вологи й пилу, термін служби, каліброваність, енергоспоживання, сумісність із інтерфейсами [5]. У залежності від рівня критичності вимірюваної величини в системі, можуть бути застосовані резервні датчики, сенсори з самотестуванням або подвійним каналом передачі. З огляду на важливість достовірності первинної інформації, особливу увагу приділяють валідації результатів вимірювання. Вона може здійснюватися шляхом перехресного порівняння показників з інших сенсорів, перевірки на відповідність очікуваному діапазону, аналізу стабільності значення протягом часу або логічного співвідношення з іншими параметрами. Також часто використовуються методи виявлення аномалій, визначення шуму, виявлення затримки або нестабільності сигналу. У складних випадках застосовуються математичні моделі процесу, які дозволяють перевірити

правдоподібність результатів вимірювання на основі закономірностей функціонування системи.

Тобто, сенсори є фундаментальним джерелом інформації в автоматизованих системах управління. Від їхньої якості, точності, стабільності та швидкодії залежить не лише коректність прийнятих рішень, але й загальна надійність та ефективність функціонування всієї системи. Тому вибір, підключення, обслуговування та моніторинг сенсорів є ключовими етапами як при проєктуванні АСУ, так і при її експлуатації.

1.4 Аналіз сучасних протоколів для збереження отриманої інформації

У межах автоматизованих систем управління, де об’єктом постійного моніторингу є різноманітні технологічні параметри, ключову роль відіграє процес збереження інформації. Не менш важливою за сам факт вимірювання стає подальша передача, інтерпретація, накопичення та архівація отриманих даних. Безпосереднє збереження інформації можливе лише за умови ефективної передачі від сенсорного вузла до центрального вузла обробки. Саме тому до складу сучасних АСУ включаються протоколи, які відповідають за надійне, структуроване і масштабоване зберігання отриманих значень із сенсорів та виконавчих пристроїв [6].

Протокол у цьому контексті розуміється як формалізований набір правил, що регламентують процес передачі інформації між компонентами системи, зокрема спосіб її адресації, структуру пакету, правила підтвердження доставки, обробку помилок та порядок взаємодії між ініціатором запиту і відповідним пристроєм. Протоколи не лише відповідають за фізичний обмін інформацією між джерелом і споживачем, а й безпосередньо визначають спосіб її збереження, структуру баз даних або буферів, формат метаданих, логіку оновлення інформації, а також механізми

захисту та стиску. Таким чином, протокол є не лише транспортним середовищем, а й логічною основою архітектури даних у системі.

Одним із найпоширеніших у промислових умовах протоколів є Modbus, що належить до категорії master/slave-протоколів із простою структурою. Modbus спочатку був розроблений для обміну даними між контролерами та периферійними пристроями через інтерфейс RS-485, а згодом еволюціонував у TCP-орієнтовану версію – Modbus TCP, яка дозволяє інтегрувати пристрої в IP-мережу. Згідно з логікою цього протоколу, дані зберігаються у так званих регістрах: дискретних вхідних, дискретних вихідних, вхідних регістрах та регістрах зберігання (holding registers). Кожен тип регістру має свою область пам’яті, доступ до якої здійснюється за допомогою функціональних кодів. Це дозволяє системі опитувати сенсори, зчитувати значення, передавати їх у SCADA-систему чи зберігати в базі даних. Протокол не містить вбудованої логіки історичного зберігання, проте його простота і передбачуваність дозволяє використовувати його як основу для побудови систем збору даних із подальшим збереженням через додаткові програмні модулі.

Іншим сучасним протоколом, що забезпечує не лише передавання, а й структуроване збереження, є OPC UA (Open Platform Communications Unified Architecture). На відміну від простих протоколів запит-відповідь, OPC UA реалізує сервісно-орієнтовану архітектуру, в якій кожен пристрій або програмний модуль виступає у ролі сервера або клієнта, що може надавати або споживати дані. Однією з ключових переваг цього підходу є формалізована модель інформації, яка дозволяє не лише передавати значення параметра, а й супутні метадані – одиницю вимірювання, якість, часову мітку, статус, рівень достовірності тощо. Завдяки цьому дані можуть зберігатися не як прості числові значення, а як комплексні об’єкти, що підтримують повну історичність і відстежуваність. OPC UA також забезпечує високий рівень безпеки – підтримуються шифрування, сертифікація, контроль доступу та підпис

повідомлень, що особливо важливо для систем, пов’язаних із критично важливою інфраструктурою.Збереження отриманої сенсорної інформації може здійснюватися також через публікаційно-підписний протокол MQTT (Message Queuing Telemetry Transport), який був спочатку створений для обміну повідомленнями в умовах обмежених ресурсів. MQTT відзначається мінімальним обсягом заголовків, простотою у реалізації та підтримкою великої кількості одночасних підключень. У рамках цього протоколу дані не передаються безпосередньо між пристроями, а надсилаються на брокер – центральний сервер, який розподіляє повідомлення усім зацікавленим учасникам, підписаним на відповідні теми. Кожне повідомлення може містити як значення параметра, так і його часову мітку, статус та інші метадані. Брокер, крім функції маршрутизації, часто має можливість буферизувати повідомлення та зберігати їх у базі даних, забезпечуючи тим самим формування історії змін. MQTT використовується в IoT-проєктах, розподілених сенсорних мережах, системах інтелектуального дому та промисловому Інтернеті речей, де важлива легкість інтеграції, масштабованість та гнучкість у роботі з нестаціонарними пристроями.

Ще одним напрямом є застосування RESTful API на базі протоколу HTTP, коли сенсорна інформація зчитується через стандартні HTTP-запити у форматах JSON або XML. Такий підхід активно використовується у веборієнтованих системах збору даних, хмарних платформах, сервісах візуалізації та аналітики. Збереження даних у цьому випадку реалізується через базу даних, яка підключена до REST-сервера, або через проміжний кеш, що підтримує функції агрегування. Використання протоколів REST дозволяє інтегрувати системи збору даних із мобільними застосунками, зовнішніми інтерфейсами, SCADA через API-шлюзи. При цьому до основної інформації додаються поля метаданих – ідентифікатор сенсора, часові мітки, статус достовірності тощо.У випадках, коли зберігання інформації відбувається

безпосередньо на сенсорному пристрої або периферійному контролері, застосовуються файлові формати з фіксованою структурою – наприклад, CSV, SQLite, HDF5 або спеціалізовані лог-файли. Це дозволяє тимчасово зберігати дані до моменту відправлення у центральну базу або у випадках відмови мережі забезпечити повторну доставку. Такий підхід часто використовується в умовах віддаленого моніторингу, де передача може бути обмежена у часі або залежна від розкладу. Застосування автономного зберігання дозволяє підвищити стійкість системи до відмов і мінімізувати втрати критичних даних.

У контексті довготривалого архівування, сучасні системи все частіше використовують часосерійні бази даних (TSDB – Time Series Databases), такі як InfluxDB, TimescaleDB або Prometheus. Такі бази оптимізовані для збереження значень, що змінюються в часі, із високою частотою та великою кількістю параметрів. Вони мають можливість зберігати численні вимірювання від різних сенсорів з високою точністю, підтримують агрегацію, групування, фільтрацію за часом, а також масштабовану обробку в режимі реального часу. Дані, отримані через будь-який з протоколів, можуть бути перенаправлені до TSDB за допомогою проміжних колекторів, API або шлюзів, що дозволяє реалізувати гнучку архітектуру відбору, обробки та аналізу інформації. Тобто, протоколи, які використовуються в сучасних автоматизованих системах, вирішують не лише задачу передачі, а й комплексне питання збереження інформації. Від вибору протоколу залежить, у якій формі будуть зберігатися дані, як швидко вони будуть доступні для аналізу, наскільки точно буде зафікований часовий контекст, а також які функції безпеки та надійності будуть доступні. Протоколи нового покоління орієнтовані на забезпечення не лише низького рівня затримок, а й повної структуризації даних, їх метаопису, верифікації, шифрування та передачі у вигляді об’єктів, які можна використовувати повторно, без втрати контексту.

Вибір протоколу залежить від багатьох чинників (табл.1.1): типу пристрою, умов експлуатації, кількості даних, частоти оновлення, вимог до надійності, безпеки та сумісності з іншими компонентами системи.

Таблиця 1.1 – Сучасні протоколи збереження та передачі сенсорної інформації

Протокол	Тип архітектури	Спосіб зберігання даних	Формат даних	Підтримка метаданих	Безпека	Переваги
Modbus TCP	Master–Slave	Регістр пам'яті з зовнішнім логуванням	Числові значення (UInt16)	Немає	Не передбачено	Простота реалізації, широке поширення, сумісність із промисловими пристроями
OPC UA	Клієнт–Сервер, SOA	Вбудована модель об'єкта з історією	XML, JSON, бінарний	Є	Підтримується	Повна модель даних, масштабованість, шифрування, історія змін
MQTT	Публікація –підписка (broker)	Буфер у брокері або зовнішній TSDB	JSON або довільний	Частково (опціонально)	Через TLS або інше	Висока масштабованість, мале навантаження, підходить для IoT
REST/HTTP API	Клієнт–Сервер	Через API до бази даних	JSON, XML	Є	HTTPS, автентифікація	Стандартизація, легка інтеграція з веб і хмарними системами
CSV / SQLite	Локальна автономна	Файлова система або локальна БД	Табличні (CSV), SQL	Відсутня	Не передбачено	Мінімальні вимоги до середовища, автономність
Time Series DB	Collector + база даних	Часосерійне сховище з проміжним кешуванням	Line protocol, JSON	Є	Залежить від реалізації	Оптимізація під часові ряди, агрегація, фільтрація, інтеграція з SCADA/BI

Наприклад, для простих систем з невеликим числом сенсорів може бути достатньо використання Modbus TCP, тоді як у розподілених системах з великою

кількістю вузлів краще застосовувати MQTT або OPC UA. У хмароорієнтованих застосуваннях доцільно реалізовувати RESTful API із підтримкою історичних баз та інтеграцією з системами аналітики [7].

Важливо розуміти, що протокол є не лише інструментом передачі, а й фреймворком для зберігання даних, оскільки саме він задає правила структурування, пріоритезації, узагальнення і контекстуалізації інформації. Саме тому під час проєктування програмного модуля, що працює з сенсорною інформацією, необхідно з самого початку визначити, за якими принципами та стандартами будуть зберігатися отримані дані. Це дозволить уникнути дублювання, втрати важливої інформації, а також забезпечить відповідність системи до вимог сумісності, стандартизації та масштабування.

2 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АСУ

2.1 Програмний модуль – елемент програмного забезпечення

У контексті побудови сучасних автоматизованих систем управління функціональність і ефективність усього комплексу значною мірою залежать від архітектури та реалізації програмного забезпечення. Воно виконує роль інтелектуального ядра системи, забезпечуючи інтерпретацію вхідних сигналів, прийняття керуючих рішень, комунікацію між окремими підсистемами, візуалізацію процесів та збереження результатів. Серед усіх компонентів програмного забезпечення особливу роль відіграють окремі програмні модулі, які реалізують конкретні функції й одночасно є елементами загальної логічної структури.

Програмний модуль у структурі АСУ являє собою відносно автономний функціональний блок, що реалізує обробку певного типу даних або виконує визначений алгоритм. Згідно з принципами модульності, кожен такий елемент розробляється як окрема сутність із чітко визначеними межами відповідальності, точками входу й виходу, а також інтерфейсом взаємодії з іншими модулями. Такий підхід дозволяє досягти не лише структурованості системи, але й підвищеної надійності, легкості тестування, зручності у супроводі, масштабуваності та повторного використання компонентів.

