Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – магістр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка клієнтського застосунку автоматизованої системи неперервного зважування гірничої маси в умовах ГЗК»

Виконала студентка гр. КН-23м _____ Кондеров А.С.

Керівник _____ Бугра А.В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« *11* » *липня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентці групи ЗКН-24м *Кондерову Артему Сергійовичу*

1. **Тема кваліфікаційної роботи:** *«Розробка клієнтського застосунку автоматизованої системи неперервного зважування гірничої маси в умовах ГЗК»*

затверджено наказом по університету № *594с* від *11.07.2025 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 92 с., додатки, презентація у Microsoft PowerPoint (12 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Бугра А.В.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*21.07.25*
2	*Розділ 1*	*18.08.25*
3	*Розділ 2*	*22.09.25*
4	*Розділ 3*	*20.10.25*
5	*Висновки*	*03.11.25*
6	*Оформлення кваліфікаційної роботи*	*17.11.25*
7	*Підготовка презентації та графічного матеріалу*	*25.11.25*
8	*Підготовка доповіді до захисту*	*01.12.25*

6. Дата видачі завдання: *28.06.2024 р.*

Керівник _____ /Бугра А. В./

7. Запевнення: *Я, Кондеров Артем Сергійович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Кондеров А. С./

АНОТАЦІЯ

Кондеров А.С. Розробка клієнтського застосунку автоматизованої системи неперервного зважування гірничої маси в умовах ГЗК.

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Комп'ютерні науки» зі спеціальності 122 – Комп'ютерні науки. – Криворізький національний університет, Кривий Ріг, 2025.

Робота складається зі вступу, трьох розділів, висновків, переліку використаної літератури з 31 позицій. Загальний обсяг роботи становить 92 сторінок, з яких основний зміст роботи викладено на 76 сторінках, включає 41 рисунок.

Кваліфікаційна робота присвячена розробці програмного забезпечення системи неперервного зважування гірничої маси на стрічкових конвеєрах із використанням промислового стандарту OPC UA та сучасних веб-технологій. Актуальність дослідження зумовлена потребою гірничо-збагачувальних підприємств у точному та оперативному контролі масових потоків, підвищенні рівня автоматизації та інтеграції технологічних даних у цифрові платформи.

У роботі проаналізовано сучасні апаратні та програмні рішення для систем неперервного зважування, виявлено їхні обмеження та визначено вимоги до нової системи. Розроблено архітектуру клієнт–серверної системи на основі технологій ASP.NET Core, MS SQL Server та React, яка забезпечує збір, зберігання та візуалізацію даних у реальному часі. Реалізовано фоновий OPC UA-клієнт для періодичного зчитування параметрів ваги і швидкості конвеєра, REST API для роботи з історичними даними та SignalR-канал для передачі потокових даних. Розроблено веб-застосунок для відображення технологічних параметрів, побудови графіків та аналізу статистики.

Проведено практичну апробацію системи, підтверджено її працездатність, стабільність і відповідність функціональним вимогам.

Ключові слова:

OPC UA, КОНВЕЄРНІ ВАГИ, НЕПЕРЕРВНЕ ЗВАЖУВАННЯ, ASP.NET CORE, REACT, SIGNALR, ПРОМИСЛОВА АВТОМАТИЗАЦІЯ

ANNOTATION

Konderov A.S. Development of a client application for an automated system for continuous weighing of rock mass for a mining and processing plant.

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program «Computer science» in specialty 122 – «Computer science». – Kryvyi Rih National University, Kryvyi Rih, 2025.

The work consists of an introduction, three sections, conclusions, a list of used literature from 31 items. The total volume of the work is 92 pages, of which the main content of the work is set out on 76 pages, includes 41 figures.

The thesis is devoted to the development of software for a continuous weighing system used to measure bulk material on conveyor belts, based on the OPC UA industrial communication standard and modern web technologies. The relevance of the research arises from the need of mining and beneficiation plants for accurate, real-time monitoring of material flow, increased automation, and seamless integration of technological data into digital platforms.

