Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №__ від ____ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

«КЛІЄНТ-СЕРВЕРНА СИСТЕМА ЗАХИЩЕНОГО ДИСТАНЦІЙНОГО МОНІТОРИНГУ ПРОМИСЛОВОГО ОБЛАДНАННЯ»

Спеціальність ***123 – Комп'ютерна інженерія***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **І. В. Барбашин**
(підпис)

Науковий керівник _____ **С. М. Рева**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, 4 розділів, висновків, списку використаних джерел і 5 додатків. Загальний обсяг роботи складає 113 сторінок, із яких 46 сторінок основної частини з 8 рисунками, 4 таблицями та 21 найменуванням у списку використаних джерел.

Метою кваліфікаційної роботи є створення клієнт-серверної системи для збору, первинної обробки, збереження та візуалізації даних від промислового обладнання з вбудованими механізмами захисту.

Об'єкт дослідження – процес дистанційного моніторингу, збору та захищеної передачі телеметричних даних промислового обладнання в умовах сучасного виробництва.

Предмет дослідження – архітектурні рішення, методи криптографічного захисту, а також програмно-апаратні засоби побудови розподіленої системи промислового моніторингу.

Проблема, яка вирішується в кваліфікаційній роботі полягає в тому, щоб безпечно та ефективно підключити вразливе промислове обладнання до сучасної IT-інфраструктури, оптимізувати великі обсяги переданої телеметрії за допомогою периферійних обчислень та гарантувати надійний криптографічний захист даних під час їх передачі через недовірені мережі.

Область застосування – виробничі підприємства, системи промислового Інтернету речей, комплекси диспетчерського контролю та системи прогнозованого технічного обслуговування обладнання, а також освітній процес за спеціальністю 123 «Комп'ютерна інженерія».

Ключові слова: інформаційні технології, комп'ютерна інженерія, клієнт-серверна система, дистанційний моніторинг, промислове обладнання, телеметрія, Modbus TCP, периферійні обчислення, криптографічний захист, база даних часових рядів, PostgreSQL, Grafana, інтернет речей.

ABSTRACT

The explanatory note for the bachelor's thesis consists of an introduction, 4 chapters, conclusions, a list of references, and 5 appendices. The total length of the thesis is 113 pages, of which 46 pages comprise the main body, including 8 figures, 4 tables, and 21 entries in the list of references.

The purpose of the thesis is to develop a client-server system for collecting, preprocessing, storing, and visualizing data from industrial equipment, incorporating built-in security mechanisms.

The object of the study is the process of remote monitoring, collection, and secure transmission of telemetry data from industrial equipment in modern production environments.

The subject of the study is architectural solutions, cryptographic protection methods, as well as software and hardware tools for building a distributed industrial monitoring system.

The problem addressed in this thesis is to securely and efficiently connect vulnerable industrial equipment to modern IT infrastructure, optimize large volumes of transmitted telemetry using edge computing, and ensure reliable cryptographic protection of data during its transmission over untrusted networks.

Scope of application: manufacturing enterprises, industrial Internet of Things systems, dispatch control complexes, and predictive equipment maintenance systems, as well as the educational process in the field of study 123 "Computer Engineering."

Keywords: information technology, computer engineering, client-server system, remote monitoring, industrial equipment, telemetry, Modbus TCP, edge computing, cryptographic protection, time-series database, PostgreSQL, Grafana, Internet of Things.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 6

ВСТУП 7

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ 9

1.1 Аналіз систем промислового моніторингу 9

1.2 Аналіз існуючих рішень дистанційного моніторингу 10

1.3 Огляд протоколів передачі даних та їх вразливості 11

1.4 Вектори атак та ризики компрометації телеметрії 13

1.5 Постановка задачі та формування вимог до системи 14

Висновок до розділу 1 15

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ 16

2.1 Загальна топологія клієнт-серверного комплексу 16

2.2 Проєктування логіки периферійного шлюзу 18

2.3 Безпека, автентифікація та захист даних 20

2.4. Проєктування логічної структури сховища даних 22

Висновок до розділу 2 25

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ 26

3.1 Вибір технологічного стеку 26

3.1.1 Основна мова програмування 26

3.1.2 Мережева взаємодія та HTTP-сервер 26

3.1.3 Форматування даних 27

3.1.4 Система керування базами даних 27

3.1.5 Криптографічний захист 27

3.2 Розробка програмного модуля шлюзу 28

3.3 Реалізація серверної частини 31
3.4 Специфікація форматів криптографічної взаємодії 32
3.5 Організація взаємодії з базою даних PostgreSQL 35
Висновок до розділу 3 36
РОЗДІЛ 4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ 37
4.1 Апаратно-програмне середовище тестового стенда 37
4.2 Розробка імітаційної моделі обладнання 37
4.2.1 Методика тестування 37
4.2.2 Аналіз продуктивності 39
4.2.3 Аналіз накладних витрат на криптографію 39
4.3 Інтеграція підсистеми візуалізації 40
4.3.1 Архітектурні принципи взаємодії Grafana з моделлю даних 40
4.3.2 Дашборд загальної доступності обладнання 41
4.2.3 Організація параметрів групи обладнання 43
4.2.4 Реалізація цифрової картки одиниці обладнання 44
Висновок до розділу 4 45
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТКИ 52

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	–	Application Programming Interface
ERP	–	Enterprise Resource Planning
HTTP	–	HyperText Transfer Protocol
IoT	–	Internet of Things
IP	–	Internet Protocol
JSON	–	JavaScript Object Notation
MES	–	Manufacturing Execution System
REST	–	Representational State Transfer
SCADA	–	Supervisory Control And Data Acquisition
SQL	–	Structured Query Language
SSD	–	Solid-State Drive
TCP	–	Transmission Control Protocol
БД	–	база даних
ДСТУ	–	державний стандарт України
КСЗІ	–	комплексна система захисту інформації
ОЗП	–	оперативний запам’ятовуючий пристрій
ОС	–	операційна система
ПЛК	–	програмований логічний контролер
СКБД	–	система керування базами даних
ЧПК	–	числове програмне керування

ВСТУП

У зв'язку з модернізацією сучасного промислового виробництва та необхідністю постійного контролю за станом технологічного обладнання, все більш актуальною постає проблема збору та безпечної передачі телеметричних даних. Головна технічна проблема полягає у величезних обсягах інформації, які генерується контролерами, та необхідності їх обробки в умовах обмеженої швидкості промислових мереж.

Актуальність роботи. Серед прикладних областей, де така проблема є вкрай актуальною, слід віднести цифровізацію підприємств критичної інфраструктури та перехід виробництв до стратегій передбачуваного технічного обслуговування. Сучасні та застарілі технологічні установки генерують тисячі параметрів щосекунди, але їх опитування здебільшого відбувається за класичним протоколом Modbus TCP, який проєктувався для ізольованих мереж і не має вбудованих механізмів автентифікації чи шифрування. Передача таких даних у відкритому вигляді створює критичні ризики: від промислового шпигунства до атак типу «людина посередині» та несанкціонованої підміни команд. Використання виключно хмарних платформ не завжди є можливим через правила цифрового суверенітету підприємств. Отже, задача розробки автономної, високопродуктивної системи з використанням периферійних обчислень та криптографічного захисту є вкрай актуальною.

Метою дослідження є створення клієнт-серверної системи для збору, первинної обробки, збереження та візуалізації даних від промислового обладнання з вбудованими механізмами захисту.

Об'єкт дослідження – це процеси дистанційного моніторингу, збору та захищеної передачі телеметричних даних від промислового обладнання в умовах недовірених мереж.

Методи дослідження: методи системного аналізу для проєктування розподіленої архітектури, концепція периферійних обчислень для агрегації

даних, методи криптографічного захисту інформації, теорія реляційних баз даних та документо-орієнтованих баз даних часових рядів.

Предмет дослідження – архітектурні рішення, алгоритми збору, шифрування, безпечної передачі та зберігання телеметричної інформації між вузлами системи.

Завдання дослідження:

1. Проведення аналізу предметної області, існуючих рішень дистанційного моніторингу та методів побудови систем для збору телеметрії.
2. Розробка загальної архітектури клієнт-серверної системи (клієнта для збору даних, серверного REST API та бази даних часових рядів).
3. Розробка програмного модуля периферійного шлюзу для опитування обладнання за протоколом Modbus TCP.
4. Проєктування та програмна реалізація алгоритмів шифрування і безпечної передачі телеметричної інформації між вузлами системи.
5. Налаштування підсистеми збереження даних та інтеграція інструментів візуалізації.
6. Тестування створених окремих компонентів програмного забезпечення та всієї системи в цілому.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ

1.1 Аналіз систем промислового моніторингу

Сучасне виробництво сильно змінилося завдяки технологіям. Тепер машини не працюють самі по собі, а пов'язані між собою через інтернет. Головна ідея полягає в тому, щоб машини повідомляли про свій стан, щоб інженери бачили все, що відбувається, в режимі реального часу.

Це повністю змінює те, як підприємства керують своїми машинами. Раніше компанії використовували традиційний підхід до обслуговування обладнання, який базувався на двох стратегіях: реактивній або планово-попереджувальній. Тепер системи Інтернету речей для промисловості дають змогу застосовувати стратегію передбачуваного обслуговування [1].

Система постійно стежить за тим, як працюють машини, і помічає навіть найменші проблеми. Це дає змогу вирішити проблему до того, як машина зламається і зупинить усе виробництво. Крім того, постійний моніторинг дозволяє автоматично підраховувати загальну ефективність роботи обладнання.

Незважаючи на переваги, розробка та впровадження таких систем стикається з двома архітектурними та технічними викликами:

1. Проблема старого обладнання. Більшість промислових контролерів, які зараз працюють на заводах, були спроєктовані для місцевої, окремої роботи. Вони використовують особливі промислові протоколи зв'язку, не завжди підтримують сучасні стандарти передачі даних та не мають вбудованих механізмів інтеграції з хмарними сервісами.

2. Проблема обсягу та швидкості генерації даних. Сучасне технологічне обладнання може бути опитуваним з частотою від десяти до ста разів на секунду, генеруючи сотні та тисячі показників щосекунди. Пряма передача такого великого обсягу даних від багатьох установок до центрального сервера призводить до перевантаження місцевих мереж

підприємства та нераціонального використання дискового простору на серверах баз даних.

Наявність цих викликів означає потрібність відмови від класичної централізованої моделі та перехід до розподіленої архітектури з використанням периферійних обчислень [2]. Тепер первинна обробка, агрегація та фільтрація даних відбуватиметься безпосередньо біля джерела їх виникнення. Лише після цього очищена інформація передається на центральний сервер.

1.2 Аналіз існуючих рішень дистанційного моніторингу

Існує багато готових рішень для дистанційного моніторингу. Але кожне з них має свої великі обмеження, коли мова йде про безпечний збір даних. На великих підприємствах часто використовують класичні системи керування та збору даних, такі як Siemens WinCC або AVEVA [3]. Це надійні інструменти, які були перевірені роками, але використання їх виключно для задач збору метрик часто виявляється економічно та архітектурно недоцільним. Головна проблема полягає у їхній моделі ліцензування, за якою підприємство мусить платити за кожен окремий параметр, який вони хочуть відстежувати. Сучасна промислова установка може створювати тисячі параметрів кожну секунду, тому вартість такої системи зростає дуже швидко. Крім того, класичні системи керування та збору даних не дуже гнучкі: вони прив'язують підприємство до одного виробника та важко налаштовуються для безпечної передачі даних через інтернет без побудови складних та дорогих спеціальних мереж.

Як альтернатива таким системам, останні роки показали, що хмарні платформи IIoT, такі як AWS IoT, Microsoft Azure або PTC ThingWorx, стають все популярнішими [4]. Вони пропонують багато можливостей для аналітики та роботи з великими даними, що допомагає вирішувати проблему масштабування. Проте, разом з цим, виникають нові труднощі. По-перше, хмарні рішення потребують постійного і стабільного підключення до глобальної мережі, чого не завжди можна досягнути в умовах цеху. По-друге,

виникає проблема цифрового суверенітету. Застосування популярних хмарних систем моніторингу часто стикається з труднощами у секторі критичної інфраструктури та військово-промислового комплексу. Це відбувається тому, що дані сектори мають дуже суворі правила безпеки та великий ризик витоку важливої інформації. Через це маршрутизація через сервери за кордоном для обробки даних є вкрай небажаною. Підприємства віддають перевагу автономним системам. Такі системи встановлюються безпосередньо на обладнанні підприємства та забезпечують його незалежність від зовнішніх постачальників послуг. Це дозволяє їм самостійно контролювати свої дані та забезпечувати їхню безпеку.

Популярні відкриті рішення могли б стати хорошим вибором. До них можна віднести використання візуальних середовищ, таких як Node-RED, разом із часовими базами даних [5]. Їх легко розгорнути, і вони не потребують купівлі ліцензій. Проте у них є одна велика проблема. Вони не використовують апаратні ресурси дуже ефективно. Якщо застосувати інтерпретовані мови та середовища на периферійних шлюзах, це приведе до надмірного споживання оперативної пам’яті та процесорного часу під час обробки великих потоків телеметрії від багатьох установок. Крім того, базові відкриті рішення зазвичай не мають стандартних механізмів захисту каналу зв'язку за допомогою криптографії.

Сучасним підприємствам потрібне легке, високопродуктивне рішення, яке можна повністю розгорнути на власних серверах підприємства без прив’язки до хмарних вендорів. Воно має ефективно працювати на малопотужних промислових комп’ютерах і забезпечувати захист даних на рівні протоколу.

1.3 Огляд протоколів передачі даних та їх вразливості

Протокол Modbus є одним із найстаріших та найпоширеніших стандартів обміну даними у сфері промислової автоматизації. Його розробили у 1979 році для програмованих логічних контролерів [6]. Потім його змінили

для сучасних мереж Ethernet і назвали Modbus TCP. Сьогодні він вважається стандартом промислового зв'язку, тому що він відкритий, простий і не має обмежень ліцензій. Більшість сучасних верстатів з ЧПК, датчиків і контролерів підтримують цей протокол, тому він став загальною мовою спілкування на рівні цеху, незалежно від компанії-виробника обладнання. Протокол Modbus дуже широко використовується, тому його можна вважати універсальним способом спілкування у сфері промислової автоматизації.

Архітектура Modbus базується на клієнт-серверній моделі обміну повідомленнями. Технологічна установка виступає в ролі сервера, який очікує на вхідні з'єднання на мережевому порту 502 [7]. Розроблюваний периферійний шлюз відіграє роль клієнта, який ініціює запити на читання. Інформаційна модель протоколу має наступний вигляд: усі дані всередині контролера представлені у вигляді 16-бітних регістрів або однобітних прапорців. Щоб отримати актуальну телеметрію, клієнт формує запит із вказівкою стартової адреси регістра та кількості байтів для читання, після чого обладнання повертає запитуваний числовий масив.

Саме ця простота була дуже добре підходила для старих систем. Протокол Modbus було створено понад сорок років тому для роботи в мережах підприємства, які були повністю ізольовані від зовнішнього світу, де не передбачалося існування внутрішніх чи зовнішніх загроз. Як наслідок, у стандарті Modbus TCP немає механізмів автентифікації та авторизації пристроїв [8]. Будь-який вузол мережі, який може встановити з'єднання з обладнанням, може зчитувати важливі дані й навіть надсилати команди на примусовий запис у керуючі регістри.

Більше того, протокол не пропонує ніяких засобів криптографічного захисту. Усі дані та команди передаються по мережі у відкритому вигляді. Це означає, що зловмисник може легко займатися промисловим шпигунством. Крім того, він може змінювати дані про стан обладнання, щоб приховати проблеми від диспетчера.

Підключення такого вразливого обладнання до корпоративних мереж або до інтернету створює дуже великі ризики для підприємства. Саме ця проблема робить неможливим безпечне створення сучасних систем промислового інтернету речей без додаткового захисту. Цю роль виконує спеціальний периферійний шлюз. Він спілкується з обладнанням по протоколу Modbus тільки в межах захищеного сегмента локальної мережі. Далі він упаковує дані в захищені криптографічні контейнери. Такі дані можна безпечно передавати на центральний сервер через не дуже захищені мережі.

1.4 Вектори атак та ризики компрометації телеметрії

Архітектурні недоліки класичних промислових протоколів, описані у попередньому підрозділі, формують широку поверхню для атак. Коли промислові мережі з'єднуються з іншими комп'ютерними мережами компанії, захист стає менш ефективним. У разі проникнення зловмисника у внутрішню мережу підприємства, незахищені потоки телеметрії стають мішенню для реалізації низки векторів атак, які можна класифікувати за впливом на базові принципи інформаційної безпеки.

Перший і найпоширеніший вектор атак – це пасивне перехоплення трафіку, яке порушує принцип конфіденційності. Протокол Modbus TCP працює з відкритим текстом. Зловмисник може легко отримати доступ до мережі і використати аналізатори трафіку для збору даних про роботу підприємств. Це не пошкоджує обладнання, але дозволяє конкурентам дізнатися про справжні обсяги виробництва, як працює обладнання, і які алгоритми застосовуються. Хоча безпосередньої шкоди обладнанню це не завдає, такий вектор створює умови для промислового шпигунства: аналіз телеметрії дозволяє конкурентам вирахувати реальні обсяги виробництва, специфічні налаштування режимів або алгоритми роботи установок, які є комерційною таємницею підприємства.

Значно більшу загрозу становлять атаки на цілісність даних, особливо атаки типу «людина посередині» та підміна даних [9]. Зловмисник може

перенаправити трафік між установкою і системою моніторингу через свій вузол. Це дає змогу не лише читати, а й модифікувати пакети даних. У результаті система моніторингу на центральному сервері буде показувати, що установка працює нормально, приховуючи реальну проблему від людини, яка слідкує, аж до моменту, коли обладнання зламається зовсім.

Крім того, відсутність автентифікації джерела даних робить систему вразливою до атак повторного відтворення та несанкціонованої ін'єкції команд. Зловмисник може записати справжній пакет даних про роботу установки та надсилати його на сервер багато разів, роблячи вигляд, що це відбувається з самого обладнання. Або він може створити шкідливий запит на запис інформації безпосередньо у контролер обладнання, що може вивести з ладу систему охолодження або змінити швидкість обертання двигунів.

Нарешті, промислові контролери вразливі до атак на відмову в обслуговуванні. Мережеві модулі більшості ПЛК мають вкрай обмежені обчислювальні ресурси оперативної пам'яті та слабку систему захисту від атак на мережу. Навіть велика кількість справжніх запитів до системи контролю обладнання може перевантажити контролер.

Програмно-апаратний комплекс повинен забезпечувати надійну ізоляцію вразливого обладнання, автентифікацію вузлів збору та наскрізне шифрування телеметрії перед її передачею до серверної бази даних.

1.5 Постановка задачі та формування вимог до системи

Під час аналізу галузі та виявлення проблем промислових мереж, основним завданням цієї роботи є створення комплексного рішення, яке безпечно та ефективно підключає старе обладнання до сучасної IT-інфраструктури. Основою цієї системи має стати програма-клієнт, яка дозволить мінімізувати використання апаратних ресурсів. Програма буде діяти як розумний посередник: вона повинна опитувати технологічні установки за допомогою протоколу Modbus, але замість відправки кожного отриманого байта на центральний сервер – виконувати попередню обробку

даних. Для цього потрібно створити спеціальний алгоритм, який буде видаляти несуттєві або випадкові зміни показників.

Зібрана й відфільтрована телеметрія повинна миттєво передаватися на серверну частину комплексу. Ця інформація повинна маршрутизуватися й зберігатися у спеціалізованій базі даних часових рядів, яка працює з неперервними потоками мікроподій.

Одна з найважливіших вимог до розроблюваної системи – це забезпечення безпеки. Оригінальний протокол Modbus не має захисту, тому всі дані, які покидають межі цеху й передаються через зовнішні мережі, повинні бути захищені. Система повинна використовувати токени доступу для клієнтських вузлів, щоб не дозволити підключення неавторизованих пристроїв. Але токени й дані не можуть передаватися у відкритому вигляді. Клієнт повинен передавати всю телеметрію у криптографічних контейнерах, використовуючи державні алгоритми шифрування та електронного підпису.

Висновок до розділу 1

У першому розділі проаналізовано проблеми цифровізації виробництв та систем промислового моніторингу. З'ясовано, що старі контролери генерують величезні обсяги даних, а існуючі SCADA-системи, хмарні IoT-платформи або відкриті рішення є дуже дорогими, ресурсоємними або не відповідають вимогам безпеки критичної інфраструктури.

Окремо розглянуто проблему безпеки протоколу Modbus TCP, який працює без автентифікації та передає дані у відкритому вигляді. Пряме підключення такого обладнання до сучасних мереж створює широку поверхню для різних кібератак.

На основі виявлених недоліків сформовано вимоги до розробки власного програмно-апаратного комплексу. Таким рішенням має стати простий периферійний шлюз. Він локально збиратиме, оброблятиме і об'єднуватиме дані з обладнання, забезпечуючи їхній обов'язковий захист перед відправкою до центральної бази даних часових рядів через недовірені мережі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Загальна топологія клієнт-серверного комплексу

Для вирішення завдань, сформульованих на етапі аналізу предметної області, було спроєктовано розподілену клієнт-серверну архітектуру. Її головна мета – забезпечити надійну ізоляцію вразливого промислового обладнання від зовнішніх мереж, оптимізувати обсяги переданих даних та гарантувати криптографічний захист телеметрії. Загальна топологія розробленого програмно-апаратного комплексу складається з трьох ключових рівнів (рис. 2.1):

- рівень периферійних обчислень та збору даних;
- рівень захищеної передачі даних;
- рівень централізованої обробки та візуалізації.

Рівень периферійних обчислень та збору даних розташовується безпосередньо у виробничих приміщеннях підприємства. Основними елементами цього рівня є технологічні установки, такі як промислові верстати з ЧПК, гідравлічні преси та конвеєрні лінії. Вони функціонують в ізольованій локальній мережі та підтримують обмін даними за протоколом Modbus TCP. Центральним вузлом цього рівня є розроблений периферійний програмний шлюз. Архітектурно він являє собою легкосумісну службу, що розгортається на локальному промисловому комп'ютері. Уся поведінка шлюзу керується конфігураційним файлом у форматі JSON. У цьому файлі визначається топологія опитування, яка включає:

- перелік технологічних установок, їх IP-адреси та порти;
- специфікація регістрів для кожного типу обладнання;
- індивідуальні інтервали опитування.

Крім того, шлюз не відправляє кожен показник окремо. Він збирає дані з багатьох установок одночасно, групує їх у спільний пакет і лише після цього відправляє на сервер.



Рисунок 2.1 – Загальна топологія програмно-апаратного комплексу.