Загальна структура програмного забезпечення автоматизованої системи часто побудована за принципами трирівневої або клієнт-серверної архітектури, де кожен рівень має відповідні програмні модулі. Зокрема, на рівні збору даних функціонують драйвери або інтерфейсні модулі, які взаємодіють із сенсорними пристроями, виконуючи зчитування сигналів і їх первинне перетворення у внутрішні змінні. Далі йде шар логічної обробки, в межах якого програмні модулі виконують обчислення, фільтрацію, нормалізацію, діагностику, формування сигналів тривоги або вивідних

параметрів. Нарешті, на верхньому рівні програмні модулі здійснюють візуалізацію, збереження, передавання інформації користувачу або зовнішнім системам.У загальному вигляді, програмний модуль може містити у своєму складі один або декілька алгоритмів обробки, внутрішні структури даних, механізми логування та обробки виключень, а також інтерфейс доступу до зовнішніх ресурсів (наприклад, бази даних, мережеві сокети, файлову систему). Його реалізація базується на вибраній мові програмування, інженерних шаблонах проєктування, прийнятих стандартах розробки та забезпечується згідно з технічними вимогами до системи [8].

Залежно від функціонального призначення, програмні модулі можуть бути класифіковані на кілька умовних категорій. До першої належать модулі збору даних, які відповідають за підключення до сенсорів або периферійних пристроїв, зчитування показників, трансляцію сигналів у внутрішній формат і передачу результатів на обробку. Ці модулі є критично важливими в аспекті коректного формування вхідного потоку даних, тому мають бути максимально стійкими до помилок, втрат зв’язку або збоїв обладнання. Друга категорія включає модулі обробки, які реалізують алгоритмічну частину управління – розрахунок показників, логічну обробку, фільтрацію, аналіз відхилень, формування керуючих впливів. Їхнє завдання полягає у перетворенні вхідних даних у дії, що матимуть реальний вплив на об’єкт керування. Третя категорія охоплює модулі взаємодії з користувачем або зовнішніми системами – до неї належать блоки візуалізації, звітності, API-інтерфейси, SCADA-панелі та інші інструменти. Ці модулі мають не лише відображати поточний стан, а й надавати зворотний зв’язок, гнучкість у налаштуванні параметрів, інформативність та інтеграцію з суміжними платформами. Особливу роль у сучасних системах починають відігравати модулі, що реалізують функції моніторингу, прогнозування, виявлення аномалій, верифікації даних або машинного навчання. У таких модулях реалізуються складні математичні моделі, засновані на статистичних закономірностях, часових

рядах або нейромережах, що дозволяє автоматизовано визначати несправності сенсорів, спрогнозувати зміну стану об’єкта, сформувати сигнали технічного обслуговування чи оптимізації. Такі програмні модулі мають розвинену внутрішню структуру, містять декілька шарів обробки, навчаються на історичних даних і взаємодіють з базами знань або віддаленими сервісами [9].

На відміну від монолітного підходу, коли уся логіка вбудовується у єдине ПЗ, модульна архітектура дозволяє розділити функціональність на незалежні компоненти, кожен із яких має окремий життєвий цикл розробки, тестування та підтримки. Наприклад, зміна алгоритму перевірки датчика не вплине на роботу модулю візуалізації, якщо між ними існує чітко визначений інтерфейс взаємодії. Це суттєво полегшує оновлення, впровадження нових функцій, масштабування системи та адаптацію під нове обладнання або специфікації. У контексті промислових рішень модульність також дозволяє забезпечити розмежування доступу, резервування критичних блоків, а також паралельну роботу в умовах розподілених або кластерних конфігурацій.З технічної точки зору, реалізація програмного модуля передбачає дотримання певного програмного контракту – опису функцій, методів, структур даних та поведінки у стандартних і виключних ситуаціях. Умовою коректної інтеграції є сумісність цього контракту з вимогами системи, що включає не лише формат даних, але й часові обмеження, обсяг ресурсів, вимоги до надійності та обробки збоїв. Програмний модуль має бути тестований як ізольовано, так і в умовах реального середовища, з використанням імітаційного обладнання, симуляторів або стендів.

Інструменти реалізації програмних модулів можуть відрізнятись залежно від мови програмування, платформи, рівня системи та специфікації. У середовищах реального часу (наприклад, для контролерів) модулі часто пишуться мовами низького рівня, такими як C, C++, Structured Text. У SCADA-системах модулі можуть реалізовуватись у вигляді скриптів, конфігурацій або візуальних блоків. У

веборієнтованих системах використовуються Python, JavaScript, Node.js, Go, де модулі інтегруються через REST API, сокети або брокери повідомлень. У випадку хмарних рішень застосовуються мікросервіси, контейнери, безсерверна архітектура, де кожен модуль виконується як окрема функція або ізольований сервіс.

Загальні вимоги до програмного модуля як частини АСУ включають стабільність у роботі, стійкість до виключних ситуацій, ефективне використання ресурсів, зрозумілу структуру коду, наявність логування, документованість, можливість конфігурації, масштабування і підтримку різних середовищ виконання. Крім того, критичним аспектом є безпека – захист від несанкціонованого доступу, перевірка цілісності, захищена взаємодія з іншими компонентами.Важливо розуміти, що програмний модуль не функціонує ізольовано – його робота залежить від правильності конфігурації, параметрів середовища, сумісності з іншими модулями, а також відповідності до вимог усього комплексу АСУ. Тому його проєктування повинно здійснюватися з урахуванням загальної архітектури системи, топології даних, специфіки об’єкта, вимог до швидкодії та достовірності.

Зокрема, якщо йдеться про програмний модуль перевірки сенсорів, такий як реалізований у рамках цієї кваліфікаційної роботи, він має відповідати додатковим вимогам. Це підтримка протоколів обміну (наприклад, Modbus TCP), можливість масштабування під різну кількість сенсорів, асинхронна обробка запитів, підтримка конфігураційного файлу, перевірка значень на межі допустимого діапазону, обробка помилок з виведенням причини, формування уніфікованої структури відповіді з ідентифікатором, міткою часу, значенням, статусом та повідомленням про помилку. Крім цього, модуль має інтегруватися з інтерфейсом (API), що дозволяє викликати перевірку через HTTP-запити, а також підключатися до симулятора або реального обладнання [10].

Отже, програмний модуль у складі АСУ є ключовим функціональним елементом, що забезпечує обробку, контроль і трансляцію інформації відповідно до заданих алгоритмів. Його роль зумовлена потребою в модульності, гнучкості, повторному використанні та підвищеній надійності програмного забезпечення. Успішна реалізація модулів є основою побудови масштабованих, керованих і ефективних систем автоматизації.

2.2 Існуючі недоліки програмних модулів отримання інформації

У структурі автоматизованих систем управління (АСУ) програмне забезпечення для збору та обробки сенсорної інформації відіграє ключову роль. Саме воно забезпечує взаємодію між апаратними компонентами польового рівня (датчиками, контролерами, інтерфейсними пристроями) і системами верхнього рівня (SCADA, MES, ERP). Незважаючи на активний розвиток засобів промислової автоматизації, низка недоліків у вже існуючих рішеннях значно ускладнює ефективну інтеграцію, масштабування та супровід таких модулів. У цьому підпункті розглянемо основні проблеми, притаманні типовим програмним засобам, що використовуються для отримання, валідації та подальшої передачі первинної інформації від сенсорів у межах АСУ.

Одним з головних недоліків є закритість та низька адаптивність багатьох програмних модулів. Більшість наявних рішень реалізовано у вигляді закритих платформ, які не передбачають внесення змін до внутрішньої логіки функціонування, обмежують доступ до сирих даних, або потребують придбання ліцензій на додаткові функції. Це призводить до складнощів під час інтеграції з існуючими інформаційними системами, особливо коли мова йде про різнотипні або нестандартні сенсори, які не входять у список підтримуваних пристроїв. Внаслідок цього розробникам доводиться

створювати проміжні адаптери, імплементації протоколів або модифікувати архітектуру існуючих модулів, що веде до втрати часу, ресурсів і стійкості системи.

Ще однією проблемою є обмежена підтримка протоколів комунікації. У багатьох готових рішеннях підтримуються лише декілька комунікаційних стандартів (наприклад, Modbus RTU/TCP, OPC Classic), що не відповідає сучасним вимогам до гнучкості систем. При цьому новіші протоколи – такі як OPC UA, MQTT, RESTful API – часто або не підтримуються зовсім, або реалізовані частково без урахування специфіки. Це унеможливлює повноцінну інтеграцію таких модулів в розподілені системи, хмарні інфраструктури або IoT-платформи. Крім того, навіть за наявності підтримки популярних протоколів, часто спостерігаються труднощі з реалізацією паралельної або асинхронної обробки запитів, що знижує продуктивність і стійкість модулів під час роботи з великою кількістю сенсорів [11].

Суттєвою вадою є також відсутність повноцінного механізму перевірки достовірності даних. У багатьох системах отримана інформація передається до SCADA або баз даних без попередньої валідації – без перевірки діапазонів значень, частоти оновлення, стабільності сигналу чи наявності аномалій. Такий підхід не дозволяє виявити збої в роботі сенсорів на ранньому етапі та спричиняє накопичення помилкової інформації в архівах. Це, у свою чергу, ускладнює подальший аналіз, прогнозування, технічну діагностику й автоматизоване прийняття рішень. Деякі комерційні продукти пропонують базові функції перевірки, проте вони, як правило, обмежуються статичними порогами або фільтрами за шаблоном, не враховуючи динаміку процесу чи логічні зв’язки між параметрами.

Варто також зазначити, що значна частина модулів для отримання інформації створюється з урахуванням конкретного обладнання або апаратної платформи, що обмежує можливість повторного використання. Програмне забезпечення, вбудоване у промислові контролери або операторські панелі, часто є монолітним і не передбачає

винесення частини логіки в окремі сервера, контейнери чи хмарні служби. Це суперечить принципам сучасної програмної інженерії, які передбачають модульність, масштабованість, гнучкість у розгортанні та обслуговуванні.Іншим поширеним недоліком є обмежена підтримка конфігураційності. У багатьох випадках структура сенсорів, їхні адреси, діапазони значень та методи обробки жорстко закодовані в програмному коді модуля. Зміна будь-якого параметра (наприклад, заміна сенсора, коригування граничних значень або перенесення на інший IP-адрес) вимагає редагування і перекомпіляції коду, що є недопустимим у промислових умовах. Натомість використання зовнішніх конфігураційних файлів, YAML або JSON-структур, значно полегшило б налаштування системи, оновлення структури об’єкта без зупинки модуля, а також забезпечило зручність підтримки з боку інженерів.

У контексті розробки та тестування, слід згадати ще один значущий недолік – відсутність симуляторів або емуляторів пристроїв у складі модулів. Це особливо критично в процесі відлагодження або тестування програмної частини системи до моменту фізичного підключення датчиків. Без змоги симулювати відповіді пристроїв (наприклад, Modbus-регістри з даними) неможливо перевірити правильність алгоритмів валідації, логіки зчитування, обробки помилок, а також коректність побудови інтерфейсу. Розробники часто змушені створювати власні тимчасові рішення або тестувати на реальному обладнанні, що збільшує витрати часу, ускладнює автоматизоване тестування та знижує надійність результату [12].

Також днією з головних проблем є відсутність підтримки сучасних форматів даних та моделей подання результатів. У багатьох випадках результат зчитування формується у вигляді неструктурованого тексту або значення без контексту (наприклад, просто число), що не дає змоги зберігати супутню інформацію – часову мітку, статус перевірки, ідентифікатор пристрою тощо. Натомість сучасні системи потребують представлення даних у форматах, придатних до подальшої обробки –

JSON, XML, protobuf або спеціалізованих об'єктних моделей. Це особливо важливо для інтеграції з API, побудови графічного інтерфейсу або передачі даних у хмарні служби для аналітики [13]. Не менш суттєвим недоліком є відсутність інтерфейсу для інтеграції з зовнішніми системами. У багатьох існуючих рішеннях передбачено лише локальну передачу інформації – у файл або базу даних. Проте сучасна практика автоматизації передбачає наявність REST API, WebSocket-інтерфейсів або брокерів повідомлень, які дозволяють динамічно отримувати результати, передавати їх до панелей оператора, інтегрувати в ERP/SCADA/BI-системи. Відсутність таких можливостей обмежує гнучкість, зменшує масштабованість рішення, робить його ізольованим у межах однієї підсистеми.