The work analyzes contemporary hardware and software solutions for continuous weighing systems, identifies their limitations, and formulates requirements for a new software architecture. A client–server system based on ASP.NET Core, MS SQL Server, and React was designed to collect, store, and visualize technological data in real time. The developed solution includes a background OPC UA client for periodic acquisition of conveyor weight and speed parameters, a REST API for accessing historical data, and a SignalR channel for streaming updates. A web application was implemented to display real-time values, generate dynamic charts, and perform statistical analysis.

Practical testing confirmed the system’s correctness, stability, and compliance with the specified functional requirements.

Keywords:

OPC UA, CONVEYOR SCALES, CONTINUOUS WEIGHING, ASP.NET CORE, REACT, SIGNALR, MS SQL SERVER, INDUSTRIAL AUTOMATION

ЗМІСТ

ВСТУП

Сучасні гірничо-збагачувальні підприємства функціонують в умовах зростаючих вимог до точності технологічних операцій, енергоефективності та оптимізації виробничих процесів. Одним з ключових елементів технологічного циклу є контроль масових потоків матеріалу, що транспортується конвеєрними лініями. Від достовірності та своєчасності даних про вагу та швидкість транспортування залежать ефективність роботи збагачувальних фабрик, коректність дозування сировини, якість кінцевої продукції та стабільність виробничого процесу в цілому. Саме тому системи неперервного зважування є критично важливими для забезпечення технологічної дисципліни та підвищення рівня автоматизації виробництва.

Попри значний розвиток апаратних засобів, програмне забезпечення, що супроводжує промислові вагові системи, часто характеризується обмеженою гнучкістю, закритими протоколами, недостатньою інтегрованістю з корпоративними платформами та обмеженими можливостями аналітики. Особливої уваги потребує питання оперативного доступу до даних у реальному часі, що є важливим для диспетчерського контролю та оптимізації виробничих рішень. Використання застарілих або монолітних систем ускладнює модернізацію виробничих ліній, впровадження цифрових технологій та перехід до концепцій Індустрії 4.0.

У цих умовах актуальною є розробка відкритої, масштабованої та технологічно сучасної інформаційної системи, яка забезпечує збір, зберігання, оброблення та візуалізацію даних конвеєрних ваг у реальному часі. Особливого значення набуває застосування стандарту OPC UA як універсального протоколу взаємодії промислового обладнання з цифровими платформами, а також використання сучасних веб-технологій для надання оператору гнучких і зручних засобів моніторингу.

Метою даної дипломної роботи є розроблення програмного забезпечення системи неперервного зважування гірничої маси на базі протоколу OPC UA та

сучасних веб-технологій, яке забезпечує оперативний доступ до технологічних параметрів, їх довгострокове зберігання та аналітичну обробку.

Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

– здійснити аналіз сучасних апаратних і програмних рішень для систем неперервного зважування та визначити їхні технічні особливості й недоліки;

– сформувати функціональні та нефункціональні вимоги до програмного забезпечення;

– розробити архітектуру системи на основі стандарту OPC UA, технологій ASP.NET Core, MS SQL Server та React;

– реалізувати серверну частину, яка забезпечує підключення до OPC UA, періодичне зчитування даних, їх збереження та надання інтерфейсів для доступу;

– розробити клієнтський веб-застосунок з підтримкою моніторингу даних у реальному часі та динамічної візуалізації історичних вибірок;

– здійснити тестування та практичну апробацію системи, підтвердивши її працездатність у типових виробничих умовах;

– провести аналіз безпеки, визначити рекомендації щодо захищеного розгортання та експлуатації.

Об’єктом дослідження є процес неперервного зважування гірничої маси на стрічкових конвеєрах.

Предметом дослідження є методи та засоби цифрової обробки, збереження та представлення даних вагових систем у реальному часі.