Рівень захищеної передачі даних забезпечує асинхронну передачу даних від шлюзів до центрального сервера через недовірені мережі. Враховуючи вимоги безпеки, на цьому рівні не передаються сирі дані. Зібраний на попередньому етапі масив телеметрії разом з ідентифікаційним токеном шлюзу інкапсулюється у криптографічний контейнер. Транспортування цього контейнера здійснюється поверх стандартного протоколу HTTP за допомогою REST API методів.

Рівень централізованої обробки та візуалізації складається з трьох взаємопов’язаних підсистем: модуля прийому, підсистеми збереження та

підсистеми візуалізації. Модуль прийому працює як окрема служба, яка не зберігає жодних даних. Коли він отримує запит від шлюзу, сервер розшифровує його. Валідація прав доступу відбувається за допомогою конфігураційного списку дозволених клієнтів, який завантажується в оперативну пам'ять сервера. Відмова від використання реляційної бази даних на етапі автентифікації дозволяє мінімізувати затримки. Відмова від використання звичайної бази даних під час перевірки прав доступу допомагає зменшити затримки. Після успішної перевірки отримані дані розпаковуються, потім вони розбиваються на частини та відправляються до бази даних часових рядів. Кінцевий користувач взаємодіє з системою через незалежну підсистему візуалізації, яка виконує запити до бази даних та відображає стан обладнання у вигляді графіків та індикаторів.

2.2 Проєктування логіки периферійного шлюзу

Периферійний шлюз – це частина системи, яка постійно взаємодіє з промисловим обладнанням та збирає його дані. Життєвий цикл шлюзу розпочинається з етапу ініціалізації, під час якого відбувається парсинг конфігураційного файлу. Логіка роботи програми у такому випадку відокремлена від фізичної топології цеху. Додавання нових установок, зміна їхніх IP-адрес або модифікація списку цільових регістрів здійснюється без необхідності перекомпіляції вихідного коду.

Промислове обладнання в цеху працює з різною інтенсивністю, тому шлюз спроєктовано так, щоб він міг опитувати кожну установку за її власним графіком. Наприклад, гідравлічний прес може змінити свої показники дуже швидко, тому програма повинна запитувати дані з нього дуже часто – кожні пів секунди. Але для конвеєрної стрічки чи температури двигуна достатньо перевіряти дані раз на п'ять секунд.

Щоб шлюз міг добре працювати й виконувати всі ці завдання одночасно, під час запуску програми, читається конфігураційний файл і створюється окремий потік для кожної установки. Якщо якесь обладнання вийде з ладу або

його кабель пошкодиться, програма не зупиниться й не буде чекати відповіді, а інші потоки продовжать збирати дані з своїх верстатів.

Сам процес збору даних працює наступним чином. Шлюз паралельно звертається до різних IP-адрес та портів установок у мережі. Протокол Modbus віддає інформацію у вигляді звичайних байтів. Завдання шлюзу на цьому етапі – перетворити їх у зрозумілі фізичні величини. Програма автоматично обробляє отримані дані відповідно до налаштувань: наприклад, звичайні цілі числа залишає як є, а там, де потрібна висока точність, склеює два регістри разом, щоб отримати дробове число. Після цієї обробки метрики від усіх верстатів зводяться в один спільний список в оперативної пам’яті шлюзу, де вони очікують на пакетну відправку до сервера.

Для підтримки постійного стану обладнання в системі передбачено механізм серцебиття. Навіть якщо верстат знаходиться в режимі очікування, шлюз із заданою періодичністю генерує контрольні сигнали, що дозволяє серверній частині відрізняти нормальний простій обладнання від аварійного обриву зв’язку.

Іншою важливою функцією шлюзу є оптимізація вихідного мережевого трафіку. Якщо встановлювати нове з’єднання для кожної окремої метрики, це призвело б до великого навантаження на мережу і перевантаження центрального сервера. Для вирішення цієї проблеми реалізовано алгоритм буферизації та відправки даних пакетами. Замість негайної відправки, зібрані метрики накопичуються в оперативній пам’яті шлюзу. Процес відправки цих метрик відбувається, коли відбувається одна з двох запланованих подій: коли кількість записів у пам’яті досягає максимального ліміту, щоб не перевантажити пам'ять при великій кількості даних, або коли закінчується певний часовий інтервал, щоб гарантувати актуальність даних навіть при низькій активності.

Детальний алгоритм паралельної обробки даних, нормалізації метрик та роботи потоку-диспетчера наведено на рисунку 2.2.

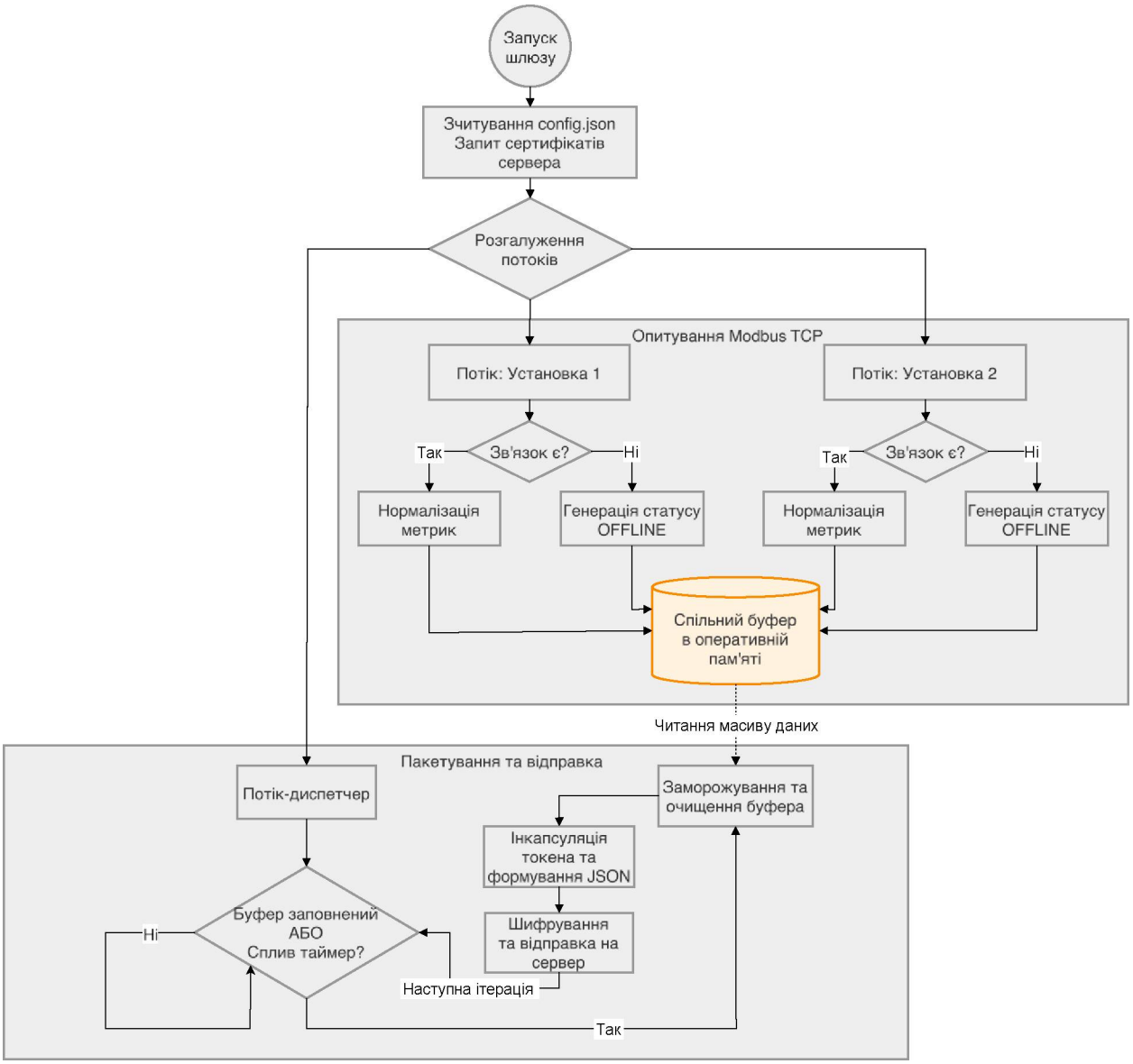


Рисунок 2.2 – Схема обробки даних на периферійному шлюзі.

2.3 Безпека, автентифікація та захист даних

Після збору, обробки та підготовки даних обладнання до відправки постає питання про безпечну передачу цих даних на центральний сервер. Оскільки канал зв’язку, будь то корпоративна мережа або інтернет, не вважається повністю безпечним, архітектура системи була розроблена з урахуванням безпеки з самого початку.

У багатьох веб-додатках для перевірки доступу використовуються токени, які надсилаються у відкритому вигляді у заголовках запитів HTTP. Однак для системи моніторингу промислового обладнання такий підхід є

недостатньо безпечним. Якщо хтось перехопить дані, він зможе одразу отримати ідентифікатор і використовувати його для шкідливих цілей. Тому в розробленій системі застосовано принцип глибокої інкапсуляції: секретний токен клієнта поміщається безпосередньо всередину JSON-пакета разом із показниками телеметрії. Далі цей пакет перетворюється на захищений криптографічний контейнер.

Для побудови захищеного каналу обміну даними система використовує інфраструктуру відкритих ключів. Спочатку клієнтський шлюз запитує у центрального сервера його відкриті сертифікати. Один сертифікат необхідний для шифрування даних, а інший - для перевірки електронного підпису. Обидва сертифікати генеруються на базі єдиного закритого ключа сервера згідно з стандартом цифрового підпису ДСТУ 4145-2002 [10].

Під час передачі даних відбувається шифрування корисного навантаження шляхом встановлення динамічно захищеної сесії. Протокол розподілу ключових даних реалізований у відповідності до ДСТУ ISO/IEC 11770-3:2023 [11]. Після цього виконується пакування зашифрованої телеметрії у цифровий конверт згідно з технічними специфікаціями до міжнародних рекомендацій IETF RFC 5652 [12], які затверджені наказом Адміністрації Держспецзв'язку №687 від 27.10.2020 р [13].

На стороні сервера процес прийому розроблено без збереження стану сесій. Коли сервер отримує зашифрований контейнер, він розшифровує його за допомогою свого особистого ключа. Після цього він вилучає звідти клієнтський ідентифікатор і відразу перевіряє, чи є він у списку тих, яким дозволено працювати, який зберігається в оперативній пам'яті.

Під час перевірки ідентифікатора сервер діє швидко, щоб не сповільнювати роботу. Для цього він не здійснює пошук в базі даних для кожного окремого повідомлення. Натомість він один раз зчитує інформацію зі спеціального файлу під час запуску і зберігає її в пам'яті. Якщо ідентифікатор клієнта відповідає тому, що є в списку, сервер далі обробляє інформацію і записує її в базу даних.

Якщо ідентифікатор клієнта неправильний або повідомлення неможливо розшифрувати, сервер одразу припиняє зв'язок та повідомляє клієнту про те, що щось пішло не так.

Повну послідовність криптографічної взаємодії, від отримання сертифікатів до підтвердження запису в базу даних, наведено на рисунку 2.3.

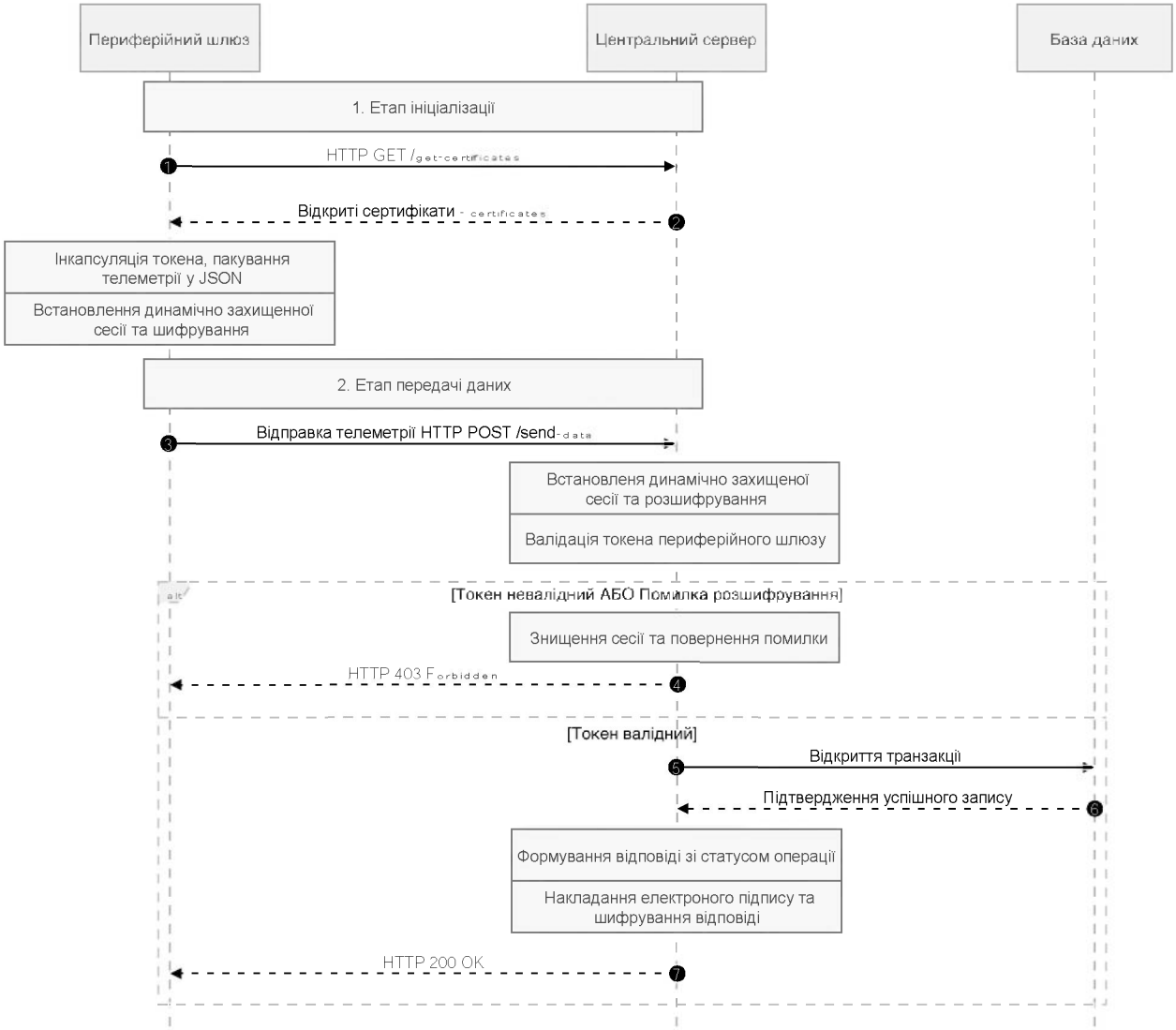


Рисунок 2.3 – Діаграма захищеної передачі телеметрії.

2.4. Проєктування логічної структури сховища даних

Останнім кроком роботи з інформацією є її довготривале збереження. Дані, які надходять від промислового обладнання, мають специфічний характер. Вони генеруються безперервно, з високою частотою і відображають стан установки в конкретний момент часу. Важливою особливістю такої

інформації є те, що вона ніколи не змінюється і не видаляється в процесі роботи – система функціонує виключно на додавання нових записів. Через це використання класичних реляційних моделей сховищ у їхньому базовому вигляді є технічно неефективним, адже при накопиченні мільйонів рядків швидкість запису та пошуку почне стрімко падати.

Для вирішення цієї проблеми логічна архітектура сховища проєктується на базі концепції баз даних часових рядів [14]. Головна ідея цього підходу полягає в тому, що основним індексом для організації даних та оптимізації дискових операцій стає часова відмітка. Завдяки цьому система здатна групувати та записувати метрики великими послідовними блоками, мінімізуючи навантаження на дискову підсистему сервера.

Оскільки всі налаштування автентифікації шлюзів, списки дозволених клієнтів та карти регістрів обладнання винесені у конфігураційні файли й обробляються сервером (або клієнтом) безпосередньо в оперативній пам’яті, структура бази даних звільняється від необхідності зберігати складні таблиці зв’язків чи довідники пристроїв. Це дозволяє спроєктувати логічну модель даних як лінійну, але максимально оптимізовану структуру часових подій. Кожен запис у сховищі містить декілька атрибутів:

- час фіксації показників на стороні клієнта;
- унікальний ідентифікатор конкретного обладнання;
- поточний статус роботи установки;
- показники датчиків обладнання у вигляді бінарного JSON-об’єкта.

Логічну схему трансформації інформаційних потоків на сервері, починаючи від дешифрації вхідного криптоконтейнера і закінчуючи розподілом зазначених атрибутів у сховищі та їх передачею до підсистеми візуалізації, наведено на рисунку 2.4.

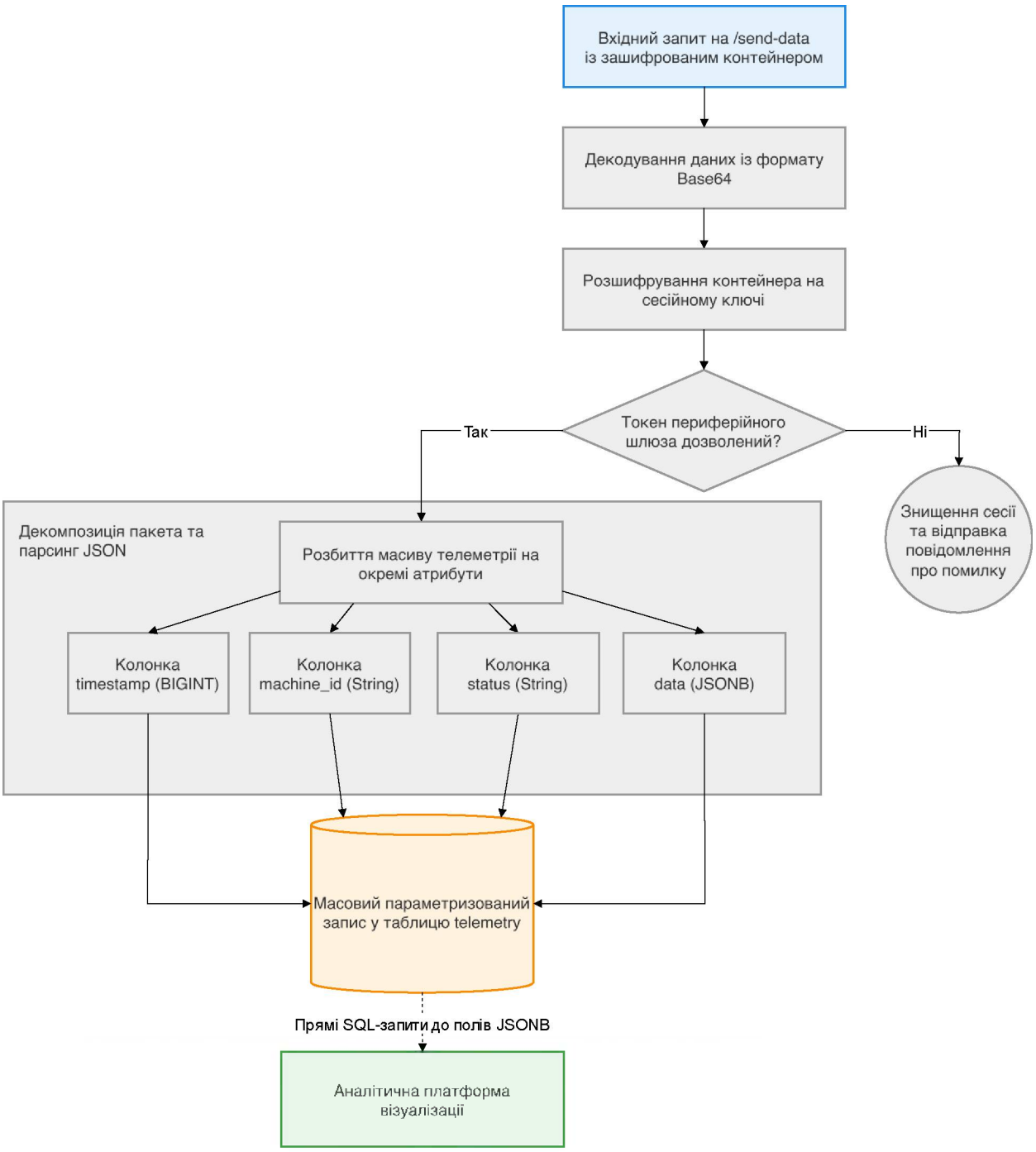


Рисунок 2.4 – Діаграма потоків даних під час обробки та збереження телеметрії.

Промислове обладнання розрізнене – різні верстати можуть передавати різні дані. Документо-орієнтований підхід дозволяє зберігати будь-який набір метрик у межах однієї таблиці. Це також надає системі велику гнучкість при масштабуванні: якщо в майбутньому до установки буде підключено новий

датчик, його показники автоматично почнуть зберігатися у базі без необхідності зупинки сервера чи модифікації схеми бази даних.

Структура бази даних складається з двох частин: реляційних колонок і JSONB. Це ідеально підходить для підсистеми візуалізації даних. Графіки на диспетчерських панелях завжди показують зміну параметрів за певний час або відфільтровують відключені верстати. Тому окремі колонки для часу, ідентифікатора та статусу дозволяють базі даних створювати ефективні індекси. Це гарантує швидку вибірку необхідних даних навіть при великих обсягах інформації.

Висновок до розділу 2

У другому розділі спроєктовано загальну трирівневу архітектуру системи, яка включає рівні периферійних обчислень, захищеної передачі та централізованої обробки даних.

Розроблено логіку багатопотокового периферійного шлюзу, який опитує вразливе обладнання за протоколом Modbus TCP і застосовує пакетування та буферизацію для оптимізації мережевого трафіку.

Для гарантування безпеки передачі даних впроваджено механізм інкапсуляції, за яким токен клієнта розміщується всередині JSON-пакета разом із телеметрією, після чого весь інформаційний блок перетворюється на захищений криптографічний контейнер на базі державних стандартів.

Збереження інформації організовано за концепцією баз даних часових рядів із використанням гібридного підходу: базові параметри індексуються в реляційних колонках, а динамічні метрики датчиків зберігаються у форматі JSONB.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Вибір технологічного стеку

При виборі кожної технології було розглянуто основні варіанти, які використовуються в індустрії, щоб знайти оптимальний баланс між продуктивністю, безпекою та швидкістю розробки.