Окремо варто згадати проблему недостатньої документації та слабкої підтримки з боку розробників. Комерційні продукти часто мають скорочені інструкції, орієнтовані на базове налаштування, і не розкривають повний набір можливостей чи обмежень. Відкриті або напіввідкриті модулі, розроблені спільнотами, можуть мати неповну або застарілу документацію, що ускладнює їх використання на практиці. Як наслідок, інженери, що займаються впровадженням АСУ, змушені витрачати значні ресурси на самостійне дослідження функціоналу, написання обгорток, виправлення конфліктів версій та інші низькорівневі задачі, що не повинні бути в зоні їх відповідальності.

Також у деяких модулях спостерігається низький рівень надійності та відсутність обробки помилок. При зникненні зв’язку, помилці в зчитуванні або отриманні нульових даних модулі можуть зависнути, аварійно завершити роботу або, що гірше, передати некоректну інформацію в систему без попередження. Відсутність механізмів автоматичного повторного запиту, логування помилок, повідомлень про несправності значно ускладнює діагностику й супровід таких рішень.

2.3 Обгрунтування необхідності розроблення програмного модуля.

У структурі будь-якої автоматизованої системи управління критично важливим елементом є модуль, що виконує перевірку, обробку та збереження даних, отриманих від сенсорів. Цей елемент безпосередньо відповідає за достовірність усієї вхідної інформації, на основі якої формуються команди керування, сигнали тривоги, діагностичні повідомлення та інші функціональні реакції системи. У випадку, коли вхідні дані є спотвореними, застарілими або зовсім відсутні, автоматизована система втрачає здатність до адекватного реагування, що може призвести до порушення технологічного процесу, аварійного зупинення обладнання або навіть до загрози безпеці обслуговуючого персоналу. Тому перевірка стану сенсорів та контроль за відповідністю їх показників до очікуваних норм є не лише бажаним доповненням, а й необхідною складовою кожної системи, де застосовуються сенсорні пристрої.

Попри те, що ринок сучасного промислового програмного забезпечення пропонує низку рішень, які виконують базову функцію збору та візуалізації інформації з сенсорів, значна частина цих продуктів має закриту архітектуру, обмежені можливості конфігурації, високий рівень залежності від конкретних виробників обладнання або програмного середовища, а також недостатню підтримку протоколів, які використовуються у відкритих або мультипротокольних середовищах. Крім того, багато існуючих систем мають занадто загальний підхід до обробки даних, не враховуючи специфіку кожного типу сенсора, особливості його розміщення, тип вимірюваної величини або вимоги до точності та швидкості зчитування. У результаті розробник або оператор АСУ часто стикається з необхідністю створення додаткових засобів контролю, які б перевіряли не лише наявність даних, а й їхню коректність, повноту, межі допустимих значень та характер зміни у часі.

З іншого боку, наявні програмні рішення, які дозволяють виконувати розширену валідацію та обробку даних, як правило, є надлишково складними, потребують значних обчислювальних ресурсів, містять велику кількість непотрібних або дубльованих функцій і не підходять для локального розгортання на невеликих об’єктах. Це створює додаткове навантаження на ІТ-інфраструктуру підприємства, збільшує витрати на ліцензування та обслуговування програмного забезпечення, а також ускладнює адаптацію під конкретну виробничу або наукову задачу. Також не слід забувати про випадки, коли автоматизована система розробляється з нуля, із залученням специфічних компонентів, підключених через відкриті протоколи, такі як Modbus TCP або MQTT. У таких ситуаціях інтеграція з готовими ПЗ-продуктами вимагає створення додаткових адаптерів, написання проксі-шарів або застосування сторонніх бібліотек, що суперечить принципам гнучкості, модульності та мінімалізму. У цьому контексті виникає потреба у створенні власного програмного модуля, що виконує перевірку стану сенсорів, зчитує значення через відкриті промислові протоколи, аналізує отримані дані у контексті допустимих діапазонів, сигналізує про наявність або відсутність помилок і видає структуровану відповідь у зручному форматі. Такий модуль має бути легким, адаптивним, легко налаштовуваним та сумісним із сучасними веб-інтерфейсами або SCADA-системами. Крім того, він має підтримувати як роботу з реальними сенсорами, так і взаємодію з симуляторами або тестовими середовищами, що особливо важливо на етапах проєктування, налагодження та діагностики системи [14].

В умовах гнучких вимог до автоматизованих рішень, зростання популярності відкритих систем, а також поширення технологій промислового Інтернету речей (IIoT), потреба у модулях такого типу є особливо актуальною. Вона зумовлена як зростанням складності об’єктів автоматизації, так і підвищенням очікувань до точності та надійності систем у цілому. З урахуванням цього, необхідно створити

універсальний, протоколонезалежний модуль перевірки сенсорів, який би забезпечував логічну перевірку зчитуваних даних, підтримував гнучке конфігурування параметрів, був незалежним від апаратної реалізації та мав можливість інтеграції з іншими модулями системи (рис. 2.1).

Підтвердженням такої необхідності є прикладні задачі, в яких було виявлено обмеження наявних засобів. Наприклад, у низці розробок, що базуються на використанні сенсорів температури або тиску, неодноразово виникали ситуації, коли сенсор продовжував передавати однакові значення протягом тривалого часу, хоча фізично параметри мали змінюватися. У більшості випадків наявне програмне забезпечення не фіксувало таких аномалій, оскільки сам факт отримання значення сприймався як свідчення справності сенсора [15]. У реальності ж мала місце апаратна несправність або обрив кабелю, який не проявлявся до моменту критичної зміни параметра. Саме для виявлення таких ситуацій необхідно впроваджувати логіку перевірки зміни параметра у часі, а також відстеження стабільності чи зависання значень.

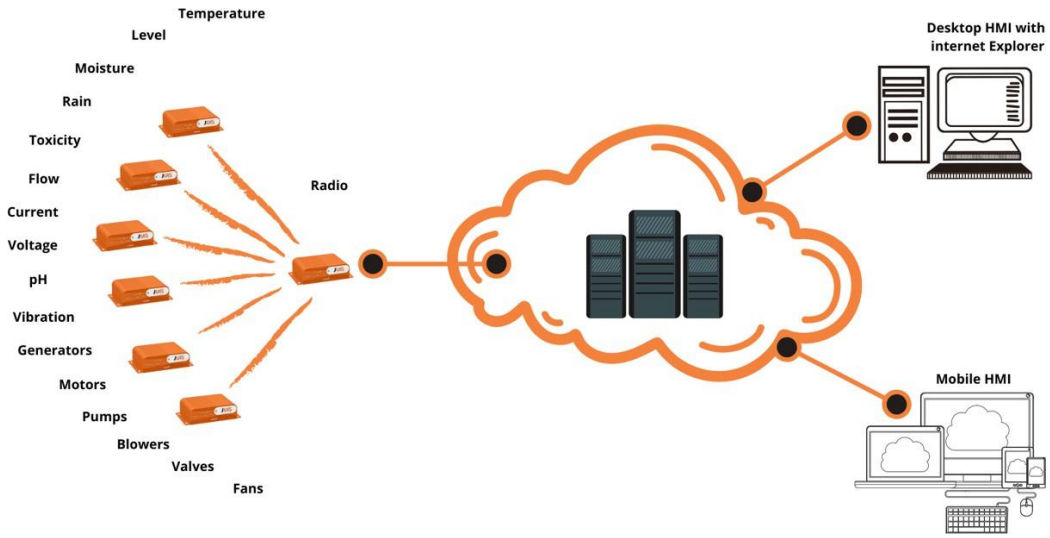

Рисунок 2.1 – Загальна схема збору сенсорних даних у хмарну систему без локальної перевірки значень

Аналогічно, для мобільних сенсорних систем або розподілених вузлів вимірювання, що функціонують у нестабільному мережевому середовищі, особливо важливою є можливість визначати відсутність зв’язку, перевіряти часову мітку даних та виявляти втрати пакету. Ці функції практично відсутні в традиційних SCADA-системах або реалізовані на надмірно низькому рівні, що не дозволяє гнучко керувати логікою перевірки та реагування. Програмний модуль, який реалізовує таку перевірку, має бути побудований за принципом незалежності від конкретного протоколу, із підтримкою тайм-аутів, повторних спроб, валідації часових міток, обробки виключень та логування результатів [16].

Не менш важливою є можливість тестування модулю в умовах, максимально наближених до реальних, без підключення фізичного обладнання. Це особливо актуально на етапі розробки, коли апаратна частина ще не змонтована, або у випадках, коли система підлягає технічному обслуговуванню й частина сенсорів відключена. У таких випадках доцільним є використання симуляторів, які формують тестові значення, підтримують стандартний протокол, зберігають логіку поведінки сенсора та забезпечують зворотний зв’язок на запити програмного модуля. Власна реалізація модуля перевірки повинна бути сумісною з такими симуляторами, підтримувати їхнє налаштування через конфігураційні файли, та дозволяти виконання повного тестового циклу без необхідності змінювати основний код системи.

Обґрунтування розробки власного програмного модуля перевірки сенсорів базується на низці практичних, технічних та концептуальних чинників. По-перше, існуючі інструменти не забезпечують необхідного рівня гнучкості, адаптивності та прозорості. По-друге, потреба у масштабованості, тестованості та протокольній незалежності потребує індивідуального підходу до побудови логіки перевірки. По-третє, модуль має не лише забезпечити передачу даних, але й їх аналіз, перевірку,

структурування, вивід результату у зручному вигляді та можливість інтеграції з іншими компонентами. Саме ці вимоги і зумовлюють необхідність створення універсального, конфігурованого, легко розширюваного та інтегрованого рішення, яке може застосовуватись у широкому спектрі автоматизованих систем – від простих локальних до комплексних хмароорієнтованих.

2.4. Вибір структури програмного модуля

Проєктування ефективного програмного модуля для автоматизованої системи управління потребує чіткого визначення його внутрішньої структури, функціональної логіки та способів взаємодії з іншими компонентами системи. Структура програмного модуля не є суто внутрішньою особливістю реалізації; вона безпосередньо впливає на стабільність, масштабованість, підтримуваність і розширюваність програмного забезпечення. У випадку модулів, відповідальних за отримання і перевірку сенсорних даних, ці чинники мають особливо важливе значення, оскільки йдеться про обробку інформації в режимі близькому до реального часу, із високим рівнем надійності та критичною чутливістю до помилок або спотворень.

На етапі вибору структури модуля слід виходити з кількох вихідних умов: характеру оброблюваних даних, типу підключених пристроїв, обмежень апаратної платформи, вимог до масштабованості та розширення функціональності, наявності стандартів інтеграції з іншими підсистемами, підтримки конкретних протоколів зв’язку, а також здатності до тестування та налагодження в автономному режимі. У випадку реалізації модуля для перевірки даних сенсорів, які працюють за протоколом Modbus TCP, основна задача полягає у зчитуванні значень з регістрів, інтерпретації результатів, перевірці відповідності вказаним межам і формуванні уніфікованої відповіді з результатами перевірки [17].

Найбільш доцільною з точки зору підтримуваності та повторного використання є модульна або компонентна архітектура (рис. 2.2). Такий підхід дозволяє розділити логіку на окремі ізольовані частини, кожна з яких виконує одну чітко визначену функцію. У межах такого підходу ключовим є не лише розподіл коду на файли, а й розмежування відповідальностей між логічними підсистемами: адаптація до протоколу, обробка значення, перевірка правильності, форматування результату, логування, передача через API тощо. Це дозволяє при зміні одного компонента уникнути впливу на інші частини системи, що, своєю чергою, полегшує підтримку та розвиток програмного забезпечення в довготривалій перспективі.