Наукова новизна роботи полягає в адаптації та інтеграції сучасних веб-технологій до промислових систем контролю масових потоків, побудові відкритої архітектури взаємодії гірничого обладнання з інформаційними сервісами та застосуванні OPC UA як єдиного каналу уніфікованого доступу до вимірювальних даних.

Практичне значення отриманих результатів полягає у створенні повноцінного прототипу програмного забезпечення, придатного до

використання в умовах гірничо-збагачувального підприємства для оперативного контролю технологічних параметрів та підвищення ефективності виробничих процесів. Запропонована система може бути інтегрована у SCADA-рівень автоматизації або у корпоративні інформаційні системи для подальшої аналітики.

Таким чином, виконана робота забезпечує комплексне вирішення актуальної задачі цифровізації вагових систем та демонструє можливість впровадження сучасних відкритих технологій для моніторингу й аналізу виробничих процесів у режимі реального часу.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ МОЖЛИВИХ ПІДХОДІВ ДО ВИРІШЕННЯ ЗАВДАННЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ЗВАЖУВАННЯ ГІРНИЧОЇ МАСИ

1.1 Коротка характеристика систем неперервного зважування гірничої маси на конвеєрній стрічці

Системи неперервного зважування матеріалів на стрічкових конвеєрах відіграють важливу роль у промислових виробництвах, де необхідно забезпечити точний контроль масової витрати сировини у режимі реального часу. Такими галузями є гірничодобувна, металургійна, цементна, енергетична, хімічна промисловість та інші виробництва, де матеріали транспортуються у сипкому стані. Конвеєрні ваги забезпечують оперативний контроль ресурсоємних технологічних процесів, підвищують точність обліку, сприяють оптимізації витрат та покращують керування потоками матеріалів.

Поряд з апаратною складовою, ключову роль відіграє програмне забезпечення – від алгоритмів обробки сигналів у ПЛК до інтеграції з інформаційними системами верхнього рівня за допомогою сучасних індустріальних протоколів, таких як OPC DA/UA, MQTT тощо [1-3]. Розвиток інформаційних технологій та промислових стандартів відкриває можливість створення більш гнучких, масштабованих і відкритих систем збору та обробки даних.

Конвеєрні ваги є вимірювальними системами, що визначають:

– миттєву масову витрату матеріалу (т/год);

– нормоване лінійне навантаження на конвеєр (кг/м),

– швидкість руху стрічки,

– накопичену масу матеріалу, що транспортується.

Принцип роботи базується на одночасному вимірюванні:

– вагового навантаження, яке створює матеріал на ролики або спеціально виділену секцію вагового моста;

– лінійної швидкості стрічки конвеєра, що вимірюється енкодером або тахогенератором.

Обчислення масової витрати визначається співвідношенням:

$$\dot{m} = q \cdot v \ , \tag{1.1}$$

де q – лінійне навантаження (кг/м), v – швидкість стрічки (м/с).

Накопичена маса визначається інтегруванням масової витрати за часом.

Стандартна конвеєрна вагова система складається з:

– вагового модуля з тензодатчиками та механічною конструкцією;

– енкодера швидкості або тахогенератора;

– аналого-цифрових перетворювачів, що підключаються до контролера;

– ПЛК або спеціалізованого вагового контролера;

– інтерфейсних модулів для передачі даних (Modbus, Profinet, Profibus, CAN, Ethernet/IP, OPC UA);

– програмного забезпечення локального або віддаленого моніторингу.

У сучасних системах усе частіше застосовуються IoT-шлюзи для передачі даних у хмарні сервіси або SCADA.

1.2 Огляд апаратних рішень для систем конвеєрного зважування

У промислових системах транспортування та дозування сипких матеріалів апаратні рішення відіграють ключову роль, оскільки саме вони формують первинні вимірювальні сигнали та забезпечують можливість їх подальшої обробки у контролерах або індустріальних інформаційних системах. Для сучасних конвеєрних ваг виробники пропонують як спеціалізовані вагові контролери, так і універсальні ПЛК із відповідними модулями вводу/виводу. Кожне з цих рішень характеризується власною архітектурою, точністю, стабільністю вимірювань, а також різними можливостями інтеграції.