3.1.1 Основна мова програмування

Зазвичай REST API сервери пишуться на Python або Java. Ці мови забезпечують дуже швидку розробку програм. Однак інтерпретовані мови, які мають збирач сміття, можуть мати непередбачувані затримки під час виконання коду і споживають багато оперативної пам'яті. Оскільки периферійний шлюз повинен працювати прямо в цеху, часто на малопотужних промислових комп'ютерах, ці ресурси є дуже обмеженими. У цьому випадку мова C++ забезпечує прямий доступ до пам'яті, гарантує високу швидкість обробки бінарних пакетів Modbus і дозволяє системі працювати роками без проблем з пам'яттю [15]. Крім того, використання однієї мови для клієнта і сервера дозволяє зробити кодову базу однорідною.

3.1.2 Мережева взаємодія та HTTP-сервер

Серед популярних бібліотек і фреймворків мовою C++ найбільше виділяються Boost.Beast, Poco та httplib. Занадто великі бібліотеки на кшталт Boost або Poco є стандартами і використовуються всюди, але вони неймовірно перевантажені функціоналом, який не потрібен для цього проєкту, і вимагають складної конфігурації при компіляції. Замість них було прийнято рішення використати бібліотеку cpp-httplib [16]. Вона складається з одного заголовного файлу і легко інтегрується в проєкт. Крім того, вона підтримує пул потоків, що означає, що наш сервер автоматично виділяє окремий потік для кожного нового запиту. Це забезпечує справжню асинхронність без ускладнення коду.

3.1.3 Форматування даних

Для формування JSON було обрано бібліотку nlohmann/json [17]. У якості альтернативи виступала бібліотека RapidJSON, яка працює трохи швидше, але дуже не зручна у користуванні. Натомість бібліотека від nlohmann/json вважається стандартом для сучасного C++.

3.1.4 Система керування базами даних

Під час вибору системи керування базами даних як альтернативи розглядалися MongoDB, MySQL, PostgreSQL та спеціалізовані рішення для часових рядів (InfluxDB та ін.). Хоча для роботи з телеметрією часто використовують InfluxDB або документо-орієнтовану MongoDB, оптимальним вибором для даного проєкту стала PostgreSQL.

Таке рішення обґрунтоване гібридними потребами розробленої системи до структури даних. З одного боку, система потребує реляційної структури для зберігання базових пошукових атрибутів та побудови ефективних індексів. З іншого боку, безпосередні показники датчиків є абсолютно динамічними для різних типів верстатів.

PostgreSQL ідеально вирішує цю архітектурну проблему завдяки підтримці бінарного формату JSONB [18]. Цей тип даних дозволяє зберігати неструктуровані показники будь-яких датчиків у межах єдиної таблиці телеметрії, зберігаючи при цьому можливість виконання прямих SQL-запитів до внутрішніх полів JSON-об’єкта. Це забезпечує дуже швидкий пошук, зручну інтеграцію з підсистемою візуалізації та побудову аналітичних графіків без необхідності складної програмної десеріалізації на рівні сервера.

3.1.5 Криптографічний захист

Бібліотека OpenSSL є світовим стандартом для шифрування, але він базується на міжнародних криптографічних алгоритмах. Оскільки система розробляється для українських промислових підприємств, вона повинна відповідати вимогам КСЗІ. Тому для шифрування пакетів та електронного

підпису було обрано бібліотеку EUSign від АТ «Інститут інформаційних технологій» [19]. Вона реалізує державні стандарти та забезпечує юридичну і технічну легітимність захищеного каналу зв'язку.

3.2 Розробка програмного модуля шлюзу

Розробка клієнтської частини комплексу фокусується на створенні автономної, відмовостійкої служби, яка здатна роками працювати на виробництві без втручання людини. Життєвий цикл програми розпочинається з ініціалізації та зчитування файлу config.json. Використовуючи бібліотеку nlohmann/json, C++ програма десеріалізує конфігурацію і перетворює її на власні внутрішні структури даних.

Після успішної ініціалізації конфігурації шлюз переходить до етапу збору телеметрії. Для реалізації одночасного опитування розрізненого обладнання розроблено багатопотокову архітектуру. Програма проходить по масиву з обладнанням і для кожної установки динамічно створює окремий системний потік. Усередині кожного потоку ініціюється TCP-з'єднання з відповідним IP та портом за протоколом Modbus.

Кожен потік працює у власному нескінченному ізольованому циклі, періодичність якого задається конфігураційним параметром poll_interval_ms. Отримана від контролера серія сирих 16-бітних регістрів нормалізується – програма автоматично склеює або трансформує байти у потрібні типи даних (int16 або float32) згідно з картою регістрів станка. Важливою деталлю є механізм діагностики з'єднання. Якщо зв'язок з одним із верстатів обривається, його персональний потік переходить у режим очікування і продовжує генерувати записи для буфера, але явно фіксує для цього обладнання статус "OFFLINE" із порожнім набором метрик. Таким чином, кожен верстат завжди присутній у фінальному пакеті з актуальним поточним статусом, що дозволяє серверу та підсистемі візуалізації миттєво фіксувати аварійні відключення.

Паралельно з потоками опитування працює потік-диспетчер, який відповідає за пакетування та відправку. Він постійно моніторить стан спільного буфера. Щойно кількість записів досягає ліміту max_batch_size, або спливає таймер flush_interval_sec, диспетчер заморожує буфер, очищає його для нових даних, і формує фінальний інформаційний пакет для сервера.

Саме на цьому етапі реалізується закладений в архітектуру принцип безпеки: ідентифікатор шлюзу інкапсулюється всередину корисного навантаження. З урахуванням механізму явного маркування статусів та індивідуальних міток часу для кожного агрегату, сформований пакет перед початком криптографічних перетворень має ієрархічну структуру (лістинг 3.1).

Лістинг 3.1 – Приклад сформованого JSON-пакета телеметрії

```
{
  "client_token": "client_workshop_1",
  "telemetry": [
    {
      "machine_id": "CNC_MILL_01",
      "timestamp": 1778707163,
      "status": "ONLINE",
      "sensors": {
        "temperature": 45,
        "pressure_bar": 12.5
      }
    }
  ]
}
```

Детальний опис кожного поля інформаційного пакета, його типів даних та обмежень наведено у таблиці 3.1.

Таблиця 3.1

Специфікація полів пакета передачі телеметрії

Назва поля	Тип	Обмеження	Опис	Обов'язковість
client_token	string	макс. 255 символів	Ідентифікатор авторизації периферійного шлюзу.	+

Продовження табл. 3.1

telemetry	array [object]	мін. 1 елемент	Масив об’єктів, що містить зібрані дані з одного або декількох верстатів.	+
machine_id	string	без пробілів	Унікальний ідентифікатор конкретного агрегату.	+
timestamp	integer (BIGINT)	64-бітне число	Локальний час фіксації показників на стороні шлюзу у форматі Unix Timestamp.	+
status	string	“ONLINE” або “OFFLINE”	Поточний стан підключення шлюзу до контролера верстата.	+
sensors	object (JSON)	-	Об’єкт ключ-значення, що містить показники датчиків. Значення можуть бути як цілими, так і дробовими числами. Може бути порожнім {} у випадку статусу “OFFLINE”.	+

Сформований структурований пакет телеметрії у вигляді рядка тексту передається до модуля криптографічного захисту, де перетворюється на зашифрований бінарний контейнер, після чого потік-диспетчер ініціює його відправку на центральний сервер за допомогою HTTP-клієнта з бібліотеки cpp-httplib.

3.3 Реалізація серверної частини

Розробка центрального вузла прийому даних базується на використанні багатопотокового C++ мікрофреймворку cpp-httplib. Головною вимогою до серверної частини є здатність обробляти велику кількість одночасних вхідних з'єднань.

Життєвий цикл сервера розпочинається з етапу ініціалізації середовища безпеки та конфігурації. За допомогою бібліотеки nlohmann/json сервер зчитує файл server_config.json, витягує з нього масив allowed_clients і завантажує його у швидку структуру даних в оперативній пам'яті. Валідація токенів відбувається з алгоритмічною складністю $O(1)$ без звернень до бази даних. Паралельно з цим ініціалізується криптографічна бібліотека EUSign, яка завантажує у пам'ять закритий ключ сервера.

Взаємодія зі шлюзами реалізована через REST API [20]. Для прийому зашифрованої телеметрії на сервері оголошується захищений ендпоїнт /send-data, який обробляє виключно POST-запити. Коли від периферійного шлюзу надходить новий HTTP-запит, сервер виконує таку послідовність дій:

1. Сервер отримує тіло запиту у форматі JSON, яке містить авторизаційні дані для відновлення сесії "authData" та власне зашифроване корисне навантаження "encryptedData". Дані декодуються з формату Base64 у бінарні масиви.

2. Сервер передає бінарний масив "authData" до модуля EUSign для ініціалізації динамічної захищеної сесії.

3. Використовуючи відновлену сесію, сервер розшифровує “encryptedData”. У разі успіху криптопровайдер повертає рядок зі структурованим масивом телеметрії у форматі JSON.

4. Система витягує з розшифрованого JSON-об’єкта значення поля “client_token” і звіряє його з кешованим списком дозволених клієнтів.

5. Якщо на будь-якому з етапів виникає помилка, сервер перериває обробку. Криптографічна сесія знищується для запобігання витоку пам’яті, а клієнту повертається відповідний HTTP-статус із JSON-повідомленням про характер помилки.

6. Після успішної комплексної перевірки валідний масив телеметрії передається до внутрішнього модуля “DatabaseManager”. Запис у базу даних PostgreSQL виконується в рамках єдиної транзакції, при цьому показники датчиків зберігаються у стовпці типу JSONB.

7. Після успішного збереження сервер формує JSON-відповідь про статус операції. Для забезпечення безвідмовності сервер накладає на відповідь свій електронний підпис за стандартом ДСТУ 4145-2002, пакує її, а потім зашифровує в рамках поточної динамічної сесії. Зашифрований бінарний пакет кодується в Base64 і відправляється клієнту.

Використання вбудованого пулу потоків фреймворку cpp-httplib дозволяє виконувати описаний криптографічний конвеєр паралельно для великої кількості заводських шлюзів одночасно.

3.4 Специфікація форматів криптографічної взаємодії

Для забезпечення конфіденційності та цілісності даних під час їх передачі через недовірені мережі, розроблений комплекс використовує ієрархічну систему криптографічних контейнерів. Зібрана телеметрія (описана в підрозділі 3.2) ніколи не передається у відкритому вигляді. Натомість вона проходить етапи накладання електронного підпису та шифрування на сесійних ключах.

Взаємодія між периферійним шлюзом та центральним сервером відбувається шляхом обміну стандартизованими JSON-об'єктами, поля яких містять бінарні дані, закодовані у форматі Base64.

Ініціатором з'єднання завжди виступає клієнт. Для передачі даних він формує зашифрований запит, який містить необхідну інформацію для відновлення динамічної сесії на стороні сервера (лістинг 3.2).

Лістинг 3.2 – Структура зашифрованого запиту від клієнта до сервера

```
{
  "authData": "MIIB... (Base64)",
  "encryptedData": "U2Fsd... (Base64)"
}
```

Детальний опис полів зашифрованого запиту наведено у таблиці 3.2.

Таблиця 3.2

Зашифрований запит від клієнта

Назва поля	Тип	Обмеження	Опис	Обов'язковість
authData	string (Base64)	-	Дані автентифікації динамічної сесії для встановлення ключа шифрування.	+
encryptedData	string (Base64)	-	Корисне навантаження, зашифроване на ключі динамічної сесії.	+

Всередині зашифрованого контейнера "encryptedData" під час формування відповіді від сервера, використовується структура підписаних даних для забезпечення невідмовності та підтвердження авторства (лістинг 3.3).

Лістинг 3.3 – Структура підписаного контейнера

```
{
  "signedData": "MIID... (Base64)"
}
```

Детальний опис полів підписаного повідомлення наведено у таблиці 3.3.

Таблиця 3.3

Підписана відповідь від сервера

Назва поля	Тип	Обмеження	Опис	Обов'язковість
signedData	string (Base64)	-	Бінарний масив даних, що містить оригінальне повідомлення разом із накладеним електронним підписом.	+

Після успішної обробки телеметрії сервер формує відповідь. Оскільки сервер вже встановив і перевірив динамічну сесію з клієнтом завдяки полю "authData" з початкового запиту, йому не потрібно повторно передавати дані автентифікації. Відповідь сервера містить лише зашифровану на сесійному ключі підписану відповідь (лістинг 3.4).

Лістинг 3.4 – Структура зашифрованої відповіді від сервера

```
{
  "encryptedData": "VGVzd... (Base64)"
}
```

Детальний опис полів зашифрованої відповіді наведено у таблиці 3.4.

Таблиця 3.4

Зашифрована відповідь від сервера прийому даних

Назва поля	Тип	Обмеження	Опис	Обов'язковість
encryptedData	string (Base64)	-	Зашифровані дані на ключі поточної динамічної сесії.	+

3.5 Організація взаємодії з базою даних PostgreSQL

Після успішної перевірки авторизаційних токенів та розшифрування криптоконтейнера, розпакований масив телеметрії передається до внутрішнього модуля збереження даних. Для забезпечення надійної взаємодії сервера з реляційною базою даних PostgreSQL використовується офіційна клієнтська бібліотека libpqxx.

З метою гарантування максимальної ізоляції процесів та уникнення взаємних блокувань, архітектура модуля DatabaseManager реалізує стратегію незалежних сесій. Для обробки кожного нового валідного масиву телеметрії сервер ініціює нове ізольоване підключення до бази даних безпосередньо всередині методу insertTelemetryBatch.

Процес збереження розпочинається з ітерації по розшифрованому масиву telemetry. Оскільки один вхідний пакет від шлюзу зазвичай містить дані одразу від декількох верстатів, запис у базу виконується масово в межах єдиної транзакції. Якщо під час запису відбудеться помилка, транзакція буде скасована цілком.

Під час формування запиту до бази даних сервер здійснює декомпозицію JSON-об'єкта кожного верстата. Базові індексні поля та текстовий статус витягуються в окремі змінні, а безпосередні показники датчиків ізолюються для запису в спеціалізовану колонку. Для захисту від SQL-ін'єкцій та підвищення продуктивності система використовує виключно параметризовані запити. Логіка формування запиту має наступний вигляд (лістинг 3.5)

Лістинг 3.5 – Логіка формування запиту

```
INSERT INTO telemetry (timestamp, machine_id, status, data)
VALUES ($1, $2, $3, $4);
```

У цьому запиті параметри набувають таких значень:

- $1 (timestamp): витягується ціле 64-бітне число, що гарантує збереження оригінального часу фіксації метрики.

- $2 (machine_id): рядковий ідентифікатор, за яким база даних будує індекси для швидкого пошуку.
- $3 (status): текстовий статус роботи верстата, винесений в окрему колонку таблиці.
- $4 (data): безпосередньо об’єкт sensors, що перетворюється у бінарний тип JSONB.

Використання типу даних JSONB для показників датчиків зберігає повну гнучкість системи. Оскільки у колонку data потрапляють лише чисті метрики, синтаксис запитів до бази даних з боку підсистеми візуалізації залишається максимально простим. Графічні платформи здатні виконувати прямі SQL-запити до внутрішніх полів цього JSON-об’єкта.

Висновок до розділу 3

У третьому розділі здійснено практичну реалізацію програмного комплексу мовою C++. Розроблено багатопотоковий периферійний шлюз, який паралельно опитує обладнання за протоколом Modbus TCP, нормалізує дані та пакетує їх у JSON-контейнери.

Серверну частину побудовано як високопродуктивний асинхронний REST API. Збереження телеметрії організовано в СКБД PostgreSQL за допомогою гібридного підходу: базові пошукові атрибути винесено в реляційні колонки, а динамічні показники датчиків транзакційно записуються у гнучкому бінарному форматі JSONB.

Окрему увагу приділено підсистемі безпеки, реалізованій за державними криптографічними стандартами за допомогою бібліотеки EUSign. Запроваджено глибоку інкапсуляцію токенів авторизації, встановлення динамічних захищених сесій та шифрування, яке перекриває вразливості протоколу Modbus і гарантує безпечну передачу телеметрії через недовірені мережі.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

4.1 Апаратно-програмне середовище тестового стенда

Для проведення повноцінного тестування розробленого програмно-апаратного комплексу, оцінки його продуктивності та стабільності під навантаженням було розгорнуто локальний тестовий стенд.

Фізичною платформою для розгортання як серверної частини, так і пулу імітаційних клієнтів виступила обчислювальна машина, апаратні характеристики якої наведено:

- ноутбук: HP ENVY x360 Convertible 15-es0xxx;
- процесор: 4-ядерний Intel® Core™ i5-1135G7 з базовою частотою 2,40 GHz; турбо – до 4,20 GHz; 8 МБ кешу L3;
- оперативна пам’ять: 16 ГБ DDR4-3200;
- накопичувач: SSD NVMe, 512 ГБ.

Програмний стек тестового середовища було побудовано на базі сучасного дистрибутиву Linux. Детальні специфікації програмного забезпечення наведено:

- операційна система: Fedora 43 Workstation Edition;
- ядро Linux: версія 7.0.9-105.fc43.x86_64.
- система управління БД: PostgreSQL 15
- компілятор: GCC 16

4.2 Розробка імітаційної моделі обладнання

4.2.1 Методика тестування

Для перевірки працездатності спроєктованої архітектури, а також для оцінки її відмовостійкості та продуктивності, було розроблено спеціалізований тестовий стенд. Оскільки залучення реального промислового обладнання для проведення навантажувальних тестів є економічно

недоцільним, тестування проводилося з використанням імітаційного моделювання.

Головним завданням етапу тестування було довести не лише здатність системи збирати дані, але й перевірити стабільність роботи сервера під час паралельної обробки багатьох криптографічних сесій. Для реалізації цього сценарію було створено віртуальне середовище, що імітує роботу цілого заводу.

На стороні сервера у файлі server_config.json було згенеровано та зареєстровано 20 унікальних токенів доступу, що символізують 20 незалежних виробничих цехів. Замість запуску одного клієнтського шлюзу з великою кількістю верстатів, було прийнято рішення розгорнути 20 ізольованих екземплярів розробленого клієнта. Кожен екземпляр отримав власний конфігураційний файл зі своїм токеном та був налаштований на паралельне опитування пулу віртуальних датчиків.

Для масового запуску та оркестрації клієнтів використовувався автоматизований python-скрипт. Після ініціалізації скрипта операційна система створила 20 незалежних фонових процесів, кожен з яких ініціював власне TCP-з'єднання з сервером, виконував процедуру отримання сертифікатів сервера та починав асинхронно відправляти пакети телеметрії.

Щоб максимально наблизити умови тесту до реальних, розмір пакетування на клієнтах був налаштований на генерацію приблизно 20-25 записів телеметрії в одному HTTP-запиті.

Моніторинг тестового стенда здійснювався на трьох рівнях:

1. Системний рівень, на якому проходила фіксація споживання ресурсів центрального процесора та оперативної пам'яті за допомогою системної утиліти btop.

2. На рівні бази даних здійснювався контроль кількості успішно збережених транзакцій та перевірка цілісності даних у таблиці telemetry.

3. На програмному рівні було вбудоване профілювання коду. У вихідний код сервера та клієнтів були інтегровані лічильники часу. З точністю

до мікросекунд вимірявся час, який витрачається на розшифрування контейнера, взаємодію з БД та зворотне шифрування відповіді, ізолювавши ці показники від загальних мережевих затримок.

4.2.2 Аналіз продуктивності

Після успішного розгортання та ініціалізації тестового стенда, автоматизований скрипт ініціював одночасний запуск 20 ізольованих клієнтських шлюзів. Навантажувальний тест тривав безперервно протягом 5 хвилин. За цей час система мала продемонструвати здатність обробляти паралельні потоки зашифрованої телеметрії без втрати пакетів, витоків пам'яті або відмов у з'єднанні.

Аналіз метрик операційної системи продемонстрував високу ефективність обраного технологічного стеку. У стані спокою сервер споживав базовий мінімум оперативної пам'яті. Під час пікового навантаження споживання оперативної пам'яті сервером зафіксувалося на позначці всього 36 МБ і залишалося стабільним без жодних коливань. Кожен клієнтський процес споживав близько 26 МБ.

Навантаження на центральний процесор з боку сервера не перевищувало 2,5%, а з боку клієнтів знаходилося на рівні 0,1%. Такі показники доводять, що використання мови C++ дозволяє системі функціонувати навіть на застарілому промисловому обладнанні або на найдешевших хмарних серверах.

За 5 хвилин безперервного агресивного тестування сервер успішно розшифрував, обробив та зберіг у базу даних PostgreSQL 32 000 записів телеметрії.

4.2.3 Аналіз накладних витрат на криптографію

Головним питанням при впровадженні складних систем кібербезпеки є оцінка накладних витрат, оскільки шифрування може суттєво уповільнити систему. Для перевірки цього було проведено профілювання життєвого циклу

одного запиту. Заміри часу на стороні сервера показали такі усереднені результати:

- Розшифрування сесійного ключа та пакета: ~12–31 мс.
- Взаємодія з PostgreSQL: ~17–39 мс.
- Підпис та шифрування відповіді: ~3–8 мс.

Загальний час повного циклу обробки одного пакету телеметрії на сервері складав у середньому від 49 до 85 мс.

Аналіз логів показує, що криптографічні перетворення займають від 41% до 66% загального часу обробки запиту. Хоча шифрування дійсно створює найбільше навантаження у системі, його абсолютне значення (близько 30-50 мілісекунд) є достатньо малим, що не перешкоджає роботі системи в режимі реального часу. Разом із мережевими затримками, які складали близько 22-38 мс, клієнтський шлюз отримував зашифровану відповідь від сервера менш ніж за 0,1 секунди.

4.3 Інтеграція підсистеми візуалізації

Накопичення великих масивів телеметричних даних у реляційному сховищі під керуванням PostgreSQL є необхідною, але недостатньою умовою для ефективного управління сучасним цифровим виробництвом. Бінарні або текстові дані, що зберігаються в таблицях бази даних, не можуть бути оперативно інтерпретовані людиною безпосередньо в момент їх надходження. Для перетворення цифрової телеметрії на інструмент прийняття управлінських та інженерних рішень необхідна підсистема візуалізації, що працює в режимі реального часу. У рамках даної роботи як таке рішення було інтегровано аналітичну платформу Grafana [21].

4.3.1 Архітектурні принципи взаємодії Grafana з моделлю даних

Інтеграція платформи Grafana з розробленим сервером прийому даних реалізована за допомогою прямого підключення до бази даних через рідний драйвер. Такий підхід усуває потребу в розробці додаткових проміжних

програмних шлюзів. Взаємодія підсистеми візуалізації зі сховищем базується на надсиланні оптимізованих SQL-запитів із періодичністю повного оновлення екрана, яка була налаштована на значення 5 секунд.