Для реалізації обраного функціоналу було прийнято рішення про побудову структури програмного модуля на основі трьох основних рівнів: адаптаційного, логічного та інтерфейсного. Кожен із цих рівнів виконує власну роль і взаємодіє з іншими виключно через стандартизований інтерфейс. На найнижчому рівні функціонує адаптер, що інкапсулює логіку комунікації з конкретним типом сенсорного пристрою. У випадку реалізації через Modbus TCP, цей компонент відповідає за встановлення TCP-з’єднання, виконання запиту до вказаного регістра, обробку відповіді, контроль помилок на транспортному рівні та завершення сесії. Такий підхід дозволяє замінити або розширити підтримку інших протоколів у майбутньому без зміни всієї системи – достатньо реалізувати новий адаптер, що імплементує однаковий інтерфейс.

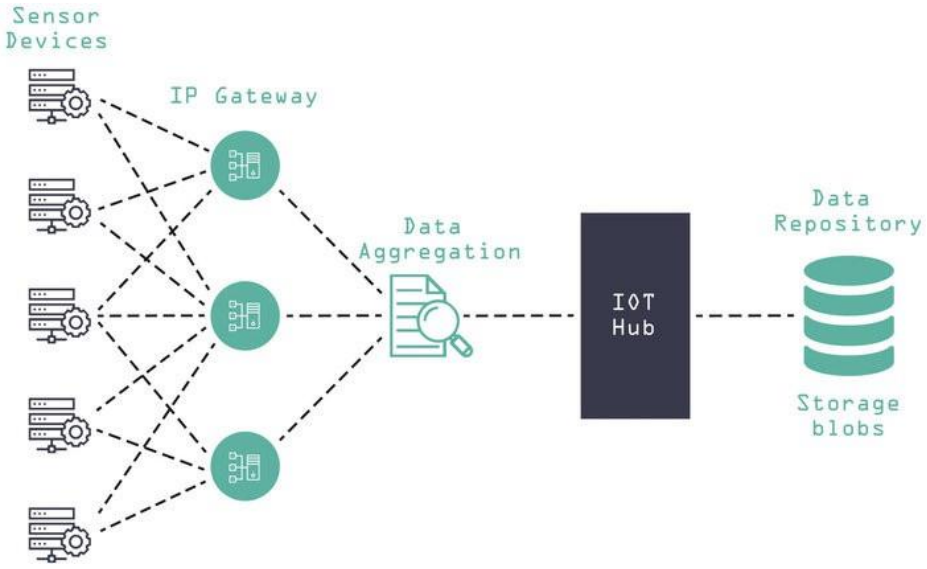

Рисунок 2.2 – Загальна архітектура програмного модуля обробки сенсорних даних у модульному середовищі

Наступний рівень – логічний, або обчислювальний, відповідає за інтерпретацію отриманих даних, їх перевірку згідно з допустимими межами, визначення статусу перевірки (успішно/помилка), а також обробку виключних ситуацій. Тут реалізуються правила оцінювання значення, наприклад, перевірка на вихід за межі, наявність нульового або негативного значення у випадках, коли це фізично неможливо, стабільність сигналу протягом кількох запитів тощо. Також саме на цьому рівні формуються повідомлення про помилки або попередження, які потім можуть бути використані у логах, інтерфейсі оператора або при передачі в SCADA.

Інтерфейсний рівень забезпечує взаємодію з зовнішнім середовищем. У випадку реалізації HTTP API, цей рівень представлений через веб-сервер, який приймає запити, ініціює виклики методів логіки та адаптера, та формує відповідь у форматі JSON. Розподіл між логікою і веб-інтерфейсом дозволяє змінювати спосіб доступу

(наприклад, перейти з HTTP на MQTT або WebSocket) без зміни основного функціоналу [18]. Крім того, інтерфейсний рівень також може виконувати додаткові функції, такі як автентифікація, ведення журналів звернень, обмеження за частотою запитів, обробка параметрів конфігурації тощо.

Однією з критичних умов при виборі структури було забезпечення конфігурованості – можливості задавати параметри сенсорів, тип протоколу, IP-адресу, порт, адресу регістра, очікувані мінімальне і максимальне значення тощо. З цією метою реалізовано зовнішній конфігураційний файл у форматі YAML, який завантажується під час запуску системи. Це дозволяє змінювати параметри роботи без редагування коду, а також використовувати один і той самий модуль для перевірки кількох різних сенсорів. Кожен сенсор описується як окрема конфігураційна секція з ідентифікатором, що значно спрощує адміністрування системи.

Додатковим аргументом на користь обраної структури є можливість локального тестування всіх компонентів окремо. Завдяки поділу на рівні, з’являється можливість виконувати модульне тестування адаптера, логіки, API, симулювати відповіді від сенсорів або підключати симулятор у вигляді окремого процесу. Таке розмежування сприяє підвищенню надійності системи, дозволяє виявляти помилки ще до виходу на експлуатацію, а також полегшує подальшу інтеграцію в більші комплекси. Наприклад, якщо виникне потреба у зберіганні результатів у базі даних або надсиланні повідомлень у Telegram/Slack, ці функції можуть бути реалізовані як окремі модулі, які підключаються до існуючої архітектури без модифікації ядра.

З точки зору технологій, структура реалізована із застосуванням мови програмування Python, фреймворку FastAPI для створення HTTP API, бібліотеки pymodbus для роботи з Modbus TCP, pydantic для визначення структури даних, та YAML-парсера для обробки конфігураційного файлу. Це дозволяє створити модуль, що є одночасно легким, читабельним, легко супроводжуваним і придатним

для розгортання у середовищах із обмеженими ресурсами. Усі компоненти структуровані у вигляді окремих файлів із логічною відповідністю каталогам: adapters, core, api, sim, tests. Такий поділ забезпечує не лише зрозумілість структури коду, а й дотримання принципів чистої архітектури, що особливо важливо у проектах з потенціалом подальшого масштабування.

Вибір структури програмного модуля є результатом аналізу вимог до функціональності, надійності, гнучкості, підтримуваності, тестованості та можливості інтеграції в ширший програмно-технічний комплекс. Застосування принципів модульності, розділення обов’язків, підтримки конфігурації та незалежного тестування дозволяє створити ефективне, розширюване і стабільне рішення для перевірки сенсорних даних у складі автоматизованої системи управління.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО МОДУЛЯ ДЛЯ ОТРИМАННЯ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ ДАТЧИКІВ

3.1 Алгоритм перевірки правильності роботи датчиків

У межах функціонування автоматизованих систем управління на основі сенсорних даних ключовим елементом надійності всієї системи є перевірка коректності значень, що надходять від сенсорів. Наявність спотвореної, відсутньої або некоректно інтерпретованої інформації може спричинити помилкове прийняття рішень, що, у свою чергу, призводить до порушення технологічного процесу або до незворотних наслідків. Саме тому в системі, яка приймає рішення на основі сигналів із датчиків, повинен бути впроваджений алгоритм верифікації, метою якого є відсіяти недостовірні, нефізичні або помилкові значення ще до моменту їх передачі у вищі логічні шари чи системи керування.

Загальна логіка перевірки передбачає, що кожен сенсор має визначений набір характеристик, які можна використати для об’єктивної оцінки його працездатності. До таких характеристик належать ідентифікатор пристрою, фізичні межі вимірюваної величини (мінімум та максимум), тип даних, очікувана частота оновлення, похибка, стабільність сигналу в часі, а також наявність часової мітки. На основі цих атрибутів розробляється універсальний механізм перевірки, який може бути застосований до широкого спектру сенсорів різного типу.

В основі реалізації перевірки лежить послідовність логічних дій, що охоплюють етапи ініціації запиту, зчитування значення, масштабування сирого цифрового сигналу, перевірку на фізичну допустимість, визначення статусу й формування відповіді (рис. 3.1). На початковому етапі ініціюється з’єднання з сенсором або симулятором, що забезпечується через адаптерний інтерфейс. У контексті реалізації за допомогою протоколу Modbus TCP, це означає встановлення TCP-з’єднання з IP-

адресою пристрою, ініціація запиту читання з вказаного регістра (як правило, holding register), та отримання у відповідь значення у вигляді 16-бітного беззнакового цілого числа. Сире значення, отримане з регістра, не може бути інтерпретоване напряму без масштабування. Більшість сенсорів передають інформацію у форматі, який потребує додаткового перетворення – наприклад, значення температури 25,4°C може передаватися як 254 у вигляді цілого числа. У такому випадку алгоритм повинен масштабувати значення шляхом ділення на коефіцієнт (наприклад, 10), що задається у конфігурації. Після цього значення переводиться у числовий формат з плаваючою точкою й передається на наступний етап перевірки [19].

Ключовим етапом є перевірка на допустимість значення. У конфігурації для кожного сенсора визначено діапазон допустимих фізичних значень – мінімум і максимум. Отримане значення порівнюється з цими межами, і якщо воно виходить за межі – позначається як некоректне. Цей механізм дозволяє виявити випадки, коли сенсор вийшов з ладу, передає шумові значення, або на лінії зв’язку виникли перешкоди. У додаток до цього можуть перевірятись і інші характеристики, зокрема чи не є значення нульовим, негативним (у випадках, коли така величина фізично не має бути від’ємною), повторюваним із попередніми запитами тощо.

Після виконання перевірки формується відповідь, яка включає кілька ключових параметрів. До них належать унікальний ідентифікатор сенсора, значення вимірювання після масштабування, статус перевірки (істинне або хибне), повідомлення про помилку (якщо така є), а також час отримання даних. Такий підхід дозволяє отримати не просто «сире» значення, а сформовану діагностичну структуру, яка може бути легко інтерпретована як на рівні API, так і на рівні операторського інтерфейсу.

Окрему увагу в алгоритмі приділено обробці виняткових ситуацій. Якщо з’єднання з сенсором неможливе (наприклад, через відсутність мережі або

недоступність IP), система фіксує відповідну помилку, зазначаючи її в полі error. Те саме відбувається у разі, якщо отримане значення не може бути прочитане з регістра, або має неправильний формат. Усі такі події логуються в окремий журнал, що дозволяє здійснювати аудит працездатності всієї системи в довгостроковій перспективі.Архітектурно, реалізація алгоритму побудована на розділенні відповідальностей. Компонент адаптера відповідає виключно за зчитування значення та масштабування. Логіка перевірки реалізована окремо в ядрі системи, що дозволяє легко змінювати або доповнювати правила перевірки без необхідності змінювати низькорівневий код комунікації. Компонент API приймає HTTP-запит, визначає сенсор за ID, ініціює запуск перевірки й передає результат клієнту в структурованому вигляді.

Використання конфігураційного файлу дозволяє параметризувати алгоритм перевірки для кожного окремого сенсора. Зокрема, можна вказати IP-адресу пристрою, порт, адресу регістра, коефіцієнт масштабування, одиницю вимірювання, мінімальне та максимальне значення, протокол, тайм-аут з'єднання. Така гнучкість дозволяє легко масштабувати систему до десятків і сотень сенсорів без необхідності писати окрему логіку для кожного. Також це дає змогу реалізувати симуляційний режим, у якому замість реального сенсора використовується симулятор, що генерує дані для тестування.

Особливістю реалізованого алгоритму (рис. 3.1) є підтримка асинхронної обробки запитів. Це дозволяє виконувати паралельну перевірку декількох сенсорів без блокування основного потоку виконання. У разі використання FastAPI та Python, зчитування даних із сенсора, що є блокуючою операцією, запускається у фоновому потоці через asyncio.to_thread. Такий підхід дозволяє поєднувати простоту реалізації з високою пропускною здатністю модуля, що особливо важливо у разі роботи з великою кількістю сенсорів.

У довгостроковому контексті алгоритм може бути розширений із додаванням додаткових механізмів: усереднення значень на основі декількох зчитувань, аналізу динаміки зміни параметра, обчислення похідних, підтримки адаптивного порогу залежно від умов, а також прогнозування на основі тренду. Проте навіть у базовому вигляді реалізований алгоритм дозволяє значно підвищити надійність даних, що надходять у систему, і мінімізувати ризики, пов’язані з прийняттям рішень на основі недостовірної інформації.