Найбільш відомими та популярними спеціалізованими ваговими контролерами є:

– Siemens Milltronics BW500, BW100, SIWAREX WT231/241;

– Schenck Intecont;

– Thermo Scientific Ramsey Micro-Tech;

– Tecweigh WT10 / WT18.

Продукція Siemens у сфері конвеєрного зважування охоплює два ключові напрями: окремі вагові контролери Milltronics (зокрема BW500 та BW100) та модулі ваговимірювальних систем сімейства SIWAREX, інтегровані у платформу SIMATIC.

Вимірювальні контролери Milltronics (рис. 1.1) є автономними спеціалізованими приладами, оснащеними аналоговими входами для тензодатчиків, каналами для отримання сигналів швидкості, а також численними інтерфейсами обміну даними [4].

Рисунок 1.1 – Конвеєрний ваговимірювальний пристрій на базі контролера Milltronics

Контролери Milltronics характеризуються високою надійністю та точністю завдяки спеціалізованим алгоритмам цифрової фільтрації, автоматичним корекціям нестабільності стрічки та вбудованим засобам діагностики. У промисловій практиці такі контролери ефективні при модернізації існуючих конвеєрних ліній, коли необхідно оновити лише вимірювальну частину без втручання у загальну систему керування.

Модулі SIWAREX WT231 та WT241 [5], на відміну від Milltronics, працюють у складі контролерів Siemens S7-1200 або S7-1500, що дозволяє інтегрувати функції зважування безпосередньо у логіку ПЛК. Такий підхід надає значно більше свободи у побудові алгоритмів, особливо у випадках, коли вагова система є частиною складного технологічного процесу. Можливості інтеграції SIWAREX у TIA Portal також спрощують налагодження та діагностику. Водночас залежність від екосистеми Siemens є обмежувальним чинником: обладнання є порівняно дорогим, а можливості інтеграції з іншими платформами безпосередньо залежать від наявності додаткових комунікаційних модулів.

Основними недоліками рішень Siemens на даний момент є висока вартість комплектуючих та ліцензій, складність тонкого налаштування при роботі в умовах значних механічних вібрацій, а також часткова закритість комунікаційних протоколів у старших моделях. Водночас їхня надійність, точність та стійкість до промислових завад роблять ці системи одними з найбільш розповсюджених на європейських промислових підприємствах.

Німецька компанія Schenck Process є одним із лідерів у сфері динамічного зважування і пропонує контролери Intecont [6], які застосовуються у високонавантажених технологічних процесах. Контролери даного класу вирізняються дуже високою точністю вимірювань, яку забезпечує багаторівнева цифрова фільтрація та автоматичні системи компенсації механічної нестабільності. Особливістю Intecont є можливість роботи з кількома каналами вимірювання, що дозволяє організовувати облік одночасно кількох потоків матеріалів, а також використовувати їх у складних задачах дозування (рис. 1.2).