Головною особливістю інтеграції є робота з розробленою раніше гібридною структурою таблиці, де фіксовані параметри, такі як мітка часу, ідентифікатор верстата та статус, винесені в окремі реляційні колонки, а динамічний набір метрик датчиків зберігається у вигляді бінарного об'єкта JSONB. Для побудови графіків часових рядів та аналітичних панелей у Grafana було застосовано спеціалізований синтаксис екстракції PostgreSQL.

Для демонстрації можливостей розробленого комплексу в рамках тестового стенда було налаштовано три рівні деталізації моніторингу. Це дозволяє показати здатність системи забезпечувати даними різних користувачів: від загального огляду статусу мережі до детального аналізу конкретної групи обладнання та глибокої діагностики окремої установки.

4.3.2 Дашборд загальної доступності обладнання

Перший рівень візуалізації призначений для вищого технічного персоналу та операторів ситуаційних центрів. Головною метою даного інтерфейсу є миттєва оцінка працездатності всього парку обладнання та локалізація точок виникнення аварійних ситуацій. Дашборд побудовано на комбінації агрегуючої панелі типу Stat та динамічної таблиці Table.

У штатному режимі функціонування підсистема здійснює сканування таблиці телеметрії та виводить великий цифровий індикатор зеленого кольору, що свідчить про повну доступність усіх об'єктів (рис. 4.1). Якщо усе обладнання працює і надсилає данні, то дашборд з аварійними об'єктами не містить ніякої інформації.

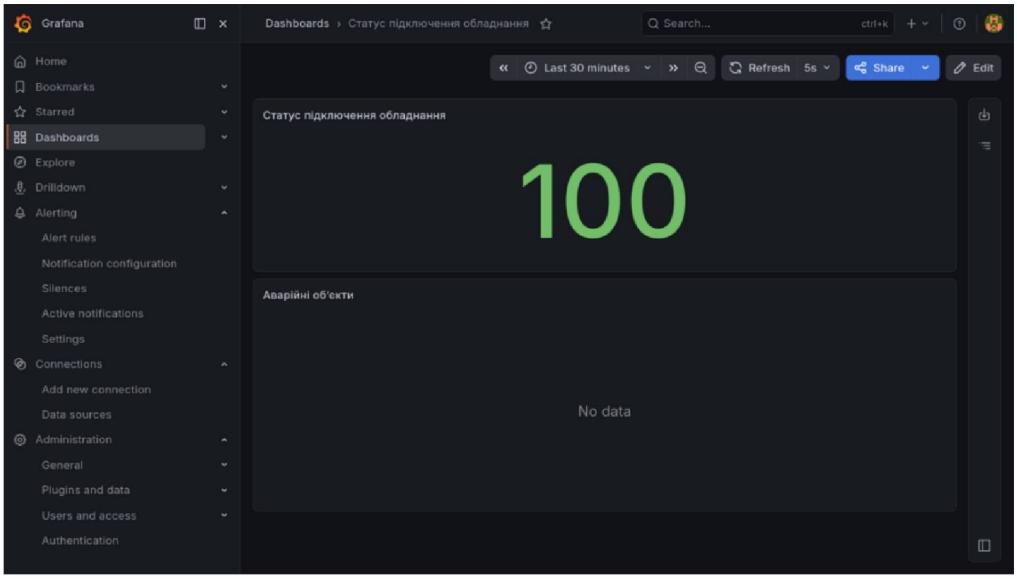


Рисунок 4.1 – Інтерфейс загального статусу за умов штатного функціонування комплексу.

У випадку виникнення апаратного збою, обриву лінії зв'язку або вимкнення живлення на периферійному шлюзі, розроблений клієнт починає надсилати пакети зі статусом OFFLINE. Платформа Grafana реагує на зміну реляційного атрибута в базі даних.

На рисунку 4.2 продемонстровано поведінку системи під час штучно змодельованого збою, внаслідок якого відключилося 5 верстатів однієї з виробничих дільниць.

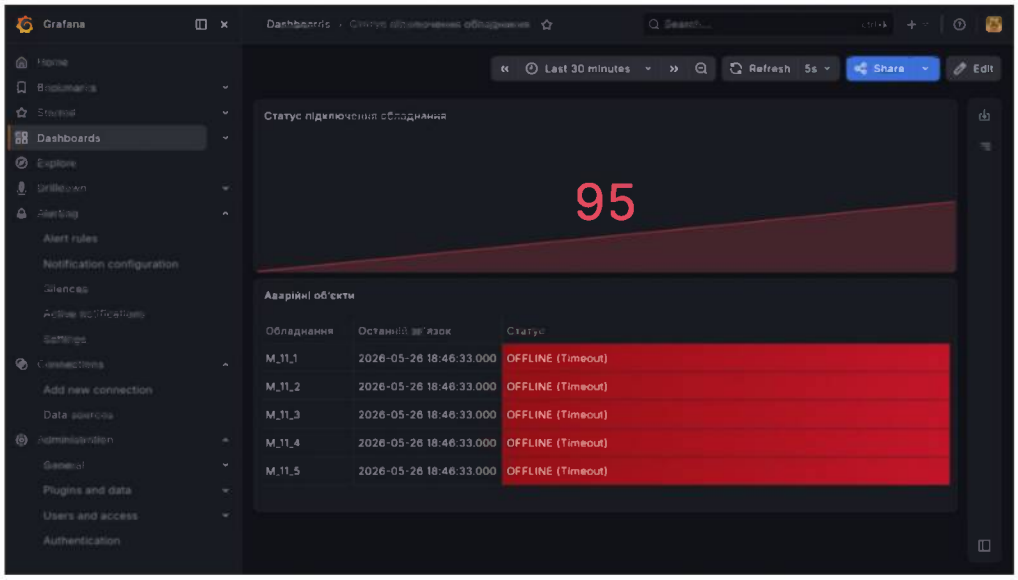


Рисунок 4.2 – Реакція підсистеми візуалізації на виникнення аварійного відключення групи обладнання.

Головний індикатор автоматично перераховує кількість активних вузлів, змінює колір на критичний червоний і виводить актуальне значення (95 активних одиниць). Одночасно з цим активується нижня панель «Аварійні об’єкти». У таблицю виводяться точні системні імена відключених агрегатів, мітка часу останнього успішного сеансу зв’язку та маркер помилки OFFLINE (Timeout).

4.2.3 Організація параметрів групи обладнання

Другий рівень інтерфейсу орієнтований на начальників цехів та технологів, які відповідають за дотримання температурних та швидкісних режимів роботи обладнання. Для цього було розроблено комплексний дашборд «Моніторинг теплового стану вузлів обладнання», який розділяє загальний масив із 100 верстатів на 4 логічні блоки, що відповідають чотирьом основним виробничим дільницям (рис. 4.3).

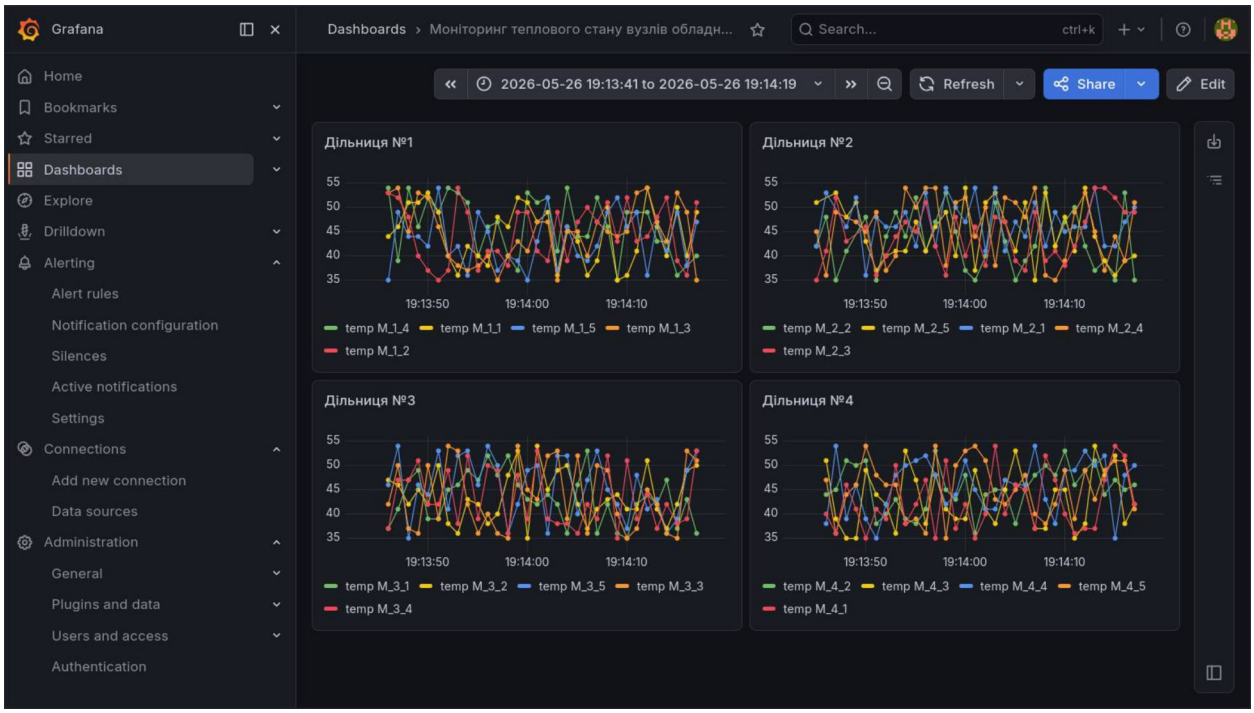


Рисунок 4.3 – Панель моніторингу теплових режимів за виробничими дільницями.

Кожна з чотирьох панелей є графіком часових рядів, на якому одночасно відображаються хронологічні лінії п’яти обраних верстатів відповідної

дільниці. Щільне розміщення графіків дозволяє технологу виконувати візуальний порівняльний аналіз. Наприклад, якщо один із верстатів починає демонструвати стрімке зростання температури, це явно свідчить про знос підшипника шпинделя або дефіцит мастила, що дозволяє реалізувати стратегію прогнозованого технічного обслуговування до моменту фізичного виходу заліза з ладу.

4.2.4 Реалізація цифрової картки одиниці обладнання

Найбільш технологічно складною та інформативною частиною підсистеми візуалізації є індивідуальна інтерактивна картка установки, розроблена для інженерів сервісної діагностики, а також для операторів прецизійних оброблювальних центрів. Цей дашборд демонструє повний цифровий зліпок конкретного вертикально-фрезерного верстата на основі експрес-парсингу складних JSONB-структур (рис. 4.4).



Рисунок 4.4 – Розширена цифрова картка моніторингу вертикально-фрезерного оброблювального центру.

Ергономіка дашборду побудована на комбінації п’яти різних типів графічних компонентів, кожен з яких оптимізовано під конкретний тип фізичної величини:

1. Панель «Головні індикатори» використовує радіальні напівкруглі шкали для відображення критичних швидкоплинних параметрів – обертів шпинделя та температури головного вузла. Шкали мають налаштовані зони безпеки: зелений сектор означає штатну роботу, жовтий – підвищене навантаження, червоний – критичний режим. На рисунку видно поточний стан: безпечні 49 °C та стабільні оберти.

2. Панель «Вібродіагностика» виводить високочастотний графік коливань рівня вібрації (в мм/с). Інженерний аналіз підсилюється червоною лінією уставки на рівні 0.7 мм/с. Якщо амплітуда вібраційного шуму перетинає цей ліміт, це є сигналом про критичний дисбаланс інструменту або руйнування посадкового місця.

3. Панель «Просторове переміщення інструменту» демонструє лінійний графік положення робочого органу по трьох просторових координатах для контролю коректності виконання керуючої програми ЧПК.

4. Панелі «Енергоспоживання та навантаження» та «Виготовлено деталей». Перша відображає інтегральний графік співвідношення споживаного струму в Амперах та загального навантаження у відсотках, що необхідно для енергетичного аудиту цеху. Друга виводить чистий лічильник готової продукції, що можна інтегрувати з системами MES/ERP для обліку темпів виробництва.

5. Панель «Стан допоміжних систем» використовує вертикальні заповнювані шкали для моніторингу параметрів гідравліки та охолодження.

Висновок до розділу 4

У четвертому розділі було проведено тестування розробленої системи в умовах, наближених до реального промислового навантаження. Для цього було розгорнуто локальний тестовий стенд та створено імітаційну модель, яка

генерувала роботу 20 незалежних клієнтських вузлів, до кожного з яких було підключено 5 одиниць обладнання. Результати навантажувального тестування підтвердили високу продуктивність та ефективність обраних архітектурних рішень: за п’ять хвилин безперервної роботи сервер успішно обробив і зберіг 32000 записів телеметрії без втрат даних. При цьому споживання ресурсів дуже мале: 36 МБ оперативної пам’яті сервера та 26 МБ клієнта, а навантаження на процесор – 1,5-2,5%.

Окремо було проведено аналіз накладних витрат на криптографічні перетворення. Було з’ясовано, що процеси шифрування, розшифрування та накладання електронного підпису займають від 41% до 66% загального часу обробки. Проте в абсолютному вимірі ці операції потребують лише 30–50 мілісекунд, що абсолютно не перешкоджає роботі системи в режимі реального часу. Такі показники доводять, що інтеграція державних криптографічних стандартів дозволяє клієнтським шлюзам отримувати зворотну відповідь від сервера менш ніж за 0,1 секунди.

Також було успішно інтегровано підсистему візуалізації на базі аналітичної платформи Grafana, яка взаємодіє з гібридною моделлю даних PostgreSQL за допомогою запитів до бінарного формату JSONB. Розроблено трирівневий диспетчерський інтерфейс: від загального дашборду доступності всього парку обладнання до глибокої діагностичної картки окремої технологічної установки. Завдяки підсистемі візуалізації зібрані телеметричні дані перетворюються на зручний інструмент для прийняття рішень, оперативного виявлення аварій та впровадження стратегій прогнозованого технічного обслуговування.

ВИСНОВКИ

У кваліфікаційній роботі було успішно вирішено актуальне науково-практичне завдання, яке полягає у розробці комплексного програмного рішення для підключення вразливого промислового обладнання до сучасної цифрової інфраструктури. На першому етапі дослідження, відповідно до плану роботи, було проведено аналіз предметної області та існуючих рішень дистанційного моніторингу. Цей аналіз виявив ключову проблему: переважна більшість сучасних і застарілих технологічних установок функціонують за відкритим протоколом Modbus TCP, який не має вбудованих механізмів автентифікації чи захисту від перехоплення даних. Хмарні платформи та популярні відкриті рішення, хоч і пропонують потужну аналітику, не здатні забезпечити цифрову ізоляцію цеху та економне використання ресурсів локальних комп'ютерів. Для вирішення цієї проблеми було спроєктовано загальну архітектуру клієнт-серверної системи, яка базується на парадигмі периферійних обчислень. Система отримала чіткий розподіл на три рівні: рівень збору даних безпосередньо в цеху, рівень захищеного транспортування через недовірені мережі та рівень централізованого збереження і візуалізації.

Переходячи до практичної реалізації системи, було виконано завдання з програмування алгоритмів збору та первинної обробки телеметричної інформації. Розроблений на мові C++ периферійний шлюз дозволив реалізувати локальний паралельний збір даних з багатьох установок одночасно. Завдяки реалізованому алгоритму буферизації та пакетування, програма здатна відфільтровувати зайвий трафік, формувати структуровані JSON-пакети та ефективно працювати навіть на малопотужних промислових комп'ютерах. У разі втрати зв'язку з окремим агрегатом система не зупиняє роботу, а коректно фіксує аварійний статус, що забезпечує високу відмовостійкість клієнтської частини.

Важливим етапом роботи розробки комплексу, стало проєктування алгоритмів шифрування та безпечної передачі даних. Усвідомлюючи ризики

промислового шпигунства та несанкціонованого втручання в роботу критичної інфраструктури, було впроваджено механізм інкапсуляції токенів авторизації всередину корисного навантаження. Проте головною перевагою розробленої системи над абсолютністю більшістю існуючих комерційних та відкритих альтернатив є відмова від загальноприйнятих міжнародних бібліотек на користь реалізації українських державних криптографічних стандартів. Для цього була використана бібліотека EUSign. На відміну від іноземних платформ моніторингу, розроблений програмний комплекс має технічну та правову базу для проходження державної експертизи та отримання атестата відповідності КСЗІ. Це відкриває можливість впровадження системи на підприємствах військово-промислового комплексу, об'єктах енергетики та інших структурах критичної інфраструктури.

Для завершення побудови повного циклу обробки даних було розроблено серверну частину та структуру бази даних. Сервер прийому, побудований за REST API архітектурою без збереження стану, здатний паралельно розшифровувати та верифікувати сотні вхідних потоків в оперативній пам'яті. Збереження телеметрії організовано на базі реляційної СКБД PostgreSQL із використанням бінарного формату JSONB. Цей гібридний підхід забезпечив системі гнучкість: структуровані пошукові індекси гарантують швидкість вибірки, а JSON-формат дозволяє зберігати будь-які специфічні показники датчиків без необхідності переробляти схему бази даних. Дані інтегруються з підсистемою візуалізації Grafana, де розроблено три рівні диспетчерського контролю: від глобальної оцінки доступності цехів до моніторингу показників окремої установки у режимі реального часу.

На фінальному етапі роботи було виконано тестування створених компонентів та всієї системи під навантаженням. Оскільки використання реального парку обладнання було неможливим, було розроблено імітаційну модель, що складалася з 20 ізольованих клієнтських вузлів, які генерували паралельні потоки зашифрованої телеметрії. Результати тестування повністю підтвердили ефективність закладених архітектурних рішень. За п'ять хвилин

безперервного опитування система успішно обробила та зберегла 32 000 записів без жодної втрати пакетів. При цьому профілювання коду довело, що використання важких криптографічних перетворень займає лише 30-50 мілісекунд, що абсолютно не перешкоджає роботі системи в реальному часі. Навантаження на центральний сервер виявилося мінімальним – споживання оперативної пам’яті не перевищило 36 МБ.

Підсумовуючи результати виконання кваліфікаційної роботи, можна стверджувати, що всі поставлені завдання були виконані в повному обсязі. Створений програмно-апаратний комплекс є готовим до впровадження продуктом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Mobley R. K. Introduction to Predictive Maintenance. Elsevier Science & Technology Books, 2002. 437 p.

2) Edge Computing: Vision and Challenges / W. Shi et al. IEEE Internet of Things Journal. 2016. Vol. 3, no. 5. P. 637–646.

3) SIMATIC WinCC. Siemens. [Електронний ресурс]. – режим доступу: URL: https://www.siemens.com/en-us/products/simatic-hmi/wincc-unified-software/ (дата звернення – 16.10.2025).

4) AWS IoT Core Documentation. Amazon Web Services. [Електронний ресурс]. – режим доступу: URL: https://docs.aws.amazon.com/iot/ (дата звернення – 16.10.2025).

5) Node-RED: Low-code programming for event-driven applications. Node-RED. [Електронний ресурс]. – режим доступу: URL: https://nodered.org/ (дата звернення – 18.10.2025).

6) Modbus Application Protocol Specification V1.1b3. Modbus Organization, Inc., 2012. 50 p.

7) Swales A. Open Modbus/TCP Specification. Release 1.0. Schneider Electric, 1999. 26 p.

8) Modbus/DNP3 State-Based Intrusion Detection System / I. N. Fovino et al. 2010 IEEE 24th International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, WA, 20–23 April 2010. 2010. P. 729–736.

9) Anatomy of Threats to the Internet of Things / I. Makhdoom et al. IEEE Communications Surveys & Tutorials. 2019. Vol. 21, no. 2. P. 1636–1675.

10) ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист інформації. Електронний цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевіряння. Держспоживстандарт України, 2003. 37 с.

11) ДСТУ ISO/IEC 11770-3:2023. Інформаційні технології. Методи захисту. Керування ключами. Частина 3. Механізми з використанням асиметричних методів. ДП «УкрНДНЦ», 2023.

12) Housley R. Cryptographic Message Syntax (CMS). RFC Editor, 2009 [Електронний ресурс]. – режим доступу: URL: https://datatracker.ietf.org/doc/html/rfc5652/ (дата звернення – 05.11.2025).

13) Технічні специфікації до RFC 5652 : затверджено Наказом Адміністрації Державної служби спеціального зв'язку та захисту інформації України від 27.10.2020 № 687 [Електронний ресурс]. – режим доступу: URL: https://zakon.rada.gov.ua/laws/show/z1273-20#n4/ (дата звернення – 05.11.2025).

14) Jensen S. K., Pedersen T. B., Thomsen C. Time Series Management Systems: A Survey. IEEE Transactions on Knowledge and Data Engineering. 2017. Vol. 29, no. 11. P. 2581–2600.

15) Stroustrup B. The C++ Programming Language. 4th ed. Addison-Wesley Professional, 2013. 1366 p.

16) Hirose Y. cpp-httplib: A C++ header-only HTTP/HTTPS server and client library [Електронний ресурс]. – режим доступу: URL: https://github.com/yhirose/cpp-httplib/ (дата звернення – 10.12.2025).

17) Lohmann N. JSON for Modern C++ [Електронний ресурс]. – режим доступу: URL: https://github.com/nlohmann/json/ (дата звернення – 15.12.2025).

18) PostgreSQL 15 Documentation: JSON Types [Електронний ресурс]. – режим доступу: URL: https://www.postgresql.org/docs/15/datatype-json.html (дата звернення – 25.12.2025).

19) Користувач центру сертифікації ключів. Бібліотека підпису (ОС Microsoft Windows, інтерфейс програмування C++) [Електронний ресурс]. – режим доступу: URL: https://iit.com.ua/download/productfiles/EUSignMSWCPPAppendixA.doc (дата звернення – 07.01.2026).

20) Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Ph.D. dissertation. University of California, Irvine, 2000. 162 p.

21) Grafana Documentation: Fundamentals [Електронний ресурс]. – режим доступу: URL: https://grafana.com/docs/grafana/latest/ (дата звернення – 18.04.2026).