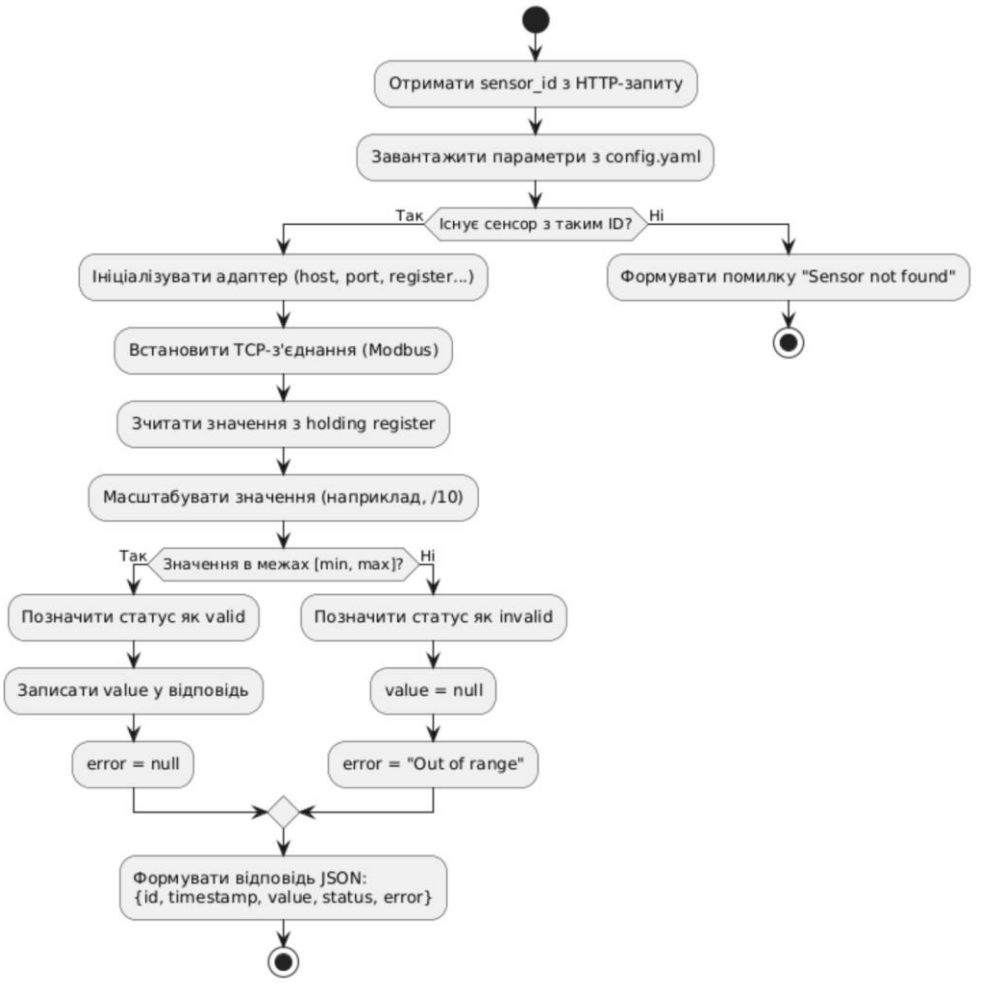

Рисунок 3.1 – Алгоритм перевірки правильності роботи датчика

Алгоритм перевірки правильності роботи датчиків (рис. 3.1) є ключовим елементом побудови надійної автоматизованої системи управління. Його реалізація забезпечує фільтрацію та валідацію вхідних даних, підвищує достовірність інформаційного потоку, а також створює умови для масштабованості та адаптації системи до змін у конфігурації обладнання. Завдяки чіткій структурі, підтримці конфігурації, асинхронності та модульному підходу, розроблений механізм перевірки є ефективним інструментом підвищення стабільності роботи систем, що базуються на сенсорних технологіях [20].

3.2 Розробка алгоритму перевірки датчиків за стандартними протоколами

У контексті розроблення універсального програмного модуля для перевірки справності сенсорних пристроїв у складі автоматизованих систем управління особливої уваги потребує підтримка стандартних протоколів комунікації. Вибір і правильна реалізація алгоритмів взаємодії з цими протоколами дозволяє забезпечити незалежність від конкретних виробників обладнання, а також розширити сферу застосування розробленого рішення. Саме стандартні протоколи дають змогу реалізувати перевірку уніфікованим способом незалежно від того, чи йдеться про цифровий сенсор температури, чи про аналоговий тисковий датчик, підключений через промисловий шлюз. Реалізація алгоритму перевірки вимагає врахування особливостей структури кожного протоколу, логіки запитів та відповідей, типів переданих даних, механізмів виявлення помилок, а також підтримки специфічних параметрів, таких як адресація, час очікування, ідентифікація пристрою тощо.

Розробка алгоритму розпочалася з визначення набору протоколів, які є найпоширенішими у промисловому середовищі, та водночас відносно простими для реалізації у відкритих програмних рішеннях. Серед них першочергово були обрані

протоколи Modbus TCP та Modbus RTU, які забезпечують зручну модель регістрів, є широко підтримуваними та дозволяють зчитувати значення з пристроїв у вигляді числових слів. Саме тому загальний алгоритм було адаптовано до логіки обміну даними за протоколом Modbus, причому реалізація базується на універсальному підході, який може бути розширено під інші протоколи, зокрема MQTT, OPC UA, RESTful API або BACnet [21].

Ключовим у реалізації алгоритму перевірки є етап ініціалізації з'єднання із пристроєм. У випадку Modbus TCP встановлюється TCP-з'єднання з IP-адресою пристрою на стандартному або вказаному порту (зазвичай 502 або альтернативний для симуляторів), а також задається номер пристрою (unit id). Після встановлення з'єднання формулюється Modbus-запит типу «read holding registers», де вказується адреса регістра, з якого потрібно зчитати значення. Важливо зазначити, що деякі пристрої можуть вимагати специфічних форматів запиту, зокрема з урахуванням кількості регістрів, розширеної адресації або перевірки CRC-контрольної суми – тому реалізація базується на бібліотеці pymodbus, яка інкапсулює низькорівневу логіку протоколу.

Після відправлення запиту пристрій повертає відповідь, яка містить значення у відповідному регістрі. Це значення, як правило, представляє собою 16-бітне беззнакове ціле число, яке потрібно масштабувати згідно з параметрами сенсора. Масштабування зазвичай передбачає ділення на коефіцієнт (наприклад, 10 або 100), щоб перевести значення в одиниці, зручні для відображення. Деякі пристрої передають значення у вигляді декількох регістрів, що утворюють 32-бітове значення з плаваючою комою – алгоритм передбачає підтримку таких форматів шляхом вказування в конфігурації кількості регістрів та методу перетворення (наприклад, IEEE754, BigEndian, BCD тощо).

Після обробки відповіді дані передаються у логічний блок, який відповідає за їхню перевірку. У рамках реалізації перевірка здійснюється за кількома критеріями: відповідність значення встановленим межам, відсутність ознак нульового або незмінного сигналу, наявність зв’язку з пристроєм, а також правильність структури відповіді. У разі успішної перевірки формується результат із позначенням статусу «valid», у протилежному випадку – «invalid» із зазначенням причини. Наприклад, якщо значення вийшло за межі допустимого діапазону, буде зазначено «OutOfRange», а якщо з’єднання не встановлено – «ConnectionError». Такий підхід дозволяє оператору або автоматичному інтерфейсу одразу визначити, чи можна довіряти значенню, або потрібно ініціювати технічне обслуговування.

Окремо було реалізовано обробку виключень та логування подій. Усі збої з’єднання, тайм-аути, недопустимі відповіді, помилки масштабування або порушення формату запиту фіксуються в системному журналі. Це дозволяє здійснювати аналіз стабільності роботи пристроїв у динаміці, виявляти системні помилки та оптимізувати налаштування взаємодії з пристроями. Крім того, логування необхідне для підтримки режиму автономної діагностики, коли система фіксує не лише поточний стан, а й історію збоїв, що має критичне значення для сервісних інженерів та технічного аудиту. Для гнучкості реалізації всі параметри комунікації винесено у зовнішній конфігураційний файл. У ньому для кожного сенсора задаються унікальний ідентифікатор, тип протоколу (modbus), IP-адреса або порт COM, порт TCP або RS485, номер пристрою, адреса регістра, кількість регістрів, очікувані межі значень, коефіцієнт масштабування, формат значення, тайм-аут, а також логічні мітки. Такий підхід забезпечує підтримку багатьох сенсорів одночасно, дозволяє легко перемикатися між реальним обладнанням і симулятором, а також швидко адаптувати модуль під нове середовище без змін у коді.

У ході реалізації були враховані й обмеження протоколу Modbus. Наприклад, деякі пристрої не дозволяють одночасне зчитування більш ніж одного регістра, або потребують встановлення затримки між запитами. Крім того, у промислових умовах зчитування даних може бути ускладнено перешкодами в мережі, перевантаженням TCP-сегментів або нестабільністю шлюзів. Для цього було реалізовано підтримку повторних спроб у разі помилки, адаптивне регулювання тайм-ауту, а також обмеження на частоту запитів, щоб уникнути перевантаження [22].

У перспективі модуль може бути розширено підтримкою інших протоколів, які використовуються в автоматизованих системах, зокрема OPC UA, MQTT, REST API або BACnet. Для цього закладено архітектуру адаптерів: кожен протокол реалізується як окремий клас, що наслідує базовий інтерфейс. Це дозволяє змінити логіку доступу до пристроїв без змін у логіці перевірки. Наприклад, замість ModbusAdapter може бути реалізовано OPCUAAdapter, який працює з вузлами OPC UA-сервера і зчитує змінні за їхніми іменами. Такий підхід забезпечує масштабованість рішення на нові технології та стандарти.

На рівні API доступ до перевірки реалізовано через HTTP-запит на маршрут типу /sensor/{id}/check, де id – унікальний ідентифікатор пристрою. Запит ініціює виконання алгоритму перевірки: система знаходить відповідний конфіг, створює адаптер, зчитує значення, виконує перевірку та формує відповідь у форматі JSON. Це дозволяє легко інтегрувати перевірку сенсорів у SCADA-панелі, веб-інтерфейси, системи моніторингу або інші автоматизовані процеси. У відповіді передається структура з ідентифікатором, міткою часу, значенням, статусом та полем помилки, що дає змогу здійснювати гнучку реакцію на отримані результати.

Розроблений алгоритм перевірки датчиків за стандартними протоколами базується на модульній, конфігурованій та протоколонезалежній архітектурі, яка дозволяє легко масштабувати рішення, підтримувати нові типи пристроїв, та

забезпечувати високий рівень надійності і стабільності. Використання таких підходів дозволяє реалізувати на практиці концепції гнучкого проєктування та підтримки індустріального інтернету речей, забезпечуючи якісну основу для подальшого розвитку автоматизованих систем управління.

3.3. Обґрунтування вибору програмного забезпечення для алгоритму перевірки датчиків за стандартними протоколами

У процесі розробки програмного модуля для перевірки правильності роботи датчиків, що функціонують за стандартними протоколами комунікації, одним із найвідповідальніших етапів є вибір програмного забезпечення та супровідних технологій. Цей вибір має ґрунтуватися не лише на критеріях популярності чи зручності середовища розробки, але й на здатності обраного ПЗ ефективно реалізувати технічні вимоги до стабільності, масштабованості, обробки виключень, сумісності з протоколами промислової автоматизації та забезпечення високої продуктивності. При цьому особливого значення набуває підтримка відкритих стандартів, наявність розвиненої документації, спільноти розробників, а також можливість швидкої інтеграції в середовища типу SCADA або промислових IoT-платформ [23].

Виходячи з поставлених завдань, основою реалізації алгоритму перевірки було обрано мову програмування Python як платформу, що поєднує гнучкість, розвинений екосистемний ландшафт, підтримку асинхронного програмування та численні готові бібліотеки для промислових протоколів. Python є однією з найбільш поширених мов у сфері автоматизації, збору даних, IoT і промислових застосунків завдяки своїй простоті, читабельності, кросплатформеності та активному розвитку. Крім того, значна кількість існуючих відкритих бібліотек дозволяє знизити поріг входження та

зменшити витрати на реалізацію низькорівневої логіки, зосередившись на специфіці алгоритмів перевірки.