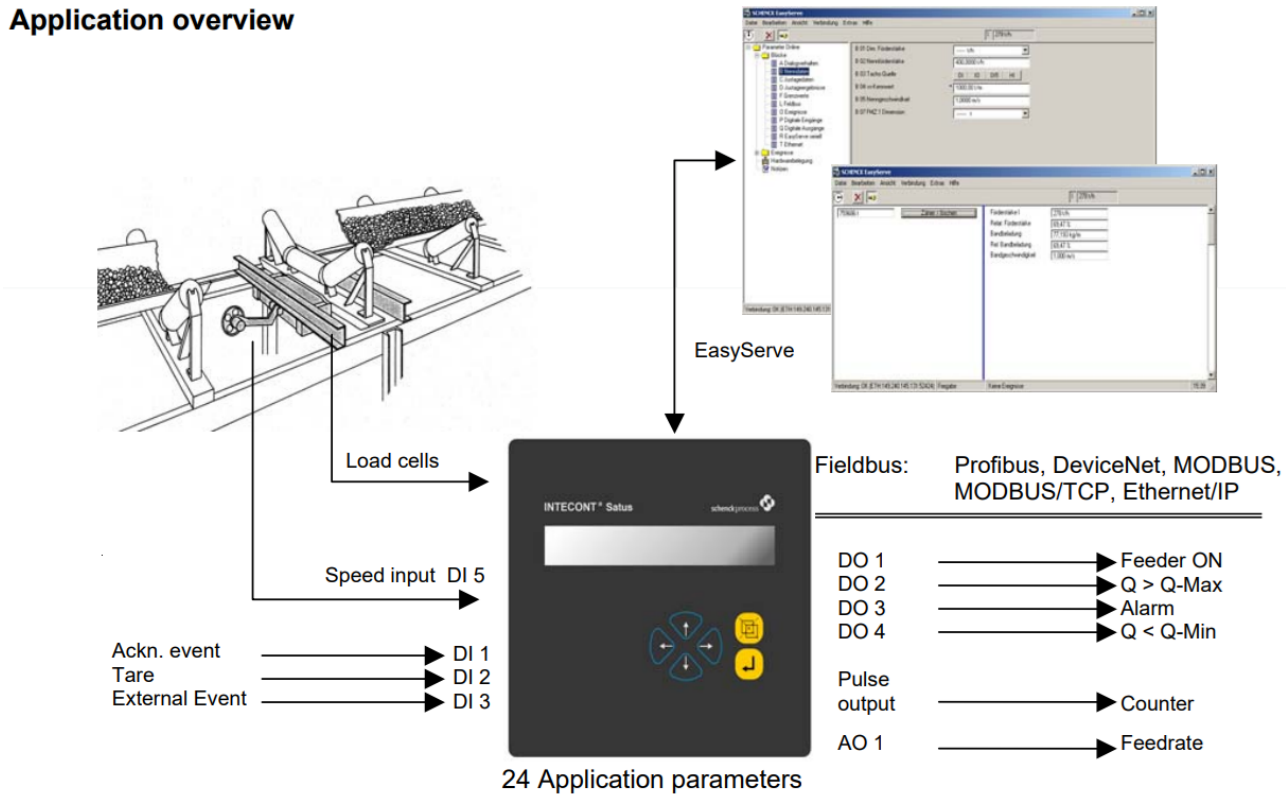

Рисунок 1.2 – Конвеєрний ваговимірювальний пристрій на базі контролера Schenck Intecont Satus

На практиці Intecont часто застосовують на підприємствах, де важливим параметром є довготривала стабільність вимірювань, наприклад у виробництві цементу, металургії та гірничодобувних кар'єрах. До переваг можна віднести також широкий спектр інтерфейсів взаємодії з промисловими мережами, включно з Profibus, Profinet, Modbus та Ethernet TCP/IP. У той же час програмне забезпечення Schenck є досить закритим, що ускладнює глибоку інтеграцію із зовнішніми користувацькими системами та не дозволяє розширювати функціональність за межі пропонованих інструментів.

Серед недоліків можна виділити високу вартість, необхідність використання фірмового обладнання для певних модифікацій, а також залежність точності від правильного механічного монтажу, який повинен виконуватися сертифікованими спеціалістами. Проте у випадках, коли вимоги до метрологічних показників є критично високими, системи Intecont залишаються одним із найкращих доступних варіантів.

Контролери серії Ramsey Micro-Tech від компанії Thermo Scientific (рис. 1.3) традиційно використовуються у гірничодобувній промисловості Північної Америки та інших регіонах із високими вимогами до надійності в умовах підвищеної запиленості та вібрацій [7]. Дані контролери виділяються збалансованим поєднанням функціональності, точності та стійкості до зовнішніх впливів. Їх конструкція орієнтована на роботу в екстремальних умовах, що визначає їх актуальність для підприємств із важкими технологічними режимами.