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача <u>першого (бакалаврського)</u> рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

<u>БАРБАШИНА ІЛЛІ ВЛАДИСЛАВОВИЧА</u>
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **<u>КЛІЄНТ-СЕРВЕРНА СИСТЕМА ЗАХИЩЕНОГО ДИСТАНЦІЙНОГО МОНІТОРИНГУ ПРОМИСЛОВОГО ОБЛАДНАННЯ</u>**
керівник роботи **<u>Рева Сергій Миколайович</u>, к.т.н., доцент,**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року***.
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) Технічні специфікації та архітектурні особливості мережевого протоколу Modbus TCP.
   2) Державні криптографічні стандарти України та документація до бібліотеки EUSign.
   3) Документація до системи керування базами даних PostgreSQL та платформи візуалізації Grafana.
   4) Технічні параметри та режими роботи промислового обладнання для побудови імітаційної моделі.
4. Перелік питань, які потрібно розробити:
   1) Проведення аналізу предметної області та розробка загальної архітектури клієнт-серверної системи для збору даних з промислового обладнання.
   2) Проєктування та програмна реалізація алгоритмів збору, шифрування, безпечної передачі та зберігання телеметричної інформації між вузлами системи.
   3) Тестування створеного окремих компонентів програмного забезпечення та всієї системи в цілому.





Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу «КЛІЄНТ-СЕРВЕРНА СИСТЕМА ЗАХИЩЕНОГО ДИСТАНЦІЙНОГО МОНІТОРИНГУ ПРОМИСЛОВОГО ОБЛАДНАННЯ»

1.	Введення	1.1. Назва: Клієнт-серверна система захищеного дистанційного моніторингу промислового обладнання. 1.2. Галузь застосування: Інформаційні технології, комп'ютерна інженерія, системи промислового інтернету речей.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 «Комп'ютерна інженерія». 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/1245 від 28 травня 2026 року (Додаток А до пояснювальної записки).
3.	Призначення розробки	3.1. Мета розробки: проєктування та розробка клієнт-серверної системи для збору, первинної обробки, збереження та візуалізації даних від промислового обладнання з вбудованими механізмами криптографічного захисту. 3.2. Призначення розробки надає можливість: опитувати верстати за протоколом Modbus TCP, оптимізувати обсяги телеметрії за допомогою

		периферійних обчислень, захищати дані при передачі через недовірені мережі за держстандартами, надійно зберігати дані у БД та візуалізувати їх у реальному часі. 3.3. Вихідні дані розробки: специфікації протоколу Modbus TCP, державні криптографічні стандарти та документація до EUSign, документація до СКБД PostgreSQL та платформи Grafana.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: багатопотоковий збір даних, нормалізація та пакетування у JSON, генерація контрольних сигналів, накладання електронного підпису та шифрування на сесійних ключах, серверна обробка REST API запитів без збереження стану, запис JSONB у БД. 4.2. Вимоги до надійності: відмовостійкість шлюзу при втраті з'єднання з окремим обладнанням, ізоляція процесів на сервері та знищення криптографічних сесій при помилках. 4.3. Вимоги до умов експлуатації: система повинна функціонувати в автономному режимі 24/7 без втручання людини. 4.4. Вимоги до складу і параметрів технічних засобів: Клієнт: малопотужний промисловий ПК (ОЗП від 512 МБ). Сервер: багатоядерний процесор, ОЗП від 2 ГБ, швидкий SSD-накопичувач. 4.5. Вимоги до інформаційної та програмної сумісності: ОС Linux або Windows, компілятор GCC, СКБД PostgreSQL версії 15+, інтеграція з Grafana.

		4.6. Вимоги до маркування та упаковки: Вимоги не пред'являються. 4.7. Вимоги до транспортування і зберігання: Вимоги не пред'являються. 4.8. Спеціальні вимоги: криптографічний захист має реалізовуватись виключно з використанням сертифікованих в Україні бібліотек для відповідності вимогам КСЗІ.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Клієнт-серверна система захищеного дистанційного моніторингу промислового обладнання» вважати: 1) Справжнє Технічне завдання на розробку виробу (Додаток Б). 2) Опис спроєктованої архітектури та структури бази даних (Розділ 2 пояснювальної записки). 3) Опис практичної реалізації програмного виробу (Розділ 3 пояснювальної записки). 4) Інструкцію та опис результатів тестування комплексу (Розділ 4 пояснювальної записки).
6.	Вимоги до техніко-економічних показників	Економічна ефективність системи досягається за рахунок: 1) Мінімізації витрат на ліцензування комерційних SCADA-систем. 2) Зниження ризиків аварійних простоїв завдяки переходу до прогнозованого технічного обслуговування. 3) Економії на апаратному забезпеченні внаслідок використання оптимізованого коду з низьким споживанням оперативної пам'яті та ресурсів ЦП.

7.	Стадії і етапи розробки	14.10.2025 – 28.10.2025	Аналіз існуючих рішень дистанційного моніторингу та формування вимог до системи.
		29.10.2025 – 29.11.2025	Проєктування архітектури клієнт-серверної взаємодії та структури бази даних.
		30.11.2025 – 27.02.2026	Розробка серверної частини системи та захищеного API.
		28.02.2026 – 29.04.2026	Програмування клієнтської частини для обробки показників, зчитаних з датчиків та їх шифрування.
		30.04.2026 – 07.05.2026	Тестування системи та її відлагодження.
		08.05.2026 – 22.05.2026	Оформлення пояснювальної записки.
		23.05.2026 – 30.05.2026	Підготовка презентації та доповіді.
8.	Порядок контролю і приймання програмного продукту	1) Контроль виконання згідно календарного плану. 2) Захист на засіданні Екзаменаційної комісії. 3) Подання пояснювальної записки у паперовому та електронному вигляді.	

Виконавець
студент групи КІ41
БАРБАШИН І.В.
(підпис)

Замовник
к.т.н., доцент
РЕВА С.М.
(підпис)

Додаток В

Програма і методика випробувань програмного виробу «КЛІЄНТ-СЕРВЕРНА СИСТЕМА ЗАХИЩЕНОГО ДИСТАНЦІЙНОГО МОНІТОРИНГУ ПРОМИСЛОВОГО ОБЛАДНАННЯ»

1. Об'єкт випробувань

1. Назва програмного виробу: «Клієнт-серверна система захищеного дистанційного моніторингу промислового обладнання»

2. Галузь застосування: Інформаційні технології, комп'ютерна інженерія, системи промислового інтернету речей.

3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям у технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи), а також оцінка її відмовостійкості, захищеності та продуктивності під навантаженням.

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі локального тестового стенда Виконавця.

3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі, відповідному цій програмі і методиці випробувань.

4. Організації, які беруть участь у випробуваннях

Випробування програмного виробу проводяться Виконавцем за безпосередньої участі Замовника (керівника кваліфікаційної роботи). Аналіз результатів випробувань та прийняття рішення щодо відповідності виробу технічному завданню здійснюється Екзаменаційною комісією під час захисту кваліфікаційної роботи.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1. працювати в середовищі дистрибутиву на базі ядра Linux (для серверної частини) та в середовищі ОС Windows або Linux (для клієнтського шлюзу) та підтримувати доступ з ОС Windows/Linux/MacOS через веб-інтерфейс візуалізації;
2. забезпечувати високу надійність: безперебійну роботу 24/7 та відмовостійкість при втраті з'єднання з окремими одиницями обладнання;
3. передбачити захист даних від несанкціонованого доступу;
4. забезпечувати сумісність з СКБД PostgreSQL та платформою візуалізації Grafana;
5. мати високу продуктивність при мінімальному споживанні оперативної пам'яті;
6. вимоги до маркування та упаковки (не висуваються);
7. вимоги до транспортування і зберігання (не висуваються);
8. спеціальні вимоги (не висуваються).

5. Вимоги до програмної документації

Програмною документацією до розробленого програмного продукту вважати:

1. Справжнє Технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки);

2. Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки);

3. опис архітектури, реалізації та результатів тестування комплексу (представити в Розділах 2, 3 та 4 пояснювальної записки);

4. текст програми (представити як Додаток Г до пояснювальної записки).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Для проведення випробувань необхідний тестовий комп'ютер (ноутбук/сервер) під керуванням ОС Linux, компілятор мови C++ (GCC), налаштована СКБД PostgreSQL, аналітична платформа Grafana, а також Python-скрипт для масового запуску імітаційних шлюзів обладнання.

6.2. Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);
- випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе перевірку комплектності програмної документації на наявність зазначеної в ТЗ документації та перевірку якості за стандартами ЄСПД.

Перелік перевірок на 2 етапі випробувань включає перевірку відповідності технічних характеристик та функціональних вимог системи заявленим в ТЗ.

Порядок запуску для проведення випробувань:

1. Запуск служби бази даних PostgreSQL та ініціалізація структури таблиць телеметрії.

2. Запуск скомпільованого бінарного файлу центрального сервера прийому даних у терміналі.

3. Запуск автоматизованого скрипта імітації для старту 20 незалежних процесів клієнтських шлюзів.

4. Перехід у веб-браузері за адресою: http://localhost:3000 для моніторингу результатів у реальному часі через інтерфейс Grafana.

Для проведення випробувань пропонується Тест 1, Тест 2 та Тест 3.

Тест 1. Перевірка продуктивності та успішної обробки телеметрії

Методика: Одночасний запуск 20 імітаційних клієнтських вузлів, які безперервно відправляють зашифровану телеметрію протягом 5 хвилин.

Очікуваний результат: Сервер приймає, успішно розшифровує пакети та записує їх у базу даних. Споживання ОЗП сервера стабільне на рівні ~36 МБ. У СКБД PostgreSQL зафіксовано 32000 успішно збережених записів у колонках типу JSONB.

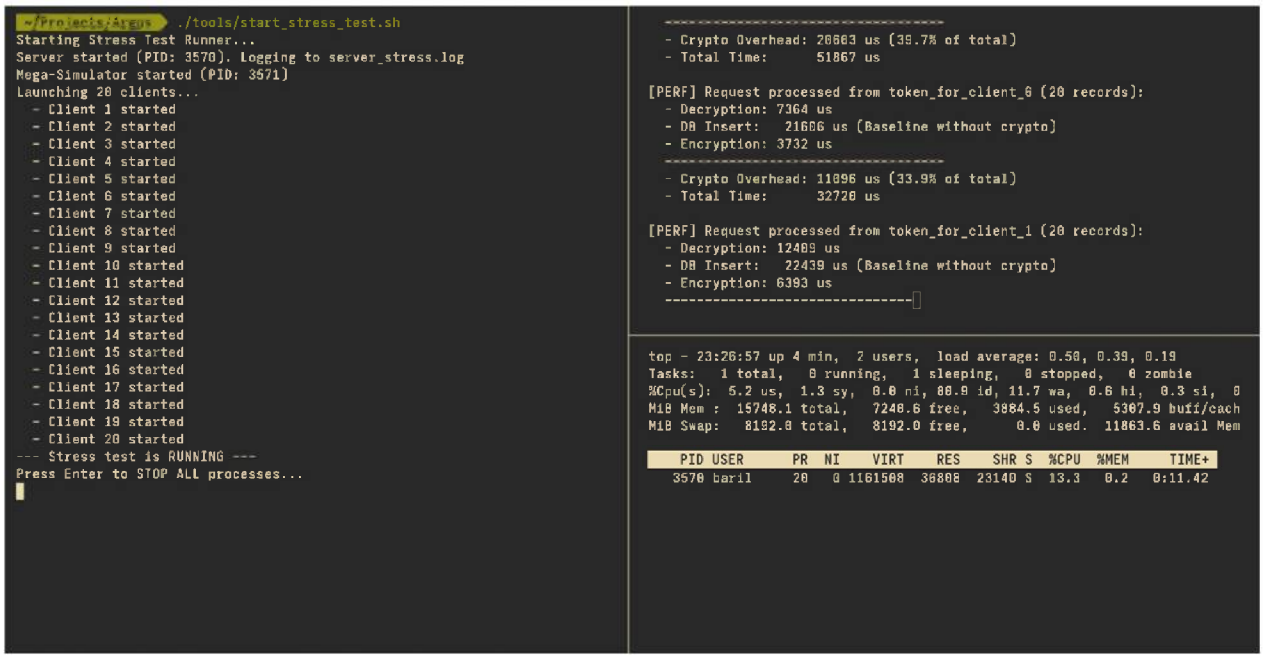


Рисунок В.1 – Тест 1

Тест 2: Перевірка відмовостійкості системи

Методика: Примусове вимкнення 5 віртуальних установок на одній з імітаційних виробничих дільниць.

Очікуваний результат: Шлюз продовжує збирати дані з іншого обладнання. Для вимкнених верстатів у пакеті явно генерується статус “OFFLINE”. У панелі Grafana головний індикатор змінює колір на червоний,

кількість активних вузлів падає до 95, а в таблиці аварійних об’єктів з’являються записи з маркуванням OFFLINE (Timeout).

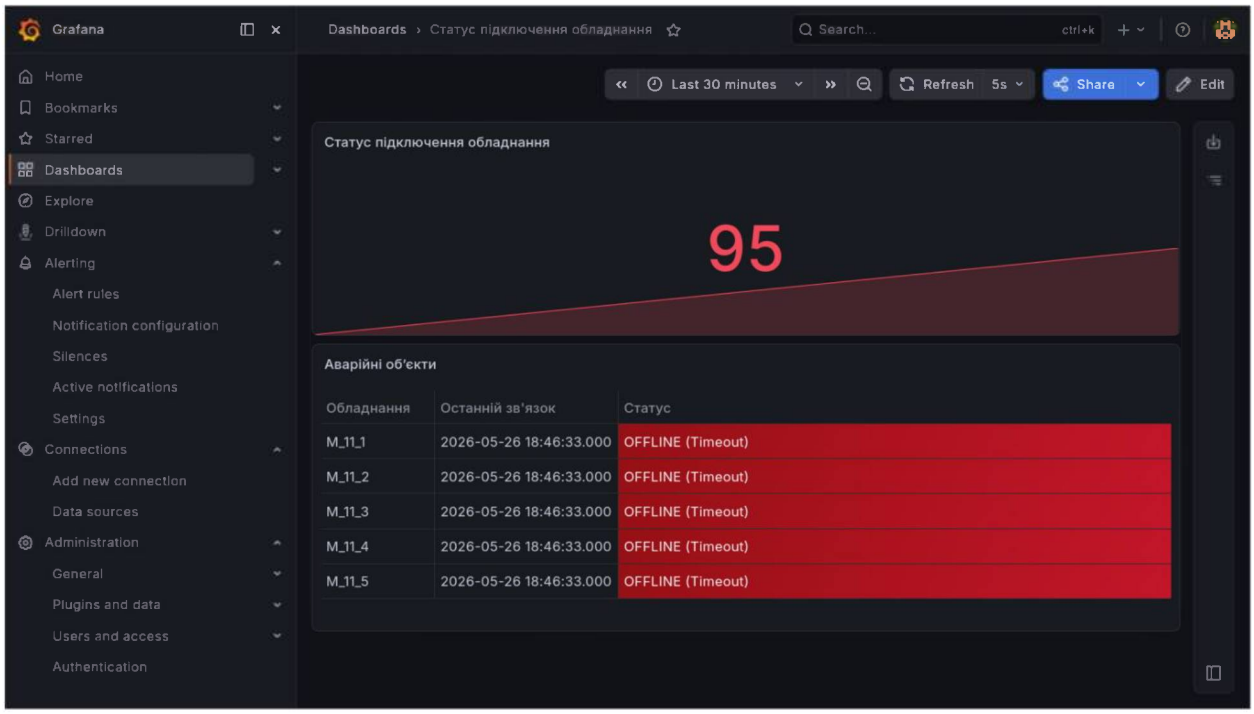


Рисунок В.2 – Тест 2

Тест 3: Перевірка криптографічного захисту та авторизації

Методика: Спроба відправки зашифрованого пакета даних з ідентифікатором клієнта, якого немає у дозволеному server_config.json, або відправка пошкодженого криптографічного контейнера.

Очікуваний результат: Сервер відхиляє з’єднання. Валідація токена не проходить. Динамічна сесія знищується без витоків пам’яті. Дані не записуються в базу PostgreSQL, а клієнт отримує помилку у вигляді HTTP-статусу відмови.

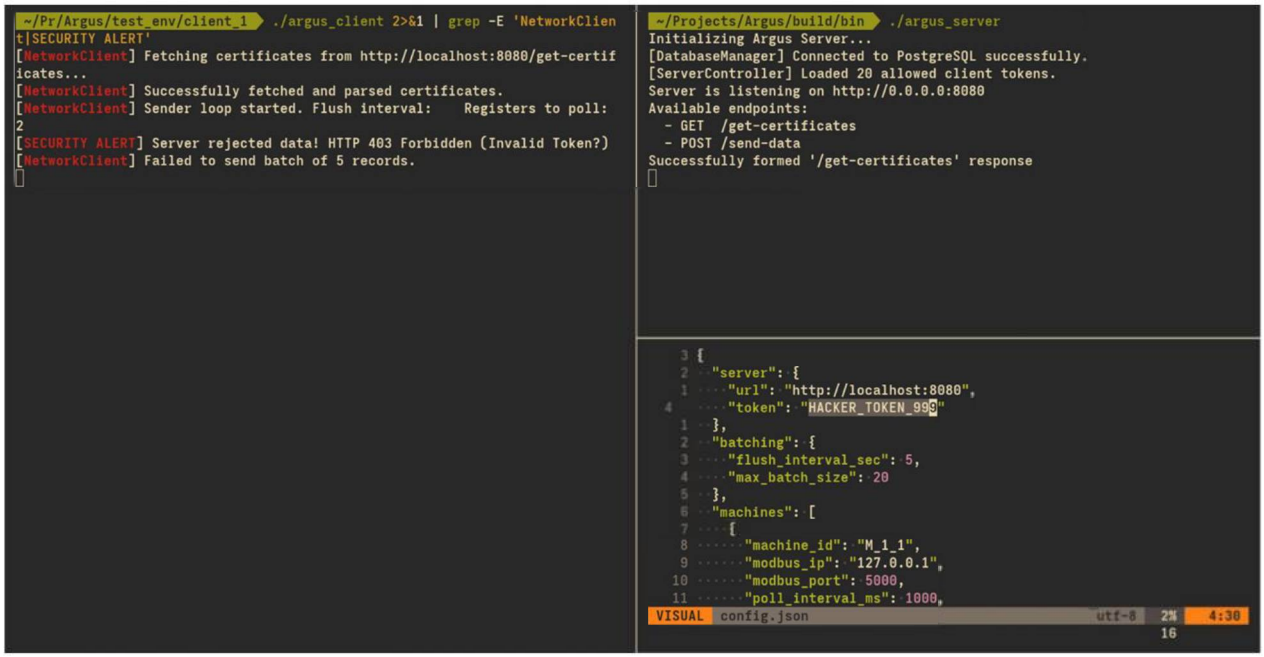


Рисунок В.3 – Тест 3

Висновки: Тест вважається пройденим, якщо відбуваються вказані операції і їх відповідне відображення у програмному продукті та підсистемі візуалізації. Тест 1 успішно пройшов випробування, Тест 2 успішно пройшов випробування і Тест 3 успішно пройшов випробування. Випробування системи пройшло успішно, вимоги технічного завдання виконані в повному обсязі.

Виконавець: студент групи КІ41, Барбашин І. В. _____
(підпис)

Додаток Г

Вихідний код периферійного шлюзу

Файл ConfigManager.h

```
#ifndef CONFIG_MANAGER_H
#define CONFIG_MANAGER_H

#include <cstdint>
#include <string>
#include <vector>

struct RegisterConfig {
  std::string name;
  uint16_t address;
  std::string type;
};

struct MachineConfig {
  std::string machine_id;
  std::string modbus_ip;
  int modbus_port;
  int poll_interval_ms;
  int heartbeat_interval_sec;
  std::vector<RegisterConfig> registers;
};

struct BatchingConfig {
  int flush_interval_sec;
  int max_batch_size;
};

struct ServerConfig {
  std::string url;
  std::string token;
  BatchingConfig batching;
};

class ConfigManager {
public:
  static ConfigManager &getInstance() {
    static ConfigManager instance;
    return instance;
  }

  bool loadConfig(const std::string &filepath);

  const ServerConfig &getServerConfig() const { return
m_serverConfig; }
  const std::vector<MachineConfig> &getMachines() const { return
m_machines; }

private:
```

```
  ConfigManager() = default;

  ServerConfig m_serverConfig;
  std::vector<MachineConfig> m_machines;
};

#endif // CONFIG_MANAGER_H
```

Файл ConfigManager.cpp

```
#include "ConfigManager.h"
#include <fstream>
#include <iostream>
#include <nlohmann/json.hpp>

using json = nlohmann::json;

bool ConfigManager::loadConfig(const std::string &filepath) {
  try {
    std::ifstream file(filepath);
    if (!file.is_open()) {
      std::cerr << "Failed to open config file: " << filepath <<
std::endl;
      return false;
    }

    json j;
    file >> j;

    m_serverConfig.url =
j.at("server").at("url").get<std::string>();
    m_serverConfig.token = j.at("server").value("token", "");

    m_serverConfig.batching.flush_interval_sec =
        j.value("batching",
json::object()).value("flush_interval_sec", 10);
    m_serverConfig.batching.max_batch_size =
        j.value("batching",
json::object()).value("max_batch_size", 50);

    m_machines.clear();
    for (const auto &machineJson : j.at("machines")) {
      MachineConfig machine;
      machine.machine_id =
machineJson.at("machine_id").get<std::string>();
      machine.modbus_ip =
machineJson.at("modbus_ip").get<std::string>();
      machine.modbus_port =
machineJson.at("modbus_port").get<int>();
      machine.poll_interval_ms =
machineJson.at("poll_interval_ms").get<int>();
```

```
      machine.heartbeat_interval_sec =
          machineJson.value("heartbeat_interval_sec", 60);

      for (const auto &regJson : machineJson.at("registers")) {
        RegisterConfig reg;
        reg.name = regJson.at("name").get<std::string>();
        reg.address = regJson.at("address").get<uint16_t>();
        reg.type = regJson.at("type").get<std::string>();
        machine.registers.push_back(reg);
      }
      m_machines.push_back(machine);
    }

    return true;
  } catch (const json::exception &e) {
    std::cerr << "JSON Parsing error in config: " << e.what() <<
std::endl;
    return false;
  } catch (const std::exception &e) {
    std::cerr << "General error loading config: " << e.what() <<
std::endl;
    return false;
  }
}
```