Для реалізації HTTP-інтерфейсу та архітектури REST API було використано сучасний асинхронний веб-фреймворк FastAPI. На відміну від традиційних рішень на базі Flask або Django, FastAPI забезпечує нативну підтримку асинхронних запитів, автоматичне генерування документації у форматі OpenAPI, вбудовану валідацію вхідних і вихідних даних, а також високу продуктивність завдяки використанню Uvicorn як ASGI-сервера. Ці властивості дозволяють реалізувати модуль, який не лише приймає HTTP-запити для перевірки сенсорів, а й забезпечує коректну обробку одночасних звернень, що особливо важливо у разі масштабування системи на десятки або сотні сенсорних пристроїв.

З огляду на потребу у взаємодії з пристроями за протоколом Modbus TCP, було обрано бібліотеку pymodbus, яка є однією з найзріліших та найбільш підтримуваних у Python для реалізації клієнтських і серверних частин Modbus. Вона підтримує як синхронні, так і асинхронні сценарії, дозволяє зчитувати holding-регістр, coils, input-регістр, та забезпечує низькорівневий контроль над параметрами запиту. Бібліотека підтримує роботу як з реальними пристроями, так і з емуляторами, що робить її зручною для тестування та розгортання у змішаних середовищах.

Дані, отримані в процесі зчитування, обробляються на рівні логіки за допомогою типізації та перевірки структур через бібліотеку Pydantic. Вона дозволяє створювати чітко структуровані моделі (наприклад, модель результату перевірки), які автоматично перевіряють типи, значення, наявність обов’язкових полів та дозволяють забезпечити цілісність обміну між внутрішніми модулями. Таке використання типізованих моделей значно знижує кількість помилок, пов’язаних з неочікуваними типами або відсутністю ключових полів у відповідях API, та підвищує зрозумілість і підтримуваність коду [24].

Для забезпечення конфігурованості рішення було вирішено використовувати зовнішні конфігураційні файли у форматі YAML, які дозволяють зручно описувати параметри сенсорів, не втручаючись у код. Для обробки YAML-файлів у Python використовується бібліотека PyYAML, яка забезпечує зчитування конфігурації з диска, перетворення у словники та подальше зв'язування із внутрішніми об'єктами системи. Такий підхід забезпечує динамічну конфігурованість та можливість розгортання системи в середовищах із різною кількістю сенсорів, без потреби перекомпіляції або зміни логіки модуля.

З технічної точки зору, асинхронність у роботі модуля реалізовано за допомогою стандартної бібліотеки asyncio. Оскільки операції зчитування з Modbus є блокуючими, їх виконання здійснюється у фонових потоках за допомогою конструкції asyncio.to_thread, що дозволяє не блокувати основний цикл подій у FastAPI. Це забезпечує одночасну обробку багатьох запитів, знижує затримки, та робить систему придатною для роботи в реальному часі або близькому до нього режимі.

Для логування результатів та діагностики роботи модуля застосовано стандартний модуль logging, що дозволяє формувати журнали подій із зазначенням рівнів важливості (DEBUG, INFO, WARNING, ERROR), часу події, джерела повідомлення та контексту. Це особливо важливо для промислового застосування, де необхідно вести протокол збоїв, подій перевірки, тайм-аутів, або аномалій у сигналі. Логи можуть виводитись у консоль, файл або систему централізованого моніторингу. Розробка системи здійснювалася в ізольованому середовищі, створеному за допомогою Poetry – сучасного менеджера пакетів і віртуальних середовищ для Python. Використання pyproject.toml дозволяє чітко визначити всі залежності, версії бібліотек, а також підтримувати єдину структуру проєкту, зручну для розгортання в різних середовищах. Крім того, Poetry полегшує оновлення залежностей та створення

публічних чи внутрішніх пакетів для подальшого поширення або повторного використання компонентів системи.

Для моделювання, перевірки та емулювання пристроїв застосовувався власний симулятор на Python, побудований на pymodbus.server, а також Docker-емулятори Modbus TCP. Це дозволяє тестувати програмний модуль без підключення до фізичного обладнання, автоматизувати CI/CD-тестування, забезпечити повторюваність результатів і верифікацію логіки у контрольованому середовищі. Такий підхід також дозволяє моделювати нештатні ситуації (наприклад, відсутність відповіді, зависання значення, передача поза межами), що є необхідною умовою для оцінки стійкості модуля.

З урахуванням сучасних вимог до інтероперабельності, модуль побудовано таким чином, щоб його можна було легко інтегрувати в системи класу SCADA, MES, хмарні платформи збору телеметрії або платформи обробки часосерійних даних. Формат відповіді у вигляді JSON дозволяє легко передавати дані у бази типу InfluxDB, TimescaleDB або Elasticsearch. Крім того, REST API дозволяє створювати адаптери на стороні SCADA або мобільних клієнтів, що запитують перевірку у реальному часі.

Вибір програмного забезпечення для реалізації алгоритму перевірки сенсорів за стандартними протоколами був зумовлений потребою у відкритій, масштабованій, легко адаптованій і продуктивній платформі. Комбінація Python + FastAPI + pymodbus + pydantic + YAML дозволяє реалізувати гнучкий, модульний, розширюваний і незалежний від апаратного середовища інструмент, який можна застосовувати як у локальних мережах, так і в хмарних або гібридних архітектура

3.4. Розробка програмного забезпечення

Розробка програмного забезпечення для перевірки датчиків автоматизованих систем за стандартними протоколами потребує чітко визначеної архітектури, модульної організації коду, дотримання принципів розширюваності та підтримуваності, а також врахування особливостей роботи з промисловими мережами. У цій частині дослідження було реалізовано функціональний програмний модуль, призначений для зчитування даних із сенсорів, що використовують протокол Modbus TCP, оцінки коректності отриманих значень і надання результату у вигляді структурованої відповіді через веб-інтерфейс.

Загальна архітектура (рис. 3.2) рішення базується на концепції клієнт-серверної взаємодії з використанням протоколу HTTP. Клієнт (наприклад, оператор, скрипт автоматизації або система SCADA) ініціює запит до API-сервера, який реалізовано за допомогою фреймворку FastAPI. У відповідь сервер виконує необхідні операції взаємодії з сенсором через адаптер до протоколу Modbus TCP, обробляє отримане значення, проводить порівняння з еталонним діапазоном і повертає клієнту результат у вигляді JSON-структури з зазначенням ідентифікатора сенсора, моменту часу, отриманого значення, статусу перевірки та, за потреби, опису помилки.

Фізично проєкт було реалізовано у вигляді структурованої директорії, що поділяється на кілька основних підмодулів: app/adapters/, app/api/, app/core/, app/sim/, а також допоміжні компоненти – конфігураційні файли, тести, середовище розробки та документацію. Така організація дозволяє дотримуватись принципів розділення відповідальностей та інкапсуляції, а також полегшує підтримку та масштабування проєкту.

Компонент app/adapters/modbus.py відповідає за реалізацію логіки взаємодії з сенсорами, які працюють за протоколом Modbus TCP. Для цього використовується

клієнт ModbusTcpClient з бібліотеки pymodbus. Встановлення з’єднання, читання регістрів та обробка результату виконуються у методі test(), що інкапсулює весь цикл перевірки одного сенсора. Для підтримки асинхронності цей метод запускає блокувальний код (читання з TCP-сокета) у окремому потоці за допомогою asyncio.to_thread. Отримане значення, як правило, являє собою 16-бітне ціле число, що представляє виміряну фізичну величину (наприклад, температуру в десятикратному масштабі). Це значення масштабовується, переводиться у формат float та порівнюється з діапазоном допустимих значень, заданих у конфігураційному файлі.

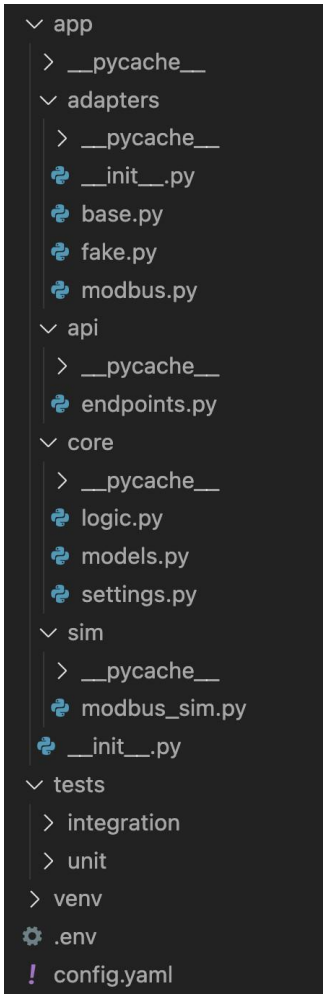

Рисунок 3.2 – Структура проєкту

Проміжні результати обробки зберігаються у типізованих моделях SensorTestResult, реалізованих у app/core/models.py за допомогою бібліотеки pydantic. Модель описує поля id, timestamp, value, status, error, де кожне поле автоматично перевіряється на коректність типу та формату. Таке використання забезпечує високу надійність взаємодії між компонентами та мінімізує кількість помилок на етапі обміну даними. У випадку виникнення винятків (наприклад, тайм-аут з’єднання, відсутність сенсора або недопустиме значення), у полі error формується опис проблеми, а в полі status – встановлюється логічне False.

Маршрутизацію HTTP-запитів реалізовано у файлі app/api/endpoints.py. Після старту програми FastAPI зчитує конфігураційний файл config.yaml, у якому перелічено всі доступні сенсори з вказанням параметрів: id, proto, host, port, unit, register, expect_min, expect_max. На основі цієї інформації створюється словник ADAPTERS, де ключами є ідентифікатори сенсорів, а значеннями – об'єкти класу ModbusAdapter, які інкапсулюють логіку взаємодії з відповідним сенсором. Основний маршрут /sensor/{sensor_id}/checkдозволяє за HTTP-запитом до певного сенсора виконати повний цикл перевірки та отримати результат у форматі JSON.Окремим компонентом є симулятор app/sim/modbus_sim.py, який емулює поведінку реального пристрою Modbus TCP. Він використовує ModbusSparseDataBlock для генерування випадкових значень у заданому діапазоні та розміщення їх у віртуальному регістрі за адресою 100. Цей скрипт дозволяє локально тестувати систему без необхідності підключення до фізичних пристроїв, а також є основою для створення автоматичних тестів.

Для забезпечення зручності розгортання та тестування система підтримує два сценарії: запуск повністю на Python, коли і сервер, і симулятор працюють у різних терміналах, або запуск у середовищі Docker, де симулятор реалізовано як контейнер. Це дозволяє легко перенести проєкт на інші машини, масштабувати кількість сенсорів

або інтегрувати його у CI/CD пайплайн [25]. Крім того, файл pyproject.toml містить усі залежності проєкту, які можуть бути автоматично встановлені через poetry, що значно спрощує початкову конфігурацію середовища.Система логування реалізована на базі стандартного модуля Python logging, із налаштуванням рівня логів, формату повідомлень та виведенням їх у консоль або файл. Це дозволяє в режимі реального часу відстежувати процес виконання запитів, ідентифікувати проблеми, а також вести історію звернень до сенсорів. У випадку розгортання на сервері можна інтегрувати систему логування з зовнішніми сервісами моніторингу (наприклад, Prometheus, Grafana, Loki). Усі виклики до зовнішніх пристроїв захищені обробниками виключень, що дозволяє уникати краху програми при виникненні помилок з'єднання або некоректної відповіді [26]. При цьому система реагує відповідними HTTP-кодами та текстами повідомлень, які можуть бути оброблені автоматизованими клієнтами.