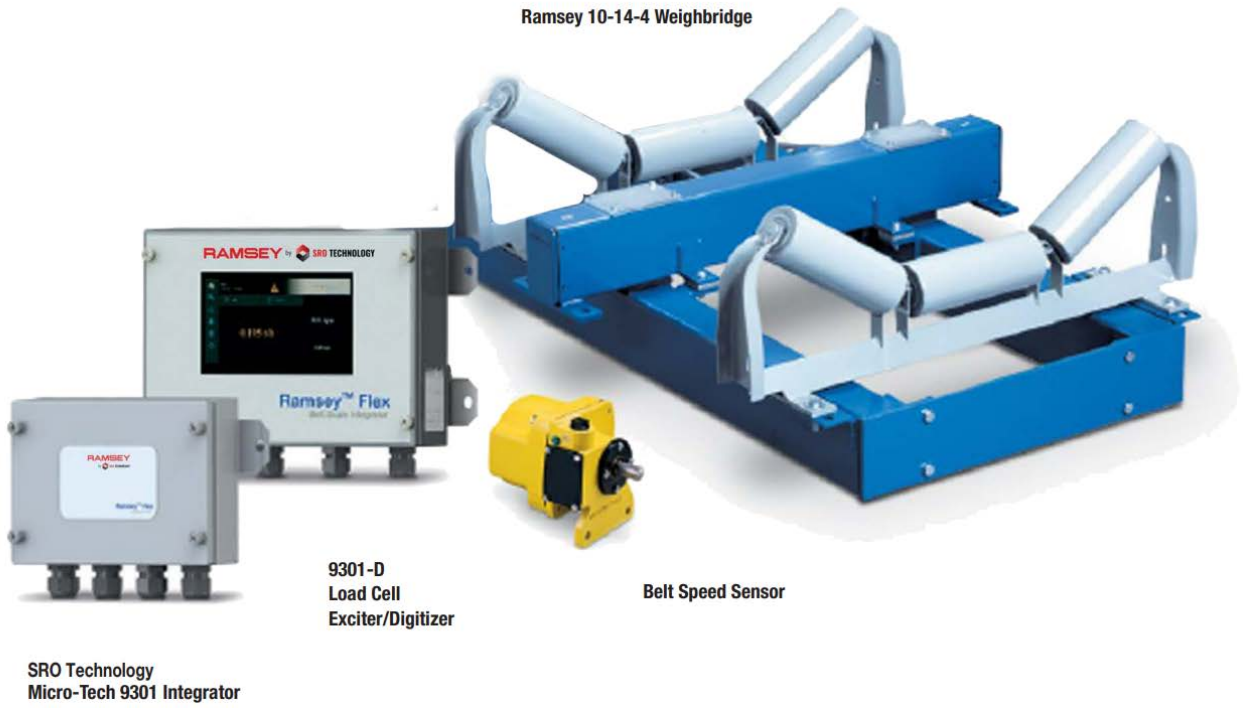

Рисунок 1.3 – Конвеєрний ваговимірювальна система на базі контролера Ramsey Micro-Tech

Однією з переваг Ramsey є широкі можливості налаштування, включно з адаптивною фільтрацією, автоматичними режимами компенсації температури та можливістю підключення до різних типів тензодатчиків [8]. Значним плюсом є також наявність розвиненої системи самодіагностики, що дозволяє швидко виявляти несправності на ранній стадії. З іншого боку, архітектура Ramsey дещо поступається конкурентам у плані відкритості та гнучкості інтеграції зі сторонніми програмними платформами. Крім того, користувачі відзначають складність конфігурації для нетипових сценаріїв експлуатації, що вимагає

участі інженерів виробника.

У загальному випадку контролери Thermo Scientific можна охарактеризувати як рішення, оптимізоване під високі навантаження та важкі умови експлуатації, проте менш універсальне у плані інтеграції та розширення функціональності порівняно з контролерами Siemens або Schenck.