Файл EUSignInterface.h

```
#ifndef EU_SIGN_INTERFACE_H
#define EU_SIGN_INTERFACE_H

#include <cstdint>
#include <vector>

bool EUSignInitialize();

void EUSignFinalize();

void EUSignFreeMemory(uint8_t *pbMemory);

bool EUSignClientCreateDynamicSession(uint8_t *pbServerKEPCert,
                                      unsigned long
dwServerKEPCertLength,
                                      void **ppvClientSession,
                                      uint8_t **ppbAuthData,
                                      unsigned long
*pdwAuthDataLength);

void EUSignDestroyDynamicSession(void *pvClientSession);

bool EUSignEncryptSessionData(void *pvSession, uint8_t *pbData,
                              unsigned long dwDataLength,
                              uint8_t **ppbEncryptedData,
```

```
                                unsigned long
*pdwEncryptedDataLength);

bool EUSignDecryptSessionData(void *pvSession, uint8_t
*pbEncryptedData,
                                unsigned long
dwEncryptedDataLength,
                                uint8_t **ppbDecryptedData,
                                unsigned long
*pdwDecryptedDataLength);

bool ParseAndAssignCertificates(const std::vector<uint8_t>
&cert1,
                                  const std::vector<uint8_t>
&cert2,
                                  std::vector<uint8_t> &outDsCert,
                                  std::vector<uint8_t>
&outKepCert);

bool EUSignVerifyDataInternal(uint8_t *pbSignedData,
                                unsigned long dwSignedDataLength,
                                uint8_t **ppbData, unsigned long
*pdwDataLength);

#endif // EU_SIGN_INTERFACE_H
```

Файл EUSignInterface.cpp

```
#include "EUSignInterface.h"
#include "EUSignCP.h"

#include <cstring>
#include <iconv.h>
#include <iostream>
#include <vector>

#define SESSION_EXPIRE_TIME_SEC 3600

#define ALGORITHM_NAME_DSTU4145 "DSTU4145"

typedef struct _EU_SIGN_STATE {
  uint32_t dwGryadaTypeIndex;
  uint32_t dwGryadaDeviceIndex;

  char *pszDSServerCertificateIssuer;
  char *pszDSServerCertificateSerial;
  char *pszKEPServerCertificateIssuer;
  char *pszKEPServerCertificateSerial;

  uint8_t *pbCMPCertificate;
  uint32_t dwCMPCertificateLength;
```

```
  void *pvKEPPrivateKeyContext;
} EU_SIGN_STATE, *PEU_SIGN_STATE;

static PEU_INTERFACE s_pEUSignInterface = NULL;
static PEU_SIGN_STATE s_pEUSignState = NULL;

static bool InitializeLibrary() {
  if (!EULoad())
    return false;

  s_pEUSignInterface = EUGetInterface();
  if (s_pEUSignInterface == NULL)
    return false;

  s_pEUSignInterface->SetUIMode(false);

  uint32_t dwError = s_pEUSignInterface->Initialize();
  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

bool ParseAndAssignCertificates(const std::vector<uint8_t>
&cert1,
                                const std::vector<uint8_t>
&cert2,
                                std::vector<uint8_t> &outDsCert,
                                std::vector<uint8_t>
&outKepCert) {
  const std::vector<uint8_t> *certs[2] = {&cert1, &cert2};

  bool foundDS = false;
  bool foundKEP = false;

  for (int i = 0; i < 2; ++i) {
    PEU_CERT_INFO_EX pCertificateInfo = nullptr;

    uint32_t dwError = s_pEUSignInterface->ParseCertificateEx(
        (unsigned char *)certs[i]->data(), certs[i]->size(),
&pCertificateInfo);

    if (dwError != EU_ERROR_NONE) {
      std::cerr << "Failed to parse certificate " << i << ".
Error: " << dwError
                << std::endl;
      return false;
    }

    if (pCertificateInfo->dwKeyUsage &
EU_KEY_USAGE_DIGITAL_SIGNATURE) {
      outDsCert = *certs[i];
      foundDS = true;
```

```
    } else if (pCertificateInfo->dwKeyUsage &
EU_KEY_USAGE_KEY_AGREEMENT) {
      outKepCert = *certs[i];
      foundKEP = true;
    }

    s_pEUSignInterface->FreeCertificateInfoEx(pCertificateInfo);
  }

  if (!foundDS || !foundKEP) {
    std::cerr
        << "Could not find both DS and KEP certificates in the
provided list."
        << std::endl;
    return false;
  }

  return true;
}

bool EUSignInitialize() {
  if (s_pEUSignInterface != NULL)
    return true;

  if (!InitializeLibrary())
    return false;

  s_pEUSignState = new EU_SIGN_STATE();
  if (s_pEUSignState == NULL)
    return false;

  memset(s_pEUSignState, 0, sizeof(EU_SIGN_STATE));

  return true;
}

void EUSignFinalize() {
  if (s_pEUSignState != NULL) {

    if (s_pEUSignInterface != NULL) {
      if (s_pEUSignState->pvKEPPrivateKeyContext != NULL) {
        s_pEUSignInterface->CtxFreePrivateKey(
            s_pEUSignState->pvKEPPrivateKeyContext);
      }
    }

    if (s_pEUSignState->pszDSServerCertificateIssuer != NULL)
      delete[] s_pEUSignState->pszDSServerCertificateIssuer;

    if (s_pEUSignState->pszDSServerCertificateSerial != NULL)
      delete[] s_pEUSignState->pszDSServerCertificateSerial;

    if (s_pEUSignState->pszKEPServerCertificateIssuer != NULL)
```

```
      delete[] s_pEUSignState->pszKEPServerCertificateIssuer;

    if (s_pEUSignState->pszKEPServerCertificateSerial != NULL)
      delete[] s_pEUSignState->pszKEPServerCertificateSerial;

    delete[] s_pEUSignState->pbCMPCertificate;

    delete s_pEUSignState;
    s_pEUSignState = NULL;
  }

  if (s_pEUSignInterface != NULL) {
    s_pEUSignInterface->Finalize();
    EUUnload();
    s_pEUSignInterface = NULL;
  }
}

void EUSignFreeMemory(uint8_t *pbMemory) {
  if (s_pEUSignInterface == NULL || pbMemory == NULL)
    return;

  s_pEUSignInterface->FreeMemory(pbMemory);
}

bool EUSignClientCreateDynamicSession(uint8_t *pbServerKEPCert,
                                      unsigned long
dwServerKEPCertLength,
                                      void **ppvClientSession,
                                      uint8_t **ppbAuthData,
                                      unsigned long
*pdwAuthDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;

  if (pbServerKEPCert == NULL || dwServerKEPCertLength == 0 ||
      ppvClientSession == NULL || ppbAuthData == NULL ||
      pdwAuthDataLength == NULL) {
    return false;
  }

  *ppvClientSession = NULL;
  *ppbAuthData = NULL;
  *pdwAuthDataLength = 0;

  uint32_t dwError = s_pEUSignInterface-
>ClientDynamicKeySessionCreate(
      SESSION_EXPIRE_TIME_SEC, NULL, NULL, pbServerKEPCert,
      dwServerKEPCertLength, ppvClientSession, ppbAuthData,
pdwAuthDataLength);

  if (dwError != EU_ERROR_NONE) {
```

```
    std::cerr << "ClientDynamicKeySessionCreate failed with
error: " << dwError
              << std::endl;
    return false;
  }

  return true;
}

void EUSignDestroyDynamicSession(void *pvClientSession) {
  if (s_pEUSignInterface == NULL || pvClientSession == NULL)
    return;

  s_pEUSignInterface->SessionDestroy(pvClientSession);
}

bool EUSignEncryptSessionData(void *pvSession, uint8_t *pbData,
                              unsigned long dwDataLength,
                              uint8_t **ppbEncryptedData,
                              unsigned long
*pdwEncryptedDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;

  if (pvSession == NULL || (dwDataLength > 0 && pbData == NULL)
||
      ppbEncryptedData == NULL || pdwEncryptedDataLength ==
NULL) {
    return false;
  }

  *ppbEncryptedData = NULL;
  *pdwEncryptedDataLength = 0;

  uint32_t dwError = s_pEUSignInterface->SessionEncrypt(
      pvSession, pbData, dwDataLength, ppbEncryptedData,
      pdwEncryptedDataLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

bool EUSignDecryptSessionData(void *pvSession, uint8_t
*pbEncryptedData,
                              unsigned long
dwEncryptedDataLength,
                              uint8_t **ppbDecryptedData,
                              unsigned long
*pdwDecryptedDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;
```

```
  if (pvSession == NULL ||
      (dwEncryptedDataLength > 0 && pbEncryptedData == NULL) ||
      ppbDecryptedData == NULL || pdwDecryptedDataLength ==
NULL) {
    return false;
  }

  *ppbDecryptedData = NULL;
  *pdwDecryptedDataLength = 0;

  uint32_t dwError = s_pEUSignInterface->SessionDecrypt(
      pvSession, pbEncryptedData, dwEncryptedDataLength,
ppbDecryptedData,
      pdwDecryptedDataLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

bool EUSignVerifyDataInternal(uint8_t *pbSignedData,
                              unsigned long dwSignedDataLength,
                              uint8_t **ppbData, unsigned long
*pdwDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;

  if (pbSignedData == NULL || dwSignedDataLength == 0 || ppbData
== NULL ||
      pdwDataLength == NULL) {
    return false;
  }

  *ppbData = NULL;
  *pdwDataLength = 0;

  EU_SIGN_INFO signInfo;
  memset(&signInfo, 0, sizeof(EU_SIGN_INFO));

  uint32_t dwError = s_pEUSignInterface->VerifyDataInternal(
      NULL, pbSignedData, dwSignedDataLength, ppbData,
pdwDataLength,
      &signInfo);

  if (dwError != EU_ERROR_NONE) {
    std::cerr << "EUSignVerifyDataInternal failed with error: "
<< dwError
              << std::endl;
    return false;
  }
```

```
  return true;
}
```

Файл ModbusPoller.h

```
#ifndef MODBUS_POOLER_H
#define MODBUS_POOLER_H

#include <atomic>
#include <thread>
#include <vector>

#include <nlohmann/json.hpp>

#include "ConfigManager.h"

class ModbusPoller {
public:
  ModbusPoller(const MachineConfig &config);
  ~ModbusPoller();

  void start();
  void stop();

private:
  void pollLoop();
  nlohmann::json processData(const std::vector<uint16_t>
&rawData);

  MachineConfig m_config;
  std::thread m_thread;
  std::atomic<bool> m_running;

  nlohmann::json m_lastSentSensors;
  std::chrono::time_point<std::chrono::system_clock>
m_lastSendTime;
  std::string m_lastStatus{"UNKNOWN"};
};

#endif // MODBUS_POOLER_H
```

Файл ModbusPoller.cpp

```
#include "ModbusPoller.h"
#include "NetworkClient.h"

#include <chrono>
#include <cstring>
#include <iostream>
#include <modbus/modbus.h>

ModbusPoller::ModbusPoller(const MachineConfig &config)
```

```
    : m_config(config), m_running(false) {}

ModbusPoller::~ModbusPoller() { stop(); }

void ModbusPoller::start() {
  if (!m_running) {
    m_running = true;
    m_thread = std::thread(&ModbusPoller::pollLoop, this);
  }
}

void ModbusPoller::stop() {
  if (m_running) {
    m_running = false;
    if (m_thread.joinable()) {
      m_thread.join();
    }
  }
}

void ModbusPoller::pollLoop() {
  std::cout << "[ModbusPoller] Starting thread for machine: "
            << m_config.machine_id << std::endl;

  modbus_t *ctx =
      modbus_new_tcp(m_config.modbus_ip.c_str(),
m_config.modbus_port);
  if (ctx == nullptr) {
    std::cerr << "[ModbusPoller] Unable to allocate libmodbus
context for "
              << m_config.machine_id << std::endl;
    return;
  }

  modbus_set_response_timeout(ctx, 2, 0);

  bool isConnected = false;

  while (m_running) {
    auto now = std::chrono::system_clock::now();
    nlohmann::json payload;
    payload["machine_id"] = m_config.machine_id;
    payload["timestamp"] =

std::chrono::duration_cast<std::chrono::seconds>(now.time_since_
epoch())
            .count();

    if (!isConnected) {
      if (modbus_connect(ctx) == -1) {
        std::cerr << "[ModbusPoller] Connection failed for "
                  << m_config.machine_id << ": " <<
modbus_strerror(errno)
```

```
                  << std::endl;

        if (m_lastStatus != "OFFLINE") {
          payload["status"] = "OFFLINE";
          payload["sensors"] = nlohmann::json::object();
          NetworkClient::getInstance().enqueueData(payload);
          m_lastStatus = "OFFLINE";
          m_lastSendTime = now;
          std::cout << "[ModbusPoller] Reported OFFLINE status
for "
                    << m_config.machine_id << std::endl;
        }


std::this_thread::sleep_for(std::chrono::milliseconds(2000));
        continue;
      }
      std::cout << "[ModbusPoller] Connected to " <<
m_config.machine_id
                << std::endl;
      isConnected = true;
    }

    nlohmann::json sensors;
    bool readSuccess = true;

    for (const auto &reg : m_config.registers) {
      uint16_t tab_reg[2] = {0, 0};
      int num_registers =
          (reg.type == "float32" || reg.type == "int32") ? 2 :
1;

      int rc = modbus_read_registers(ctx, reg.address,
num_registers, tab_reg);
      if (rc == -1) {
        std::cerr << "[ModbusPoller] Failed to read register "
<< reg.name
                  << " on " << m_config.machine_id << ": "
                  << modbus_strerror(errno) << std::endl;
        readSuccess = false;
        isConnected = false;
        modbus_close(ctx);
        break;
      }

      if (reg.type == "int16") {
        sensors[reg.name] = (int16_t)tab_reg[0];
      } else if (reg.type == "uint16") {
        sensors[reg.name] = tab_reg[0];
      } else if (reg.type == "float32") {
        float f;
        uint32_t i = ((uint32_t)tab_reg[0] << 16) | tab_reg[1];
        memcpy(&f, &i, sizeof(float));
```

```
        sensors[reg.name] = f;
      } else {
        sensors[reg.name] = tab_reg[0];
      }
    }

    if (readSuccess) {
      auto timeSinceLastSend =
          std::chrono::duration_cast<std::chrono::seconds>(now -
m_lastSendTime)
              .count();

      bool dataChanged = (sensors != m_lastSentSensors);
      bool statusChanged = (m_lastStatus != "ONLINE");
      bool heartbeatTriggered =
          (timeSinceLastSend >=
m_config.heartbeat_interval_sec);

      if (dataChanged || statusChanged || heartbeatTriggered) {
        payload["status"] = "ONLINE";
        payload["sensors"] = sensors;
        std::cout << "[ModbusPoller] Collected data from "
                  << m_config.machine_id
                  << (statusChanged
                          ? " (Status: ONLINE)"
                          : (dataChanged ? " (Changed)" : "
(Heartbeat)"))
                  << std::endl;

        NetworkClient::getInstance().enqueueData(payload);
        m_lastSentSensors = sensors;
        m_lastStatus = "ONLINE";
        m_lastSendTime = now;
      }
    }

    for (int i = 0; i < m_config.poll_interval_ms / 100 &&
m_running; ++i) {

std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }
  }

  if (isConnected) {
    modbus_close(ctx);
  }
  modbus_free(ctx);
  std::cout << "[ModbusPoller] Thread stopped for machine: "
            << m_config.machine_id << std::endl;
}
```

Файл NetworkClient.h

```
#ifndef NETWORK_CLIENT_H
#define NETWORK_CLIENT_H

#include <atomic>
#include <condition_variable>
#include <mutex>
#include <thread>
#include <vector>

#include <nlohmann/json.hpp>

#include "ConfigManager.h"

class NetworkClient {
public:
  static NetworkClient &getInstance() {
    static NetworkClient instance;
    return instance;
  }

  bool init(const ServerConfig &config);
  bool fetchCertificates();

  void start();
  void stop();
  void enqueueData(const nlohmann::json &payload);

  const std::vector<uint8_t> &getDsCert() const { return
m_dsCert; }
  const std::vector<uint8_t> &getKepCert() const { return
m_kepCert; }

private:
  NetworkClient() = default;

  void senderLoop();
  bool sendSecureBatch(const nlohmann::json &payloadArray);

  ServerConfig m_config;
  std::vector<uint8_t> m_dsCert;
  std::vector<uint8_t> m_kepCert;

  std::vector<nlohmann::json> m_queue;
  std::mutex m_mutex;
  std::condition_variable m_cv;
  std::thread m_thread;
  std::atomic<bool> m_running{false};
};

#endif // NETWORK_CLIENT_H
```

Файл NetworkClient.cpp

```
#include "NetworkClient.h"
#include "EUSignInterface.h"

#include "httplib.h"

#include <chrono>
#include <iostream>
#include <nlohmann/json.hpp>

using json = nlohmann::json;
using namespace std::chrono;

static std::vector<uint8_t> Base64Decode(const std::string &in)
{
  static const std::string base64_chars =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";
  auto is_base64 = [](unsigned char c) -> bool {
    return (isalnum(c) || (c == '+') || (c == '/'));
  };

  int in_len = in.length();
  int i = 0, j = 0, in_ = 0;
  unsigned char char_array_4[4], char_array_3[3];
  std::vector<uint8_t> ret;

  while (in_len-- && (in[in_] != '=') && is_base64(in[in_])) {
    char_array_4[i++] = in[in_];
    in_++;
    if (i == 4) {
      for (i = 0; i < 4; i++)
        char_array_4[i] = base64_chars.find(char_array_4[i]);
      char_array_3[0] =
          (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >>
4);
      char_array_3[1] =
          ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] &
0x3c) >> 2);
      char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];
      for (i = 0; (i < 3); i++)
        ret.push_back(char_array_3[i]);
      i = 0;
    }
  }
  if (i) {
    for (j = i; j < 4; j++)
      char_array_4[j] = 0;
    for (j = 0; j < 4; j++)
      char_array_4[j] = base64_chars.find(char_array_4[j]);
```

```
    char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1]
& 0x30) >> 4);
    char_array_3[1] =
        ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] &
0x3c) >> 2);
    char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];
    for (j = 0; (j < i - 1); j++)
      ret.push_back(char_array_3[j]);
  }
  return ret;
}

static std::string Base64Encode(const uint8_t *data, uint32_t
length) {
  static const char encoding_table[] =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";
  std::string out;
  uint32_t mod_table[] = {0, 2, 1};
  if (length == 0)
    return out;
  uint32_t out_len = 4 * ((length + 2) / 3);
  out.reserve(out_len);
  for (uint32_t i = 0; i < length;) {
    uint32_t octet_a = i < length ? data[i++] : 0;
    uint32_t octet_b = i < length ? data[i++] : 0;
    uint32_t octet_c = i < length ? data[i++] : 0;
    uint32_t triple = (octet_a << 0x10) + (octet_b << 0x08) +
octet_c;
    out.push_back(encoding_table[(triple >> 3 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 2 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 1 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 0 * 6) & 0x3F]);
  }
  for (uint32_t i = 0; i < mod_table[length % 3]; i++)
    out[out_len - 1 - i] = '=';
  return out;
}

bool NetworkClient::init(const ServerConfig &config) {
  m_config = config;
  if (!EUSignInitialize()) {
    std::cerr << "[NetworkClient] EUSignInitialize failed!" <<
std::endl;
    return false;
  }
  return true;
}

void NetworkClient::start() {
  if (!m_running) {
```

```
    m_running = true;
    m_thread = std::thread(&NetworkClient::senderLoop, this);
  }
}

void NetworkClient::stop() {
  if (m_running) {
    m_running = false;
    m_cv.notify_all();
    if (m_thread.joinable()) {
      m_thread.join();
    }
  }
}

void NetworkClient::enqueueData(const nlohmann::json &payload) {
  std::lock_guard<std::mutex> lock(m_mutex);
  m_queue.push_back(payload);
  if (m_queue.size() >=
(size_t)m_config.batching.max_batch_size) {
    m_cv.notify_one();
  }
}

void NetworkClient::senderLoop() {
  std::cout << "[NetworkClient] Sender loop started. Flush
interval: "
            << m_config.batching.flush_interval_sec
            << "s, Max batch: " <<
m_config.batching.max_batch_size
            << std::endl;

  while (m_running) {
    std::vector<json> batch;
    {
      std::unique_lock<std::mutex> lock(m_mutex);
      m_cv.wait_for(
          lock,
std::chrono::seconds(m_config.batching.flush_interval_sec),
          [this] {
            return !m_running ||
                   m_queue.size() >=
(size_t)m_config.batching.max_batch_size;
          });

      if (m_queue.empty()) {
        if (!m_running)
          break;
        continue;
      }

      batch = std::move(m_queue);
      m_queue.clear();
```

```
    }

    if (!batch.empty()) {
      json batchJson = batch;
      if (sendSecureBatch(batchJson)) {
        std::cout << "[NetworkClient] Successfully sent batch of
"
                  << batch.size() << " records." << std::endl;
      } else {
        std::cerr << "[NetworkClient] Failed to send batch of "
<< batch.size()
                  << " records." << std::endl;
      }
    }
  }
}

bool NetworkClient::fetchCertificates() {
  std::cout << "[NetworkClient] Fetching certificates from " <<
m_config.url
            << "/get-certificates..." << std::endl;

  httplib::Client cli(m_config.url);
  cli.set_connection_timeout(5, 0);
  cli.set_read_timeout(5, 0);

  if (auto res = cli.Get("/get-certificates")) {
    if (res->status == 200) {
      try {
        auto j = json::parse(res->body);
        if (j.contains("certificates") &&
j["certificates"].is_array() &&
            j["certificates"].size() == 2) {

          std::vector<uint8_t> cert1 =
Base64Decode(j["certificates"][0]);
          std::vector<uint8_t> cert2 =
Base64Decode(j["certificates"][1]);

          if (!ParseAndAssignCertificates(cert1, cert2,
m_dsCert, m_kepCert)) {
            std::cerr
                << "[NetworkClient] Failed to parse and assign
certificates."
                << std::endl;
            return false;
          }

          std::cout
              << "[NetworkClient] Successfully fetched and
parsed certificates."
              << std::endl;
          return true;
```

```
        }
      } catch (...) {
        std::cerr
            << "[NetworkClient] Failed to parse JSON response
from server."
            << std::endl;
      }
    }
  }
  return false;
}

bool NetworkClient::sendSecureBatch(const nlohmann::json
&payloadArray) {
  auto totalStart = high_resolution_clock::now();

  if (m_kepCert.empty()) {
    std::cerr << "[NetworkClient] KEP Certificate not loaded!"
<< std::endl;
    return false;
  }

  auto cryptoStart = high_resolution_clock::now();
  nlohmann::json authenticatedPayload;
  authenticatedPayload["client_token"] = m_config.token;
  authenticatedPayload["telemetry"] = payloadArray;

  void *pvSession = nullptr;
  uint8_t *pbAuthData = nullptr;
  unsigned long dwAuthDataLength = 0;

  if (!EUSignClientCreateDynamicSession((uint8_t
*)m_kepCert.data(),
                                        m_kepCert.size(),
&pvSession,
                                        &pbAuthData,
&dwAuthDataLength)) {
    return false;
  }

  std::string payloadStr = authenticatedPayload.dump();
  uint8_t *pbEncryptedData = nullptr;
  unsigned long dwEncryptedDataLength = 0;

  if (!EUSignEncryptSessionData(pvSession, (uint8_t
*)payloadStr.c_str(),
                                payloadStr.length(),
&pbEncryptedData,
                                &dwEncryptedDataLength)) {
    EUSignFreeMemory(pbAuthData);
    EUSignDestroyDynamicSession(pvSession);
    return false;
  }
```

```
  json requestJson;
  requestJson["authData"] = Base64Encode(pbAuthData,
dwAuthDataLength);
  requestJson["encryptedData"] =
      Base64Encode(pbEncryptedData, dwEncryptedDataLength);

  EUSignFreeMemory(pbAuthData);
  EUSignFreeMemory(pbEncryptedData);
  auto cryptoEnd = high_resolution_clock::now();

  auto netStart = high_resolution_clock::now();
  httplib::Client cli(m_config.url);
  cli.set_connection_timeout(5, 0);
  cli.set_read_timeout(5, 0);

  auto res = cli.Post("/send-data", requestJson.dump(),
"application/json");
  auto netEnd = high_resolution_clock::now();

  if (res && res->status == 200) {
    try {
      auto verifyStart = high_resolution_clock::now();
      auto respJson = json::parse(res->body);
      std::vector<uint8_t> encryptedReply =
          Base64Decode(respJson["encryptedData"]);
      uint8_t *pbDecryptedReply = nullptr;
      unsigned long dwDecryptedReplySize = 0;

      if (EUSignDecryptSessionData(pvSession,
encryptedReply.data(),
                                   encryptedReply.size(),
&pbDecryptedReply,
                                   &dwDecryptedReplySize)) {
        std::string decryptedReplyStr((char *)pbDecryptedReply,
                                      dwDecryptedReplySize);
        EUSignFreeMemory(pbDecryptedReply);
        auto signedReplyJson = json::parse(decryptedReplyStr);
        std::vector<uint8_t> signedData =
            Base64Decode(signedReplyJson["signedData"]);

        uint8_t *pbCleanReply = nullptr;
        unsigned long dwCleanReplySize = 0;
        if (EUSignVerifyDataInternal(signedData.data(),
signedData.size(),
                                     &pbCleanReply,
&dwCleanReplySize)) {
          EUSignFreeMemory(pbCleanReply);
          auto verifyEnd = high_resolution_clock::now();
          auto totalEnd = high_resolution_clock::now();

          auto durTotal =
```

```
            duration_cast<microseconds>(totalEnd -
totalStart).count();
        auto durCryptoEnc =
            duration_cast<microseconds>(cryptoEnd -
cryptoStart).count();
        auto durNet = duration_cast<microseconds>(netEnd -
netStart).count();
        auto durCryptoDec =
            duration_cast<microseconds>(verifyEnd -
verifyStart).count();

        printf("\n[CLIENT PERF] Batch sent (%zu records):\n",
               payloadArray.size());
        printf("  - Crypto Encrypt: %ld us\n", durCryptoEnc);
        printf("  - Network RTT:    %ld us (Wait for
server)\n", durNet);
        printf("  - Crypto Verify:  %ld us\n", durCryptoDec);
        printf("  - Total Cycle:    %ld us\n", durTotal);

        EUSignDestroyDynamicSession(pvSession);
        return true;
      }
    }
  } catch (...) {
  }
 } else if (res) {
  if (res->status == 403) {
    std::cerr << "[SECURITY ALERT] Server rejected data! HTTP
403 Forbidden "
                 "(Invalid Token?)"
              << std::endl;
  } else {
    std::cerr << "[NetworkClient] Server error: HTTP " << res-
>status
              << std::endl;
  }
 } else {
  std::cerr << "[NetworkClient] Connection failed." <<
std::endl;
 }

 EUSignDestroyDynamicSession(pvSession);
 return false;
}
```