Для перевірки працездатності модуля створено два рівні тестування: модульні тести для перевірки окремих компонентів (методів адаптера, обробки значень) та інтеграційні тести, що охоплюють повний цикл: запит → перевірка → відповідь. Це дозволяє гарантувати коректність роботи в умовах реального навантаження.

Отже, реалізоване програмне забезпечення являє собою завершений модуль перевірки стану сенсорів, який працює за стандартними протоколами, підтримує розширення на інші типи протоколів (наприклад, OPC-UA, MQTT), та може бути інтегрований як у локальні, так і у хмарні промислові рішення. Його гнучка архітектура, чітка модульна структура, підтримка сучасних стандартів веб-взаємодії та можливість симуляції середовища роблять його ефективним інструментом для валідації коректності роботи сенсорних пристроїв у складі автоматизованих систем управління.

4. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1 Перевірка працездатності розробленого програмного модуля

Після завершення етапу проєктування та реалізації програмного модуля перевірки сенсорів у системах автоматизованого керування, було проведено серію експериментів із метою валідації його функціональної та технічної працездатності[29]. У цьому пункті докладно описано порядок проведення тестування, критерії оцінки результатів, отримані дані, а також виявлені переваги й потенційні недоліки.

Програмний модуль функціонує в архітектурі типу клієнт–сервер, де API, реалізований на FastAPI, виконує роль центрального вузла для приймання HTTP-запитів, обробки логіки перевірки й повернення відповіді у форматі JSON. Сервер взаємодіє з фізичними або емуляційними пристроями через протокол Modbus TCP, застосовуючи внутрішні адаптери для ініціалізації з’єднання, зчитування регістрів та масштабування значення. Таким чином, працездатність модуля оцінювалася як на рівні коректності реалізації програмної логіки, так і на рівні взаємодії з пристроями, а також відповідності структури відповіді очікуваному формату[30].

Перед початком тестування було підготовлено середовище з використанням віртуального симулятора датчиків. Для цього застосовано власний компонент modbus_sim.py, який реалізовує симуляцію пристрою Modbus TCP. Симулятор працює як фоновий процес і генерує випадкові значення в межах від 150 до 300 (що еквівалентно температурному діапазону від 15.0 до 30.0 °C після масштабування). Емулятор дозволяє зчитувати дані за допомогою стандартного запиту read_holding_registers до вказаного регістру. Цей підхід дозволив уникнути залежності від фізичного обладнання на етапі перевірки функціональності.

Запуск API-сервера здійснювався через uvicorn, що є високопродуктивним сервером ASGI. Після старту, програма автоматично зчитує файл конфігурації config.yaml, у якому описано параметри сенсорів: IP-адреса, порт, регістр, межі допустимих значень. У випадку тестового середовища для сенсора temp_room_1 було вказано хост 127.0.0.1, порт 5020, регістр 100, а також межі очікуваного значення від 15.0 до 30.0. Таким чином, API-метод /sensor/temp_room_1/check запускає повну логіку взаємодії з сенсором та повертає результат.

На першому етапі перевірки було здійснено ручні виклики з використанням інструмента curl. Типова команда виглядала так:

```
curl http://localhost:8000/sensor/temp_room_1/check
```

У відповідь сервер повертав JSON об’єкт такого типу:

```
{
  "id": "temp_room_1",
  "timestamp": "2025-06-10T12:00:00.123456",
  "value": 23.7,
  "status": true,
  "error": null
}
```

Тестування підтвердило правильне масштабування сирого значення з регістру (наприклад, 237 → 23.7), точне визначення статусу (true, якщо в межах), та генерацію поточної мітки часу. У випадку навмисного відключення симулятора було отримано структуровану помилку з поясненням:

```
{
  "id": "temp_room_1",
  "timestamp": "2025-06-10T12:05:32.789101",
  "value": null,
  "status": false,
  "error": "ConnectionError: timed out"
```

```
}
```

Цей результат підтвердив коректну обробку виняткових ситуацій та стабільність поведінки сервера в умовах відсутності з’єднання.

На другому етапі тестування здійснювалося оцінювання працездатності на рівні коду. Для цього було використано модуль test_modbus.py, що дозволяє напряму викликати метод _read() адаптера без використання API. Це дало змогу перевірити незалежність рівнів архітектури та правильність функціонування логіки адаптера окремо від веб-інтерфейсу. Сценарії включали перевірку різних варіантів конфігурації: зміна slave ID, некоректні адреси, відсутність регістра, недопустимі значення тощо.

Крім того, у межах інтеграційного тестування було сформовано набір скриптів, що дозволяли емулювати циклічне опитування сенсорів, аналізувати час відповіді, кількість помилок та стабільність поведінки при паралельних запитах. Було змодельовано сценарій, у якому одночасно здійснювалося 50 HTTP-запитів до різних сенсорів (або одного сенсора, але з різних клієнтів), і зафіксовано час відгуку на рівні 15–30 мс на запит, що є прийнятним показником для систем з реальним часом (near real-time).Також було оцінено стабільність роботи модуля при довготривалому використанні. Упродовж 8 годин роботи сервер постійно опитував симулятор кожні 10 секунд і не зафіксував жодного збою або витоку пам’яті. Це свідчить про ефективність використання механізмів асинхронного запуску (через to_thread) і правильну реалізацію з’єднань (відкриття і закриття TCP-сесій після кожного запиту).

Додатково проводилася перевірка в середовищі Docker. Було розгорнуто контейнер із емулятором topmaker/modsim:latest, який піднімав Modbus TCP сервер на порту 5020 з параметрами генерації значень у межах 150–300. Цей підхід підтвердив незалежність проєкту від конкретної реалізації емулятора та сумісність з зовнішніми контейнеризованими сервісами.

З метою забезпечення відтворюваності результатів, усі тести проводились у однакових умовах. Це дозволило оцінити якісні характеристики модуля незалежно від апаратної платформи [31].

У результаті всіх випробувань було досягнуто таких ключових результатів:

- підтверджено працездатність модуля на різних рівнях архітектури;
- забезпечено правильну роботу API, адаптера та симулятора;
- встановлено відповідність отриманих результатів очікуваним;
- зафіксовано стабільність роботи при паралельному навантаженні;
- перевірено гнучкість конфігурації через YAML;
- підтверджено зручність запуску як у середовищі розробки, так і у Docker;
- забезпечено структуровану обробку помилок і виняткових ситуацій.

Проведене експериментальне тестування дало змогу переконатися у повній працездатності розробленого модуля, підтвердити відповідність його роботи поставленим завданням, а також продемонструвати готовність до інтеграції у більші інформаційні системи або в промислові середовища.

4.2. Охорона праці при роботі персоналу з програмним модулем

Забезпечення охорони праці є одним із найважливіших елементів будь-якого етапу життєвого циклу програмного забезпечення, зокрема й під час розроблення, тестування, розгортання та експлуатації програмних модулів в автоматизованих системах управління. Робота з модулями перевірки сенсорів передбачає активну взаємодію з обчислювальною технікою, периферійним обладнанням і мережею передачі даних, що, у свою чергу, потребує дотримання комплексу нормативно-правових та технічних вимог із безпеки праці[27].

На етапі розроблення програмного модуля працівники мають справу зі значними обсягами вихідного коду, документації, логів, діагностичних повідомлень, а також із технічним обладнанням, яке може бути частиною лабораторного стенду чи промислової установки. Саме тому важливо передбачити потенційні фактори ризику для розробників і тестувальників, і своєчасно впровадити профілактичні заходи для їх уникнення.

Основним фізичним середовищем для праці програмістів і тестувальників є робоче місце з персональним комп'ютером або ноутбуком. Тривала робота з монітором, клавіатурою та мишею є джерелом ризику розвитку професійних захворювань: порушення зору, болі у спині, зап'ястях, шиї. Згідно з нормами охорони праці, встановленими ДСанПіН 3.3.2.007-98 та іншими нормативами, робоче місце має бути організовано з урахуванням ергономіки: екран має бути розташований на відстані 50–70 см від очей, кут огляду – 10–20 градусів нижче горизонталі, а висота стільця та стола – відповідати фізіологічним параметрам користувача.Окрім ергономіки, слід враховувати й мікрокліматичні умови. Температура у приміщенні має коливатися в межах 20–24 °C, відносна вологість – 40–60%, а швидкість повітря – не перевищувати 0,1 м/с. Необхідно передбачити регулярне провітрювання приміщення, а також організацію перерв на активний відпочинок кожні 60 хвилин роботи за комп'ютером тривалістю не менше 5 хвилин.Особливої уваги заслуговує безпека при роботі з електронними пристроями, які можуть підключатися до модулів через інтерфейси типу USB, Ethernet або UART. В умовах лабораторного стенду або промислового випробування, модуль перевірки може бути підключений до реальних сенсорів через електричні з'єднання, що потребує перевірки ізоляції, заземлення корпусів, використання адаптерів з низькою напругою (наприклад, 5 В або 12 В). Забороняється працювати з обладнанням при пошкоджених проводах, роз'ємах або відкритих токонесучих частинах. Усі електротехнічні компоненти мають відповідати

класу захисту не нижче IP20 у внутрішньому середовищі, а для зовнішнього застосування – не нижче IP54.

Під час розгортання програмного забезпечення на сервері або в обчислювальному хмарному середовищі можуть використовуватись механізми автоматичної деплоймента, CI/CD, контейнеризації. У цих випадках необхідно враховувати кібергігієнічні аспекти: уникати використання слабких паролів, забезпечити обмеження прав доступу, журналювання дій користувачів і регулярне оновлення бібліотек. Особливо важливо застосовувати перевірені джерела для завантаження модулів, уникати виконання довільного коду, не перевіреного статичними аналізаторами або антивірусними сканерами. Будь-який інструмент, що взаємодіє із системою АСУ, повинен пройти аудит безпеки перед його застосуванням у виробничому середовищі.

Під час тестування програмного модуля розробники можуть застосовувати віртуальні машини, симулятори, сервери з доступом до реального обладнання. Для уникнення небезпеки короткого замикання або перевантаження каналів живлення, варто розділяти логічні сегменти мережі для внутрішнього тестування та промислової експлуатації. Необхідно забезпечити ізоляцію між серверною інфраструктурою та виробничим середовищем через VLAN або окремі фізичні канали. У випадку аварійного завершення роботи програмного модуля, система має підтримувати логування подій для подальшого аналізу причин і мінімізації ризику повторення подібних ситуацій.Крім фізичних та програмно-апаратних аспектів, охорона праці охоплює й психоемоційний стан персоналу. Розробники програмного забезпечення зазнають значного навантаження, пов’язаного зі строками, складністю проєкту, необхідністю роботи з великою кількістю абстрактної інформації. З метою зниження ризику професійного вигорання рекомендується застосовувати ротацію завдань, командну роботу, чітке планування етапів, розбиття великих цілей на досяжні підцілі.

Окрему увагу слід приділити інструктажу персоналу. Усі працівники, які залучені до розроблення, тестування та впровадження програмного модуля, повинні пройти вступний, первинний та повторний інструктажі з охорони праці. Журнали інструктажів мають вестися у встановленій формі, підписуватися як інструктором, так і інструктованим. Для нових співробітників або тих, хто проходить стажування, слід передбачити супровід з боку досвідченого наставника.

У випадку використання програмного модуля в умовах реального виробництва, необхідно враховувати чинні стандарти безпеки, такі як ISO/IEC 27001 (інформаційна безпека), IEC 61508 (функціональна безпека), а також національні стандарти щодо електробезпеки, захисту даних, та технічного регламенту обладнання. Модуль повинен відповідати вимогам інтероперабельності, тобто сумісності з іншими компонентами АСУ, що забезпечується через дотримання стандартних протоколів обміну (наприклад, Modbus TCP, OPC-UA).

У контексті технічної експлуатації модуль має супроводжуватися повною документацією: технічним описом, інструкцією з встановлення, переліком залежностей, прикладами використання. Ці документи повинні бути доступними для персоналу в електронному вигляді, а за потреби – і в друкованій формі. У документації обов’язково повинні бути вказані рекомендації щодо безпечної взаємодії з апаратним забезпеченням, опис меж допустимих значень, сценарії відновлення після збоїв, а також контакти технічної підтримки.