Американська компанія Tecweigh пропонує конвеєрні ваги WY10, WY15 та контролери WP50, MS50 (рис. 1.4), які орієнтовані на сегмент середнього бюджету і широко використовуються у невеликих та середніх виробництвах, де вимоги до точності є помірними, а ключовим параметром стає співвідношення ціни та функціональності [9]. Ці рішення забезпечують базовий набір функцій для обчислення масової витрати та накопиченої маси і характеризуються достатньою точністю для більшості типових задач зважування.

Рисунок 1.4 – Конвеєрні ваги Tecweigh WY15

Перевагою ваговимірювальної продукції компанії Tecweigh є простота, невисока вартість та легкість у налаштуванні. Завдяки цьому вони часто використовуються для автоматизації невеликих ліній у харчовій, аграрній або будівельній галузях. Проте такі контролери мають обмежені можливості інтеграції, а реалізовані комунікаційні інтерфейси зазвичай покривають лише

найпоширеніші та найпростіші протоколи на кшталт Modbus RTU/TCP. Відсутність розвиненого інструментарію для багаторівневої діагностики та обробки даних робить їх менш придатними для складних технологічних процесів або систем, які вимагають високої точності у динаміці.

Таким чином, продукцію компанії Tecweigh можна розглядати як економічне, але функціонально обмежене рішення, придатне для простих або допоміжних вагових задач.

У сучасних системах неперервного зважування гірничої маси дедалі більшої поширеності набуває використання програмованих логічних контролерів загального призначення, зокрема сімейства Siemens S7 [10], у поєднанні з ваговими модулями серії Siwarex (рис. 1.5).

Рисунок 1.5 – Елементи вагової платформи на базі модулів Siemens Siwarex та ПЛК серії Simatic S7

Таке рішення розглядається як компроміс між функціональністю спеціалізованих вагових контролерів і гнучкістю відкритих платформ автоматизації. ПЛК загального призначення з одного боку забезпечують достатні обчислювальні ресурси, розвинуті можливості циклічної та подійно-орієнтованої логіки, а також широке коло підтримуваних протоколів

промислової комунікації. З іншого боку, вагові модулі Siwarex виконують функції високоточного аналого-цифрового перетворення сигналів тензодатчиків, що робить подібні гібридні системи здатними забезпечувати метрологічні вимоги до промислових вагових процесів.

Функціональні можливості такого підходу значною мірою визначаються взаємодією між ПЛК та модулем Siwarex. Модуль виступає у ролі спеціалізованого A/D-перетворювача з високою роздільною здатністю, автоматичною температурною компенсацією та реалізованими всередині алгоритмами цифрової фільтрації. Обчислення основних вагових параметрів, таких як миттєва маса, фільтроване значення, статичне дозування або обчислення потоку матеріалів, можуть виконуватися як на стороні модуля, так і на рівні ПЛК, залежно від конкретної конфігурації. Такий розподіл навантаження дає змогу забезпечити оптимальний компроміс між точністю вимірювання та швидкодією загальної системи керування.

Важливим аспектом використання Siemens S7 у вагових застосуваннях є їхня інтегрованість у середовище промислової автоматизації. Контролери S7-1200 та S7-1500 підтримують широкий спектр протоколів, включно з Profinet, Modbus TCP, OPC UA та, за необхідності, MQTT. Це дозволяє реалізовувати не лише локальний контроль, а й інтеграцію з SCADA-системами, MES-рівнем та корпоративними платформами моніторингу. У порівнянні зі спеціалізованими ваговими контролерами, рішення на базі ПЛК забезпечують значно кращу гнучкість щодо побудови складних алгоритмів логіки, сценаріїв оптимізації транспортування матеріалу, а також підключення високорівневих модулів аналітики.

Сильним боком таких систем є їхня масштабованість. За необхідності можна легко розширити кількість вагових каналів, додати модулі вводу-виводу, підключити датчики стану конвеєра або інтегрувати систему з частотними перетворювачами, що регулюють швидкість конвеєрної стрічки. У багатьох випадках це дозволяє створювати інтегровані комплекси, у яких зважування стає частиною загального конвеєрного контролю і не обмежується лише

вимірюванням маси.