Файл main.cpp

```
#include "ConfigManager.h"
#include "ModbusPoller.h"
#include "NetworkClient.h"

#include <atomic>
```

```
#include <csignal>
#include <iostream>
#include <memory>
#include <vector>

std::atomic<bool> g_running{true};

void signalHandler(int signum) {
  std::cout << "\nInterrupt signal (" << signum
            << ") received. Shutting down..." << std::endl;
  g_running = false;
}

int main() {
  std::cout << "Starting Argus Client Daemon..." << std::endl;

  std::signal(SIGINT, signalHandler);
  std::signal(SIGTERM, signalHandler);

  if (!ConfigManager::getInstance().loadConfig("config.json")) {
    std::cerr << "Failed to load configuration. Exiting." <<
std::endl;
    return 1;
  }

  const auto &serverConfig =
ConfigManager::getInstance().getServerConfig();
  std::cout << "Configured Server URL: " << serverConfig.url <<
std::endl;

  auto &netClient = NetworkClient::getInstance();
  netClient.init(serverConfig);

  if (!netClient.fetchCertificates()) {
    std::cerr << "CRITICAL: Could not fetch certificates from
server. Cannot "
                 "start securely. Exiting."
              << std::endl;
    return 1;
  }

  netClient.start();

  const auto &machines =
ConfigManager::getInstance().getMachines();
  std::cout << "Loaded " << machines.size() << " machines
configurations."
            << std::endl;

  std::vector<std::unique_ptr<ModbusPoller>> pollers;

  for (const auto &machine : machines) {
    std::cout << " - Machine ID: " << machine.machine_id
```

```
              << " (IP: " << machine.modbus_ip << ":" <<
machine.modbus_port
              << ")" << std::endl;
    std::cout << "   Registers to poll: " <<
machine.registers.size()
              << std::endl;

    auto poller = std::make_unique<ModbusPoller>(machine);
    poller->start();
    pollers.push_back(std::move(poller));
  }

  std::cout << "\nInitialization complete. Modbus threads are
running. Press "
               "Ctrl+C to stop."
            << std::endl;

  while (g_running) {
    std::this_thread::sleep_for(std::chrono::milliseconds(500));
  }

  std::cout << "Stopping Modbus pollers..." << std::endl;
  for (auto &poller : pollers) {
    poller->stop();
  }

  std::cout << "Stopping Network Client..." << std::endl;
  netClient.stop();

  std::cout << "Argus Client Daemon stopped." << std::endl;
  return 0;
}
```

Додаток Д

Вихідний код сервера

Файл DatabaseManager.h

```
#ifndef DATABASE_MANAGER_H
#define DATABASE_MANAGER_H

#include <nlohmann/json.hpp>
#include <pqxx/pqxx>
#include <string>

class DatabaseManager {
public:
  static DatabaseManager &getInstance() {
    static DatabaseManager instance;
    return instance;
  }

  bool init(const std::string &connectionString);
  bool insertTelemetry(const nlohmann::json &payload);
  bool insertTelemetryBatch(const nlohmann::json
&telemetryArray);

private:
  DatabaseManager() = default;
  std::string m_connectionString;
  bool m_initialized = false;
};

#endif // DATABASE_MANAGER_H
```

Файл DatabaseManager.cpp

```
#include "DatabaseManager.h"

#include <iostream>

bool DatabaseManager::init(const std::string &connectionString)
{
  m_connectionString = connectionString;
  try {
    pqxx::connection conn(m_connectionString);
    if (conn.is_open()) {
      std::cout << "[DatabaseManager] Connected to PostgreSQL
successfully."
                << std::endl;

      pqxx::work txn(conn);
      txn.exec("CREATE TABLE IF NOT EXISTS telemetry ("
               "id SERIAL PRIMARY KEY, "
               "timestamp BIGINT, "
```

```
                "machine_id VARCHAR(255), "
                "status VARCHAR(50), "
                "data JSONB"
                ");");
      txn.commit();
      m_initialized = true;
      return true;
    }
  } catch (const std::exception &e) {
    std::cerr << "[DatabaseManager] Initialization failed: " <<
e.what()
              << std::endl;
  }
  return false;
}

bool DatabaseManager::insertTelemetry(const nlohmann::json
&payload) {
  if (!m_initialized)
    return false;

  try {
    std::string machine_id = payload.value("machine_id",
"UNKNOWN");
    int64_t timestamp = payload.value("timestamp", 0);
    std::string status = payload.value("status", "UNKNOWN");

    nlohmann::json sensors = payload.contains("sensors")
                                 ? payload["sensors"]
                                 : nlohmann::json::object();

    pqxx::connection conn(m_connectionString);
    pqxx::work txn(conn);

    conn.prepare("insert_telemetry",
                 "INSERT INTO telemetry (timestamp, machine_id,
status, data) "
                 "VALUES ($1, $2, $3, $4)");

    txn.exec_prepared("insert_telemetry", timestamp, machine_id,
status,
                      sensors.dump());
    txn.commit();

    return true;
  } catch (const std::exception &e) {
    std::cerr << "[DatabaseManager] Failed to insert telemetry:
" << e.what()
              << std::endl;
    return false;
  }
}
```

```
bool DatabaseManager::insertTelemetryBatch(
    const nlohmann::json &telemetryArray) {
  if (!m_initialized)
    return false;

  try {
    pqxx::connection conn(m_connectionString);
    pqxx::work txn(conn);

    conn.prepare("insert_telemetry_batch",
                 "INSERT INTO telemetry (timestamp, machine_id,
status, data) "
                 "VALUES ($1, $2, $3, $4)");

    for (const auto &payload : telemetryArray) {
      std::string machine_id = payload.value("machine_id",
"UNKNOWN");
      int64_t timestamp = payload.value("timestamp", 0);
      std::string status = payload.value("status", "UNKNOWN");
      nlohmann::json sensors = payload.contains("sensors")
                                   ? payload["sensors"]
                                   : nlohmann::json::object();

      txn.exec_prepared("insert_telemetry_batch", timestamp,
machine_id, status,
                        sensors.dump());
    }

    txn.commit();
    return true;
  } catch (const std::exception &e) {
    std::cerr << "[DatabaseManager] Failed to insert telemetry
batch: "
              << e.what() << std::endl;
    return false;
  }
}
```

Файл EUSignInterface.h

```
#ifndef EU_SIGN_INTERFACE_H
#define EU_SIGN_INTERFACE_H

#include <cstdint>

bool EUSignInitialize();

void EUSignFinalize();

void EUSignFreeMemory(uint8_t *pbMemory);

bool EUSignCreateDynamicSession(uint8_t *pbAuthData,
```

```
                                    unsigned long dwAuthDataLength,
                                    void **ppvServerSession);

void EUSignDestroyDynamicSession(void *pvServerSession);

bool EUSignEncryptSessionData(void *pvServerSession, uint8_t
*pbData,
                              unsigned long dwDataLength,
                              uint8_t **ppbEncryptedData,
                              unsigned long
*pdwEncryptedDataLength);

bool EUSignDecryptSessionData(void *pvServerSession, uint8_t
*pbEncryptedData,
                              unsigned long
dwEncryptedDataLength,
                              uint8_t **ppbDecryptedData,
                              unsigned long
*pdwDecryptedDataLength);

bool EUSignSignDataWithServerKey(uint8_t *pbData, unsigned long
dwDataLength,
                                 uint8_t **ppbSignature,
                                 unsigned long
*pdwSignatureLength);

bool EUSignGetServerCertificates(uint8_t
**ppbDSServerCertificate,
                                 unsigned long
*pdwDSServerCertificateLength,
                                 uint8_t
**ppbKEPServerCertificate,
                                 unsigned long
*pdwKEPServerCertificateLength);

#endif // EU_SIGN_INTERFACE_H
```

Файл EUSignInterface.cpp

```
#include "EUSignInterface.h"
#include "EUSignCP.h"

#include <cstring>
#include <fstream>
#include <iconv.h>
#include <sstream>
#include <string>

#define SERVER_KEY_PATH "EUTestPKeyAndCerts/PrivateKey/Key-
6.dat"
#define SERVER_PASSWORD_PATH
"EUTestPKeyAndCerts/PrivateKey/Password.txt"
```

```
#define SESSION_EXPIRE_TIME_SEC 3600

#define ALGORITHM_NAME_DSTU4145 "DSTU4145"

typedef struct _EU_SIGN_STATE {
  uint32_t dwGryadaTypeIndex;
  uint32_t dwGryadaDeviceIndex;

  char *pszDSServerCertificateIssuer;
  char *pszDSServerCertificateSerial;
  char *pszKEPServerCertificateIssuer;
  char *pszKEPServerCertificateSerial;

  uint8_t *pbCMPCertificate;
  uint32_t dwCMPCertificateLength;

  void *pvKEPPrivateKeyContext;
} EU_SIGN_STATE, *PEU_SIGN_STATE;

static PEU_INTERFACE s_pEUSignInterface = NULL;
static PEU_SIGN_STATE s_pEUSignState = NULL;

static bool ReadFileToString(const char *path, char **outStr) {
  std::ifstream file(path, std::ios::binary);
  if (!file)
    return false;

  std::stringstream buffer;
  buffer << file.rdbuf();

  std::string s = buffer.str();
  *outStr = new char[s.length() + 1];
  strcpy(*outStr, s.c_str());

  return true;
}

static bool InitializeLibrary() {
  if (!EULoad())
    return false;

  s_pEUSignInterface = EUGetInterface();
  if (s_pEUSignInterface == NULL)
    return false;

  s_pEUSignInterface->SetUIMode(false);

  uint32_t dwError = s_pEUSignInterface->Initialize();
  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
```

```
}

static bool LoadServerPrivateKey() {
  char *pszPassword = NULL;
  void *pvContext = NULL;

  if (!ReadFileToString(SERVER_PASSWORD_PATH, &pszPassword))
    return false;

  uint32_t dwError = s_pEUSignInterface->CtxCreate(&pvContext);
  if (dwError != EU_ERROR_NONE) {
    delete[] pszPassword;
    return false;
  }

  dwError = s_pEUSignInterface->CtxReadPrivateKeyFile(
      pvContext, (char *)SERVER_KEY_PATH, pszPassword,
      &s_pEUSignState->pvKEPPrivateKeyContext, NULL);

  delete[] pszPassword;
  s_pEUSignInterface->CtxFree(pvContext);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

static bool FindAssociatedCertificates() {
  uint32_t dwError;
  uint32_t dwIndex = 0;

  while (true) {
    PEU_CERT_INFO_EX pCertificateInfo = NULL;

    dwError = s_pEUSignInterface->CtxEnumOwnCertificates(
        s_pEUSignState->pvKEPPrivateKeyContext, dwIndex,
&pCertificateInfo,
        NULL, NULL);

    if (dwError == EU_WARNING_END_OF_ENUM)
      break;

    if (dwError != EU_ERROR_NONE)
      return false;

    if (pCertificateInfo->dwKeyUsage &
EU_KEY_USAGE_DIGITAL_SIGNATURE) {
      uint32_t dwIssuerLength =
          (uint32_t)strlen(pCertificateInfo->pszIssuer) + 1;

      s_pEUSignState->pszDSServerCertificateIssuer = new
char[dwIssuerLength];
```

```
      if (s_pEUSignState->pszDSServerCertificateIssuer == NULL)
{
        s_pEUSignInterface-
>FreeCertificateInfoEx(pCertificateInfo);
        return false;
      }

      memcpy(s_pEUSignState->pszDSServerCertificateIssuer,
             pCertificateInfo->pszIssuer, dwIssuerLength);

      uint32_t dwSerialLength =
          (uint32_t)strlen(pCertificateInfo->pszSerial) + 1;

      s_pEUSignState->pszDSServerCertificateSerial = new
char[dwSerialLength];
      if (s_pEUSignState->pszDSServerCertificateSerial == NULL)
{
        s_pEUSignInterface-
>FreeCertificateInfoEx(pCertificateInfo);
        return false;
      }

      memcpy(s_pEUSignState->pszDSServerCertificateSerial,
             pCertificateInfo->pszSerial, dwSerialLength);
    } else if (pCertificateInfo->dwKeyUsage &
EU_KEY_USAGE_KEY_AGREEMENT) {
      uint32_t dwIssuerLength =
          (uint32_t)strlen(pCertificateInfo->pszIssuer) + 1;

      s_pEUSignState->pszKEPServerCertificateIssuer = new
char[dwIssuerLength];
      if (s_pEUSignState->pszKEPServerCertificateIssuer == NULL)
{
        s_pEUSignInterface-
>FreeCertificateInfoEx(pCertificateInfo);
        return false;
      }

      memcpy(s_pEUSignState->pszKEPServerCertificateIssuer,
             pCertificateInfo->pszIssuer, dwIssuerLength);

      uint32_t dwSerialLength =
          (uint32_t)strlen(pCertificateInfo->pszSerial) + 1;

      s_pEUSignState->pszKEPServerCertificateSerial = new
char[dwSerialLength];
      if (s_pEUSignState->pszKEPServerCertificateSerial == NULL)
{
        s_pEUSignInterface-
>FreeCertificateInfoEx(pCertificateInfo);
        return false;
      }
```

```
      memcpy(s_pEUSignState->pszKEPServerCertificateSerial,
             pCertificateInfo->pszSerial, dwSerialLength);
    }

    s_pEUSignInterface->FreeCertificateInfoEx(pCertificateInfo);

    dwIndex++;
  }

  if (s_pEUSignState->pszDSServerCertificateIssuer == NULL ||
      s_pEUSignState->pszDSServerCertificateSerial == NULL ||
      s_pEUSignState->pszKEPServerCertificateIssuer == NULL ||
      s_pEUSignState->pszKEPServerCertificateSerial == NULL) {
    return false;
  }

  return true;
}

bool EUSignInitialize() {
  if (s_pEUSignInterface != NULL)
    return true;

  if (!InitializeLibrary())
    return false;

  s_pEUSignState = new EU_SIGN_STATE();
  if (s_pEUSignState == NULL)
    return false;

  memset(s_pEUSignState, 0, sizeof(EU_SIGN_STATE));

  s_pEUSignState->dwGryadaTypeIndex = (uint32_t)-1;
  s_pEUSignState->dwGryadaDeviceIndex = (uint32_t)-1;

  if (!LoadServerPrivateKey())
    return false;

  if (!FindAssociatedCertificates())
    return false;

  return true;
}

void EUSignFinalize() {
  if (s_pEUSignState != NULL) {

    if (s_pEUSignInterface != NULL) {
      if (s_pEUSignState->pvKEPPrivateKeyContext != NULL) {
        s_pEUSignInterface->CtxFreePrivateKey(
            s_pEUSignState->pvKEPPrivateKeyContext);
      }
    }
```

```
    if (s_pEUSignState->pszDSServerCertificateIssuer != NULL)
      delete[] s_pEUSignState->pszDSServerCertificateIssuer;

    if (s_pEUSignState->pszDSServerCertificateSerial != NULL)
      delete[] s_pEUSignState->pszDSServerCertificateSerial;

    if (s_pEUSignState->pszKEPServerCertificateIssuer != NULL)
      delete[] s_pEUSignState->pszKEPServerCertificateIssuer;

    if (s_pEUSignState->pszKEPServerCertificateSerial != NULL)
      delete[] s_pEUSignState->pszKEPServerCertificateSerial;

    delete[] s_pEUSignState->pbCMPCertificate;

    delete s_pEUSignState;
    s_pEUSignState = NULL;
  }

  if (s_pEUSignInterface != NULL) {
    s_pEUSignInterface->Finalize();
    EUUnload();
    s_pEUSignInterface = NULL;
  }
}

void EUSignFreeMemory(uint8_t *pbMemory) {
  if (s_pEUSignInterface == NULL || pbMemory == NULL)
    return;

  s_pEUSignInterface->FreeMemory(pbMemory);
}

bool EUSignCreateDynamicSession(uint8_t *pbAuthData,
                                unsigned long dwAuthDataLength,
                                void **ppvServerSession) {
  if (s_pEUSignInterface == NULL || s_pEUSignState == NULL)
    return false;

  if (s_pEUSignState->pvKEPPrivateKeyContext == NULL)
    return false;

  if (pbAuthData == NULL || dwAuthDataLength == 0 ||
ppvServerSession == NULL) {
    return false;
  }

  *ppvServerSession = NULL;

  uint32_t dwError = s_pEUSignInterface-
>CtxServerDynamicKeySessionCreate(
      s_pEUSignState->pvKEPPrivateKeyContext,
SESSION_EXPIRE_TIME_SEC,
```

```
      pbAuthData, dwAuthDataLength, ppvServerSession);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

void EUSignDestroyDynamicSession(void *pvServerSession) {
  if (s_pEUSignInterface == NULL || pvServerSession == NULL)
    return;

  s_pEUSignInterface->SessionDestroy(pvServerSession);
}

bool EUSignEncryptSessionData(void *pvServerSession, uint8_t
*pbData,
                              unsigned long dwDataLength,
                              uint8_t **ppbEncryptedData,
                              unsigned long
*pdwEncryptedDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;

  if (pvServerSession == NULL || (dwDataLength > 0 && pbData ==
NULL) ||
      ppbEncryptedData == NULL || pdwEncryptedDataLength ==
NULL) {
    return false;
  }

  *ppbEncryptedData = NULL;
  *pdwEncryptedDataLength = 0;

  uint32_t dwError = s_pEUSignInterface->SessionEncrypt(
      pvServerSession, pbData, dwDataLength, ppbEncryptedData,
      pdwEncryptedDataLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

bool EUSignDecryptSessionData(void *pvServerSession, uint8_t
*pbEncryptedData,
                              unsigned long
dwEncryptedDataLength,
                              uint8_t **ppbDecryptedData,
                              unsigned long
*pdwDecryptedDataLength) {
  if (s_pEUSignInterface == NULL)
    return false;
```

```
  if (pvServerSession == NULL ||
      (dwEncryptedDataLength > 0 && pbEncryptedData == NULL) ||
      ppbDecryptedData == NULL || pdwDecryptedDataLength ==
NULL) {
    return false;
  }

  *ppbDecryptedData = NULL;
  *pdwDecryptedDataLength = 0;

  uint32_t dwError = s_pEUSignInterface->SessionDecrypt(
      pvServerSession, pbEncryptedData, dwEncryptedDataLength,
ppbDecryptedData,
      pdwDecryptedDataLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}

bool EUSignSignDataWithServerKey(uint8_t *pbData, unsigned long
dwDataLength,
                                 uint8_t **ppbSignature,
                                 unsigned long
*pdwSignatureLength) {
  if (s_pEUSignInterface == NULL || s_pEUSignState == NULL)
    return false;

  if (s_pEUSignState->pvKEPPrivateKeyContext == NULL)
    return false;

  if ((dwDataLength > 0 && pbData == NULL) || ppbSignature ==
NULL ||
      pdwSignatureLength == NULL) {
    return false;
  }

  *ppbSignature = NULL;
  *pdwSignatureLength = 0;

  uint32_t dwError = s_pEUSignInterface->CtxSignData(
      s_pEUSignState->pvKEPPrivateKeyContext,
      EU_CTX_SIGN_DSTU4145_WITH_GOST34311, pbData, dwDataLength,
false, true,
      ppbSignature, pdwSignatureLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  return true;
}
```

```
bool EUSignGetServerCertificates(uint8_t
**ppbDSServerCertificate,
                                 unsigned long
*pdwDSServerCertificateLength,
                                 uint8_t
**ppbKEPServerCertificate,
                                 unsigned long
*pdwKEPServerCertificateLength) {
  if (s_pEUSignInterface == NULL || s_pEUSignState == NULL)
    return false;

  if (s_pEUSignState->pszDSServerCertificateIssuer == NULL ||
      s_pEUSignState->pszKEPServerCertificateIssuer == NULL) {
    return false;
  }

  if (ppbDSServerCertificate == NULL ||
pdwDSServerCertificateLength == NULL ||
      ppbKEPServerCertificate == NULL ||
      pdwKEPServerCertificateLength == NULL) {
    return false;
  }

  *ppbDSServerCertificate = NULL;
  *ppbKEPServerCertificate = NULL;

  uint32_t dwError = s_pEUSignInterface->GetCertificate(
      s_pEUSignState->pszDSServerCertificateIssuer,
      s_pEUSignState->pszDSServerCertificateSerial, NULL,
      ppbDSServerCertificate, pdwDSServerCertificateLength);

  if (dwError != EU_ERROR_NONE)
    return false;

  dwError = s_pEUSignInterface->GetCertificate(
      s_pEUSignState->pszKEPServerCertificateIssuer,
      s_pEUSignState->pszKEPServerCertificateSerial, NULL,
      ppbKEPServerCertificate, pdwKEPServerCertificateLength);

  if (dwError != EU_ERROR_NONE) {
    s_pEUSignInterface->FreeMemory(*ppbDSServerCertificate);
    *ppbDSServerCertificate = NULL;

    return false;
  }

  return true;
}
```