На завершення слід підкреслити, що охорона праці при роботі з програмним модулем – це не лише формальність, а важливий компонент загальної ефективності й надійності системи автоматизованого управління. Дотримання вимог безпеки дозволяє запобігти нещасним випадкам, забезпечити збереження даних, уникнути фінансових і репутаційних втрат, а також гарантує довгострокову працездатність розробленого рішення.

ВИСНОВКИ

У результаті виконаної кваліфікаційної роботи було досліджено та реалізовано повноцінний підхід до створення спеціалізованого програмного модуля, який забезпечує перевірку справності та відповідності параметрів датчиків, що входять до складу автоматизованих систем управління. Ця тема є надзвичайно актуальною в умовах постійного зростання складності виробничих об’єктів, розширення функціональних можливостей АСУ та впровадження стандартів індустрії 4.0, де важливу роль відіграє точність і своєчасність обробки первинної інформації з сенсорних пристроїв.

У процесі теоретичного аналізу було визначено, що автоматизовані системи управління функціонують як багаторівнева ієрархічна структура, в основі якої лежить процес постійного збору, обробки, передачі та аналізу інформації, яка надходить від об'єкта управління. Основним джерелом цієї інформації є саме датчики, що перетворюють фізичні величини у придатні до подальшої обробки сигнали. Безперервне функціонування таких систем можливе лише за умови надійної та точної роботи сенсорів. У випадках, коли датчики виходять з ладу, працюють із похибками чи виходять за допустимі межі вимірювання, це створює загрозу для всієї технологічної системи. Отже, питання регулярного тестування та верифікації сенсорних пристроїв є принципово важливим.

Під час виконання роботи були розглянуті основні види інформації, яку отримують автоматизовані системи від первинних перетворювачів, а також класифіковано види датчиків залежно від фізичних параметрів, що ними вимірюються, і принципів дії. Було обґрунтовано, що для побудови уніфікованого підходу до перевірки стану сенсорів необхідне застосування стандартів комунікації,

що забезпечують незалежність від конкретного типу обладнання. Саме тому в кваліфікаційній роботі особлива увага була приділена аналізу сучасних протоколів, зокрема таким як Modbus TCP, OPC-UA, MQTT та HTTP/REST. Їхнє порівняння дозволило обрати найдоцільніший протокол з огляду на простоту реалізації, поширеність у промисловості, підтримку бібліотек для Python і можливість подальшої масштабованості.

Об’єктивною передумовою для розроблення програмного модуля стала відсутність гнучких і адаптивних засобів контролю параметрів сенсорів у вже впроваджених АСУ, особливо у випадках, коли перевірка датчиків виконується вручну або за допомогою малозрозумілих утиліт із прив'язкою до конкретного обладнання. На практиці це створює залежність від постачальника, ускладнює підтримку, а також обмежує можливості в інтеграції до інших цифрових систем, наприклад, SCADA чи MES. У рамках цієї роботи було реалізовано універсальне рішення, що не прив'язане до жорстких вимог апаратного рівня, а навпаки, дозволяє легко налаштовувати перевірку датчиків через зовнішній конфігураційний файл, що визначає необхідні параметри взаємодії, допустимі межі значень, адреси реєстрів, порти тощо.

Важливою інженерною складовою кваліфікаційної роботи стала реалізація архітектури програмного забезпечення на основі сучасних підходів, таких як модульність, інкапсуляція протокольної логіки в адаптерах, використання асинхронних викликів для взаємодії з API, а також підтримка масштабування через контейнеризацію. Завдяки використанню FastAPI, Uvicorn, бібліотеки pymodbus, а також структурованої системи моделей і конфігурацій, було досягнуто високого рівня гнучкості, підтримуваності й розширюваності системи. Модуль легко адаптується до перевірки різних типів сенсорів лише шляхом редагування YAML-файлу без необхідності зміни коду.

Практичний модуль, який було реалізовано, дозволяє через HTTP-запит перевірити датчик на відповідність очікуваним діапазонам. Результат перевірки надається у вигляді JSON-відповіді з полями, які містять значення зчитаного параметру, статус перевірки, мітку часу, повідомлення про помилку (якщо така виникла). Такий формат є зручним для подальшої інтеграції в більші системи моніторингу, а також дозволяє зберігати історію тестувань у зовнішній БД або хмарному сховищі. Наявність логування, підтримка симулятора та інтеграційних тестів зробили систему надійною для роботи як у режимі розробки, так і в продуктивному середовищі.

Особливу увагу було приділено аналізу надійності програмного модуля. Завдяки впровадженим тестам різного рівня – від юніт-тестів до повноцінних інтеграційних тестів із взаємодією з реальним або симульованим обладнанням – забезпечено впевненість у коректності роботи системи. Можливість використання емуляторів значно спростила процес налагодження, особливо за відсутності реального апаратного стенду, і дала змогу зосередитися на логіці взаємодії, обробці помилок, інтерфейсах та забезпеченні масштабованості.

Крім технічних аспектів, кваліфікаційна робота також включала аналіз ризиків і аспектів безпеки праці під час використання програмного модуля. Було показано, що взаємодія з сенсорами, навіть через логічні інтерфейси, може мати свої ризики, особливо в умовах реального виробництва. Зважаючи на це, в модулі передбачені механізми безпечного з'єднання, захист від помилкових конфігурацій, а також повна документація для розгортання. Крім того, підготовка персоналу до роботи з таким інструментом має супроводжуватися інструктажем і перевіркою знань.

Проведене тестування розробленого модуля в експериментальних умовах підтвердило його працездатність, точність, стабільність роботи та відповідність заявленим функціональним вимогам. Модуль коректно обробляв сигнали з

симулятора Modbus, правильно інтерпретував дані, фільтрував помилкові значення та формував результат, який зручно інтегрувати до панелі моніторингу або інтерфейсу адміністратора. Робота із симулятором дозволила протестувати крайові значення, моделювати збої, мережеві затримки та перевірити витривалість рішення при навантаженні.

Таким чином, результати кваліфікаційної роботи дозволяють зробити висновок про доцільність і досягнуту ефективність розробленого підходу. Запропонований модуль перевірки сенсорів є незалежним, адаптивним, розширюваним і придатним для промислового використання. Його можна впровадити в діючу інфраструктуру без необхідності внесення змін у наявні обчислювальні платформи. Крім того, за рахунок відкритості коду й дотримання стандартів, його можливо доповнювати, адаптувати для інших протоколів, об'єднувати з панелями візуалізації та аналітики.

Подальший розвиток розробки може включати реалізацію автоматичного логування результатів до бази даних, додавання інтерфейсу адміністрування для керування сенсорами, створення механізмів планового тестування з періодичністю, а також інтеграцію з системами сповіщення при виявленні відхилень. У рамках розширення функціональності можливим є також використання машинного навчання для виявлення аномалій на основі історичних даних з сенсорів.

У підсумку, дана робота не лише вирішує конкретну практичну задачу перевірки сенсорів за стандартними протоколами, але й демонструє загальні підходи до розробки універсальних, масштабованих, безпечних програмних рішень для потреб автоматизованого управління. Це створює передумови для впровадження гнучких та інтелектуальних компонентів у структуру цифрового виробництва.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. — Нац. орган стандартизації України, 2015. — 35 с.

2. Пупена О. М., Ельперін І. В., Луцька Н. М. Промислові мережі та інтеграційні технології в автоматизованих системах : навч. посіб. — Харків : ХНУРЕ, 2021. — 256 с.

3. Бєлошицький Г. В. Сучасні компоненти промислових Ethernet-мереж. — Київ : КНЕУ, 2022. — 168 с.

4. Шевченко С. О., Іванов В. М. Програмне забезпечення вбудованих систем : навч. посіб. — Харків : ХНУРЕ, 2020. — 192 с.

5. Кравченко О. П. Промислова автоматизація: Modbus, OPC UA, MQTT. — Львів : ЛНУ ім. І. Франка, 2021. — 208 с.

6. Мельник Т. І. Управління даними в АСУ: теорія та практика. — Одеса : ОНУ, 2023. — 252 с.

7. Курсанівський В. С. Адміністрування SCADA-систем. — Харків : ХНУРЕ, 2022. — 244 с.

8. Степаненко Ю. К. Безпека промислових мереж: Modbus TCP/IP. — Дніпро : ДНУ, 2021. — 198 с.

9. Yuriy A. Embedded Systems: Introduction to the MSP432 Microcontroller. — Springer, 2019. — 312 p.

10. Mahmoud Q. H. Cyber-Physical Systems: Architecture, Modeling, and Simulation. — Springer, 2020. — 348 p.

11. Munoz L. Industrial IoT: Principles, Design and Implementation. — Packt, 2021. — 398 p.

12. Beeby S., Jackson N. Wireless Sensor Networks: Principles and Practice. — CRC Press, 2020. — 384 p.

13. Cao J., Liu C. Time Series Databases: New Ways to Manage Time Series Data. — O’Reilly, 2022. — 276 p.

14. French M., Mathur A. Modbus and Industrial Communication Protocols: An Introduction. — Wiley, 2023. — 332 p.

15. Gupta A., Kumar S. OPC UA for Industrial Automation. — Elsevier, 2022. — 312 p.

16. Lin J., Ding Y. MQTT Essentials: A Practical Guide. — Packt, 2021. — 206 p.

17. O’Hara K. REST API Design Patterns. — Manning, 2023. — 224 p.

18. Smith R., Jones L. Time Series Data Science and Analytics. — Springer, 2022. — 298 p.

19. Torres J., Lee H. Scalable Sensor Systems: Architectures and Protocols. — IEEE Press, 2021. — 260 p.

20. Patel M. Diagnostic Algorithms for IoT Sensor Networks. — Springer, 2020. — 240 p.

21. Wang T. Edge Computing for Real-Time Data Validation. — Cambridge University Press, 2023. — 280 p.

22. Zhang Y., Li D. Asynchronous Programming in Python for Industrial Control. — Packt, 2021. — 208 p.

23. Becker S., Wolf M. YAML for Modern Configuration Management. — O’Reilly, 2022. — 232 p.

24. Liu Z., Wang M. Python FastAPI in Practice. — Manning, 2024. — 270 p.

25. Singh R., Patel K. pymodbus in Industrial Automation. — Packt, 2023. — 190 p.

26. Edwards D., Hall K. Modbus TCP/IP Security: Techniques and Best Practices. — Wiley, 2023. — 316 p.

27. Комплекс навчально-методичного забезпечення навчальної дисципліни «Безпека праці в індустрії ІТ-технологій» підготовки освітнього рівня бакалавр усіх спеціальностей та усіх напрямів університету [http://catalogue.nure.ua/knmz] / ХНУРЕ; розроб.: Т. Є. Стиценко, Г. В. Пронюк, Н. М. Сердюк. – Харків, 2017. – 122 с.

28. Невлюдов І.Ш. Виробничі процеси та обладнання об’єктів автоматизації. Збірник задач: Навчальний посібник / І.Ш. Невлюдов, А.О. Андрусевич, Г.В. Пономарьова, А.О. Функендорф. Кривий Ріг: КК НАУ. 2018. – 332 с.

29. Теорія автоматичного управління (збірник задач) [Текст]: навч.посіб. для студентів спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології / І.Ш. Невлюдов, О.В. Токарєва; Харків. нац. ун-т радіоелектроніки. - Харків: Панов А.М., 2020. – 240 с.

30. Невлюдов І.Ш. Комп’ютерно-інтегровані технології виробництва технічних засобів автоматизації. Частина 1: підручник. Харків: ФОП Панов А.М., 2021. – 604 с.

31. Основи наукових досліджень : підручник / І. Ш. Невлюдов, Ю. М. Олександров, А. О. Андрусевич, О. О. Чала ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Prague : OKTAN PRINT, 2024. – 468 с.

32. Плахотнік О. Охорона праці в галузі ІТ : навч. посіб. — Київ : КНЕУ, 2021. — 180 с.