Файл ServerController.h

```
#ifndef SERVER_CONTROLER_H
#define SERVER_CONTROLER_H

#include "httplib.h"
#include <string>
#include <unordered_set>

class ServerController {
public:
  ServerController();

  bool loadConfig(const std::string &filepath);

  void handleGetCertificates(const httplib::Request &req,
                             httplib::Response &res);
  void handleSendData(const httplib::Request &req,
httplib::Response &res);

private:
  std::unordered_set<std::string> m_allowedTokens;
};

#endif // SERVER_CONTROLER_H
```

Файл ServerController.cpp

```
#include "ServerController.h"
#include "DatabaseManager.h"
#include "EUSignInterface.h"

#include <iostream>
#include <nlohmann/json.hpp>
#include <vector>

using json = nlohmann::json;

static std::string Base64Encode(const uint8_t *data, uint32_t
length) {
  static const char encoding_table[] =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";
  std::string out;
  uint32_t mod_table[] = {0, 2, 1};

  if (length == 0)
    return out;

  uint32_t out_len = 4 * ((length + 2) / 3);
  out.reserve(out_len);
```

```
  for (uint32_t i = 0; i < length;) {
    uint32_t octet_a = i < length ? data[i++] : 0;
    uint32_t octet_b = i < length ? data[i++] : 0;
    uint32_t octet_c = i < length ? data[i++] : 0;

    uint32_t triple = (octet_a << 0x10) + (octet_b << 0x08) +
octet_c;

    out.push_back(encoding_table[(triple >> 3 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 2 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 1 * 6) & 0x3F]);
    out.push_back(encoding_table[(triple >> 0 * 6) & 0x3F]);
  }

  for (uint32_t i = 0; i < mod_table[length % 3]; i++)
    out[out_len - 1 - i] = '=';

  return out;
}

static std::vector<uint8_t> Base64Decode(const std::string &in)
{
  static const std::string base64_chars =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";
  auto is_base64 = [](unsigned char c) -> bool {
    return (isalnum(c) || (c == '+') || (c == '/'));
  };

  int in_len = in.length();
  int i = 0;
  int j = 0;
  int in_ = 0;
  unsigned char char_array_4[4], char_array_3[3];
  std::vector<uint8_t> ret;

  while (in_len-- && (in[in_] != '=') && is_base64(in[in_])) {
    char_array_4[i++] = in[in_];
    in_++;
    if (i == 4) {
      for (i = 0; i < 4; i++)
        char_array_4[i] = base64_chars.find(char_array_4[i]);

      char_array_3[0] =
          (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >>
4);
      char_array_3[1] =
          ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] &
0x3c) >> 2);
      char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];
```

```
      for (i = 0; (i < 3); i++)
        ret.push_back(char_array_3[i]);
      i = 0;
    }
  }

  if (i) {
    for (j = i; j < 4; j++)
      char_array_4[j] = 0;

    for (j = 0; j < 4; j++)
      char_array_4[j] = base64_chars.find(char_array_4[j]);

    char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1]
& 0x30) >> 4);
    char_array_3[1] =
        ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] &
0x3c) >> 2);
    char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];

    for (j = 0; (j < i - 1); j++)
      ret.push_back(char_array_3[j]);
  }

  return ret;
}

#include <fstream>

ServerController::ServerController() {}

bool ServerController::loadConfig(const std::string &filepath) {
  try {
    std::ifstream file(filepath);
    if (!file.is_open()) {
      std::cerr << "[ServerController] Failed to open server
config: "
                << filepath << std::endl;
      return false;
    }

    json j;
    file >> j;

    if (j.contains("allowed_clients") &&
j["allowed_clients"].is_array()) {
      for (const auto &token : j["allowed_clients"]) {
        m_allowedTokens.insert(token.get<std::string>());
      }
      std::cout << "[ServerController] Loaded " <<
m_allowedTokens.size()
```

```
                << " allowed client tokens." << std::endl;
    }
    return true;
  } catch (const std::exception &e) {
    std::cerr << "[ServerController] Failed to load config: " <<
e.what()
              << std::endl;
    return false;
  }
}

bool DecryptRequest(const std::string &requestBody, json
&outPayload,
                    void **ppvSession) {
  try {
    auto requestJson = json::parse(requestBody);

    if (!requestJson.contains("authData") ||
        !requestJson.contains("encryptedData")) {
      return false;
    }

    std::vector<uint8_t> authData =
Base64Decode(requestJson["authData"]);
    std::vector<uint8_t> encryptedData =
        Base64Decode(requestJson["encryptedData"]);

    if (!EUSignCreateDynamicSession(authData.data(),
authData.size(),
                                    ppvSession)) {
      return false;
    }

    uint8_t *pbDecryptedPayload = nullptr;
    unsigned long dwDecryptedPayloadSize = 0;
    if (!EUSignDecryptSessionData(*ppvSession,
encryptedData.data(),
                                  encryptedData.size(),
&pbDecryptedPayload,
                                  &dwDecryptedPayloadSize)) {
      EUSignDestroyDynamicSession(*ppvSession);
      *ppvSession = nullptr;
      return false;
    }

    std::string decryptedStr((char *)pbDecryptedPayload,
                             dwDecryptedPayloadSize);
    outPayload = json::parse(decryptedStr);

    EUSignFreeMemory(pbDecryptedPayload);
    return true;
  } catch (...) {
    return false;
```

```
  }
}

bool EncryptResponse(void *pvSession, const json
&responsePayload,
                     std::string &outEncryptedBody) {
  try {
    std::string payloadStr = responsePayload.dump();

    uint8_t *pbSignature = nullptr;
    unsigned long dwSignatureSize = 0;
    if (!EUSignSignDataWithServerKey((uint8_t
*)payloadStr.c_str(),
                                     payloadStr.length(),
&pbSignature,
                                     &dwSignatureSize)) {
      return false;
    }

    json signedObject;
    signedObject["signedData"] = Base64Encode(pbSignature,
dwSignatureSize);
    std::string signedStr = signedObject.dump();

    EUSignFreeMemory(pbSignature);

    uint8_t *pbEncryptedData = nullptr;
    unsigned long dwEncryptedDataSize = 0;
    if (!EUSignEncryptSessionData(pvSession, (uint8_t
*)signedStr.c_str(),
                                  signedStr.length(),
&pbEncryptedData,
                                  &dwEncryptedDataSize)) {
      return false;
    }

    json finalResponse;
    finalResponse["encryptedData"] =
        Base64Encode(pbEncryptedData, dwEncryptedDataSize);
    outEncryptedBody = finalResponse.dump();

    EUSignFreeMemory(pbEncryptedData);
    return true;
  } catch (...) {
    return false;
  }
}

void ServerController::handleGetCertificates(const
httplib::Request &req,
                                              httplib::Response
&res) {
  uint8_t *pbDSServerCertificate = nullptr;
```

```
  unsigned long dwDSServerCertificateLength = 0;
  uint8_t *pbKEPServerCertificate = nullptr;
  unsigned long dwKEPServerCertificateLength = 0;

  if (!EUSignGetServerCertificates(
          &pbDSServerCertificate, &dwDSServerCertificateLength,
          &pbKEPServerCertificate,
&dwKEPServerCertificateLength)) {
    std::cerr << "Failed to get server certificates from EUSign"
<< std::endl;
    res.status = 500;
    res.set_content("{\"error\":\"Internal Server Error\"}",
                    "application/json");
    return;
  }

  std::string dsBase64 =
      Base64Encode(pbDSServerCertificate,
dwDSServerCertificateLength);
  std::string kepBase64 =
      Base64Encode(pbKEPServerCertificate,
dwKEPServerCertificateLength);

  EUSignFreeMemory(pbDSServerCertificate);
  EUSignFreeMemory(pbKEPServerCertificate);

  json responseJson;
  responseJson["certificates"] = json::array({dsBase64,
kepBase64});

  res.status = 200;
  res.set_content(responseJson.dump(), "application/json");

  std::cout << "Successfully formed '/get-certificates'
response" << std::endl;
}

void ServerController::handleSendData(const httplib::Request
&req,
                                      httplib::Response &res) {
  auto startTime = std::chrono::high_resolution_clock::now();

  void *pvSession = nullptr;
  json decryptedPayload;

  auto decStart = std::chrono::high_resolution_clock::now();
  bool decSuccess = DecryptRequest(req.body, decryptedPayload,
&pvSession);
  auto decEnd = std::chrono::high_resolution_clock::now();

  if (!decSuccess) {
    res.status = 400;
```

```
    res.set_content("{\"error\":\"Failed to decrypt or parse
request\"}",
                    "application/json");
    return;
  }

  std::string clientToken =
decryptedPayload.value("client_token", "");
  if (m_allowedTokens.find(clientToken) ==
m_allowedTokens.end()) {
    res.status = 403;
    res.set_content("{\"error\":\"Authentication failed\"}",
                    "application/json");
    EUSignDestroyDynamicSession(pvSession);
    return;
  }

  auto dbStart = std::chrono::high_resolution_clock::now();
  size_t recordCount = 0;
  if (decryptedPayload.contains("telemetry") &&
      decryptedPayload["telemetry"].is_array()) {
    recordCount = decryptedPayload["telemetry"].size();
    DatabaseManager::getInstance().insertTelemetryBatch(
        decryptedPayload["telemetry"]);
  } else {
    recordCount = 1;

DatabaseManager::getInstance().insertTelemetry(decryptedPayload)
;
  }
  auto dbEnd = std::chrono::high_resolution_clock::now();

  json replyJson = {{"status", "success"}, {"message", "Data
processed"}};
  std::string encryptedBody;

  auto encStart = std::chrono::high_resolution_clock::now();
  bool encSuccess = EncryptResponse(pvSession, replyJson,
encryptedBody);
  auto encEnd = std::chrono::high_resolution_clock::now();

  auto totalEnd = std::chrono::high_resolution_clock::now();

  if (encSuccess) {
    res.status = 200;
    res.set_content(encryptedBody, "application/json");
  } else {
    res.status = 500;
    res.set_content("{\"error\":\"Failed to encrypt
response\"}",
                    "application/json");
  }
```

```
  auto durTotal =
std::chrono::duration_cast<std::chrono::microseconds>(
                      totalEnd - startTime)
                      .count();
  auto durDec =

std::chrono::duration_cast<std::chrono::microseconds>(decEnd -
decStart)
          .count();
  auto durDb =

std::chrono::duration_cast<std::chrono::microseconds>(dbEnd -
dbStart)
          .count();
  auto durEnc =

std::chrono::duration_cast<std::chrono::microseconds>(encEnd -
encStart)
          .count();
  auto durCrypto = durDec + durEnc;

  printf("\n[PERF] Request processed from %s (%zu records):\n",
         clientToken.c_str(), recordCount);
  printf("  - Decryption: %ld us\n", durDec);
  printf("  - DB Insert:   %ld us (Baseline without crypto)\n",
durDb);
  printf("  - Encryption: %ld us\n", durEnc);
  printf("  ----------------------------------\n");
  printf("  - Crypto Overhead: %ld us (%.1f%% of total)\n",
durCrypto,
         (double)durCrypto / durTotal * 100);
  printf("  - Total Time:      %ld us\n", durTotal);

  EUSignDestroyDynamicSession(pvSession);
}
```

Файл main.cpp

```
#include <iostream>

#include "DatabaseManager.h"
#include "EUSignInterface.h"
#include "ServerController.h"
#include "httplib.h"

int main() {
  if (!EUSignInitialize())
    return 1;

  std::cout << "Initializing Argus Server..." << std::endl;

  std::string dbConnStr =
```

```
      "dbname=argus user=postgres password=postgres
host=127.0.0.1 port=5432";
  if (!DatabaseManager::getInstance().init(dbConnStr)) {
    std::cerr << "Warning: Could not connect to PostgreSQL.
Telemetry will NOT "
                 "be saved."
              << std::endl;
  }

  ServerController controller;
  if (!controller.loadConfig("server_config.json")) {
    std::cerr << "Warning: server_config.json not found or
invalid. No clients "
                 "will be able to connect."
              << std::endl;
  }

  httplib::Server svr;

  svr.Get("/get-certificates",
          [&controller](const httplib::Request &req,
httplib::Response &res) {
            controller.handleGetCertificates(req, res);
          });

  svr.Post("/send-data",
           [&controller](const httplib::Request &req,
httplib::Response &res) {
             controller.handleSendData(req, res);
           });

  int port = 8080;
  std::cout << "Server is listening on http://0.0.0.0:" << port
<< std::endl;
  std::cout << "Available endpoints:" << std::endl;
  std::cout << "  - GET  /get-certificates" << std::endl;
  std::cout << "  - POST /send-data" << std::endl;

  if (!svr.listen("0.0.0.0", port)) {
    std::cerr << "Failed to start server. Port might be in use."
<< std::endl;
    return 1;
  }

  return 0;
}
```

Додаток Е

Вихідний код тестових скриптів

Файл orchestrator.py

```
import os
import json
import shutil
import subprocess

NUM_CLIENTS = 20
MACHINES_PER_CLIENT = 5
START_PORT = 5000
BASE_DIR = os.path.abspath(os.getcwd())
BUILD_BIN_DIR = os.path.join(BASE_DIR, "build/bin")
TEST_ENV_DIR = os.path.join(BASE_DIR, "test_env")

def setup_stress_test():
    print(f"--- Setting up Stress Test: {NUM_CLIENTS} clients,
{NUM_CLIENTS * MACHINES_PER_CLIENT} machines ---")

    if os.path.exists(TEST_ENV_DIR):
        shutil.rmtree(TEST_ENV_DIR)
    os.makedirs(TEST_ENV_DIR)

    all_tokens = []
    machine_port_map = {}

    for i in range(1, NUM_CLIENTS + 1):
        client_id = f"client_{i}"
        client_dir = os.path.join(TEST_ENV_DIR, client_id)
        os.makedirs(client_dir)

        for item in os.listdir(BUILD_BIN_DIR):
            s = os.path.join(BUILD_BIN_DIR, item)
            d = os.path.join(client_dir, item)
            if item == "config.json" or item ==
"server_config.json":
                continue
            if os.path.isdir(s):
                shutil.copytree(s, d)
            else:
                shutil.copy2(s, d)

        token = f"token_for_client_{i}"
        all_tokens.append(token)

        machines = []
        for m in range(1, MACHINES_PER_CLIENT + 1):
            port = START_PORT + (i - 1) * MACHINES_PER_CLIENT +
(m - 1)
            m_id = f"M_{i}_{m}"
```

```
            if i == 1 and m == 1:
                regs = [
                    {"name": "spindle_temp", "address": 100,
"type": "int16"},
                    {"name": "spindle_rpm", "address": 101,
"type": "uint16"},
                    {"name": "vibration", "address": 102,
"type": "float32"},
                    {"name": "oil_pressure", "address": 104,
"type": "float32"},
                    {"name": "coolant_flow", "address": 106,
"type": "float32"},
                    {"name": "load_pct", "address": 108, "type":
"int16"},
                    {"name": "current_a", "address": 109,
"type": "float32"},
                    {"name": "pos_x", "address": 111, "type":
"float32"},
                    {"name": "pos_y", "address": 113, "type":
"float32"},
                    {"name": "pos_z", "address": 115, "type":
"float32"},
                    {"name": "parts_total", "address": 117,
"type": "uint16"},
                    {"name": "error_code", "address": 118,
"type": "uint16"}
                ]
            else:
                regs = [
                    {"name": "temp", "address": 100, "type":
"int16"},
                    {"name": "load", "address": 102, "type":
"float32"}
                ]

            machines.append({
                "machine_id": m_id,
                "modbus_ip": "127.0.0.1",
                "modbus_port": port,
                "poll_interval_ms": 1000,
                "heartbeat_interval_sec": 30,
                "registers": regs
            })
            machine_port_map[port] = m_id

        client_config = {
            "server": {
                "url": "http://localhost:8080",
                "token": token
            },
            "batching": {
                "flush_interval_sec": 5,
                "max_batch_size": 20
```

```
            },
            "machines": machines
        }

        with open(os.path.join(client_dir, "config.json"), 'w')
as f:
            json.dump(client_config, f, indent=2)

    server_config = {
        "allowed_clients": all_tokens
    }
    with open(os.path.join(BUILD_BIN_DIR, "server_config.json"),
'w') as f:
        json.dump(server_config, f, indent=2)

    with open(os.path.join(BASE_DIR, "tools/stress_ports.json"),
'w') as f:
        json.dump(list(machine_port_map.keys()), f)

    print(f"Environment ready in {TEST_ENV_DIR}")
    print(f"Server config updated with {len(all_tokens)}
tokens.")

if __name__ == "__main__":
    setup_stress_test()
```

Файл simulator.py

```
import time
import random
import struct
import json
import os
from pyModbusTCP.server import ModbusServer

def float_to_registers(f_val):
    packed = struct.pack('>f', f_val)
    reg1 = struct.unpack('>H', packed[0:2])[0]
    reg2 = struct.unpack('>H', packed[2:4])[0]
    return [reg1, reg2]

ports_file = os.path.join(os.path.dirname(__file__),
"stress_ports.json")
if not os.path.exists(ports_file):
    print("Error: stress_ports.json not found. Run
orchestrator.py first.")
    exit(1)

with open(ports_file, 'r') as f:
    ports = json.load(f)

servers = []
```

```
print(f"Starting {len(ports)} Modbus servers...")

for port in ports:
    srv = ModbusServer("127.0.0.1", port, no_block=True)
    servers.append(srv)
    srv.start()

print(f"All {len(ports)} simulators are online. Press Ctrl+C to
stop.\n")

try:
    parts_counter = 0
    while True:
        for srv in servers:
            port = srv.port

            if port == 5000:
                temp = int(45 + random.uniform(-2, 8))
                rpm = int(8000 + random.uniform(-500, 500))
                vibration = 0.5 + random.uniform(0, 0.3)
                oil = 4.2 + random.uniform(-0.2, 0.2)
                flow = 12.5 + random.uniform(-1, 1)
                load = int(65 + random.uniform(-10, 10))
                current = 15.2 + random.uniform(-1, 1)
                x = 120.5 + random.uniform(-0.1, 0.1)
                y = 45.2 + random.uniform(-0.1, 0.1)
                z = -12.3 + random.uniform(-0.05, 0.05)

                if random.random() > 0.95: parts_counter += 1
                error = 0 if random.random() > 0.98 else 0

                srv.data_bank.set_holding_registers(100, [temp])
                srv.data_bank.set_holding_registers(101, [rpm])
                srv.data_bank.set_holding_registers(102,
float_to_registers(vibration))
                srv.data_bank.set_holding_registers(104,
float_to_registers(oil))
                srv.data_bank.set_holding_registers(106,
float_to_registers(flow))
                srv.data_bank.set_holding_registers(108, [load])
                srv.data_bank.set_holding_registers(109,
float_to_registers(current))
                srv.data_bank.set_holding_registers(111,
float_to_registers(x))
                srv.data_bank.set_holding_registers(113,
float_to_registers(y))
                srv.data_bank.set_holding_registers(115,
float_to_registers(z))
                srv.data_bank.set_holding_registers(117,
[parts_counter % 65535])
                srv.data_bank.set_holding_registers(118,
[error])
            else:
```

```
                temp = int(40 + random.uniform(-5, 15))
                load = 50.0 + random.uniform(-10, 20)

                srv.data_bank.set_holding_registers(100, [temp])
                srv.data_bank.set_holding_registers(102,
float_to_registers(load))

        time.sleep(1)

except KeyboardInterrupt:
    print("\nShutting down simulators...")
    for srv in servers:
        srv.stop()
    print("Done.")
```

Файл start_stress_test.sh

```
#!/bin/bash

BASE_DIR="/home/baril/Projects/Argus"
TEST_ENV_DIR="$BASE_DIR/test_env"

echo "Starting Stress Test Runner..."

export LD_LIBRARY_PATH="$BASE_DIR/build/bin:$LD_LIBRARY_PATH"
cd "$BASE_DIR/build/bin"
./argus_server > "$BASE_DIR/server_stress.log" 2>&1 &
SERVER_PID=$!
echo "Server started (PID: $SERVER_PID). Logging to
server_stress.log"

cd "$BASE_DIR"
python3 tools/simulator.py > /dev/null 2>&1 &
SIM_PID=$!
echo "Simulator started (PID: $SIM_PID)"

sleep 3

echo "Launching 20 clients..."
for i in {1..20}
do
    CLIENT_DIR="$TEST_ENV_DIR/client_$i"
    cd "$CLIENT_DIR"
    export LD_LIBRARY_PATH="$CLIENT_DIR:$LD_LIBRARY_PATH"
    ./argus_client > /dev/null 2>&1 &
    echo "  - Client $i started"
done

echo "--- Stress test is RUNNING ---"
echo "Press Enter to STOP ALL processes..."
read
```

```
kill $SERVER_PID
kill $SIM_PID
killall argus_client
echo "All processes stopped."
```