Попри очевидні переваги, використання ПЛК загального призначення у вагових системах пов’язане з певними обмеженнями. Найбільш суттєвим недоліком порівняно зі спеціалізованими ваговими контролерами є складність інженерного процесу. Конфігурація вагових модулів Siwarex, налаштування масштабування тензометричних мостів, підбір параметрів фільтрації та узгодження з циклічною логікою ПЛК вимагає високої кваліфікації інженера-автоматника. У випадках, коли підприємство не має відповідних компетенцій, підтримка такої системи може бути суттєво ускладненою. Крім того, деякі вагові функції, доступні у спеціалізованих контролерах «з коробки», у ПЛК-орієнтованому рішенні необхідно реалізовувати окремо, що збільшує обсяг проєктування та тестування.

На тлі класичних ПЛК, промислові IoT-шлюзи на кшталт Siemens IoT2040 або інших аналогічних платформ відіграють специфічну роль у сучасних системах зважування. Вони не замінюють контролер, але забезпечують значне розширення його функціональних можливостей у напрямку цифровізації та хмарної інтеграції. IoT-шлюзи здатні виконувати функції протокольної конвертації, локального попереднього аналізу даних, агрегації телеметрії та передачі результатів у зовнішні сервіси через MQTT, AMQP або REST-інтерфейси. Вони дозволяють інтегрувати вагові системи у сучасні платформи моніторингу та аналітики, створювати системи раннього виявлення відхилень, реалізувати алгоритми підвищеної точності на базі машинного навчання або фільтрації в реальному часі за участю Node-RED чи Python-скриптів.

Вибір IoT-шлюзу як складової системи неперервного зважування визначається, перш за все, потребою у підвищенні інформаційної прозорості виробництва. У середовищах, де важливе не лише локальне керування потоком матеріалів, а й передавання великого масиву технологічних даних у корпоративні або хмарні системи, IoT-шлюз виступає ключовою ланкою цифрової інфраструктури. Проте подібні рішення також мають характерні обмеження, зокрема залежність від стабільності мережевих каналів, порівняно

нижчу надійність у суворих промислових умовах та необхідність забезпечення кібербезпеки на більш високому рівні, ніж у традиційних ПЛК-системах.

У цілому апаратні рішення на базі ПЛК Siemens у поєднанні з модулями Siwarex і доповнені промисловими IoT-шлюзами формують гнучку та функціонально насичену інфраструктуру для систем неперервного зважування. Вони поєднують точність, що забезпечується апаратними ваговими модулями, з гнучкістю логічного керування та можливостями цифрової інтеграції, які забезпечують ПЛК та IoT-платформи. Такий підхід дає змогу адаптувати систему до специфічних умов виробництва, однак вимагає високого рівня інженерної культури та ретельного технологічного проєктування, щоб уникнути ризиків, пов’язаних зі складністю та багатокомпонентністю архітектури.

1.3 Огляд програмного забезпечення систем конвеєрного зважування

Програмне забезпечення, що призначене для супроводу промислових вагових систем, є ключовою складовою їхньої функціональності, надійності та здатності інтегруватися в масштабні цифрові екосистеми. У сучасній промисловій практиці ПЗ розвивається у двох основних напрямках: з одного боку, це вбудовані програмні комплекси, інтегровані у спеціалізовані вагові контролери або модулі типу Siwarex, а з іншого – програмні платформи та додатки верхнього рівня, які забезпечують моніторинг, діагностику, передавання даних і аналітичну обробку. Таке двошарове програмне забезпечення формує об’єднану цифрову інфраструктуру, у якій вимірювання маси стає частиною ширших процесів керування, планування, оптимізації виробництва та забезпечення можливостей відстежування матеріальних потоків.

Вбудоване ПЗ вагових контролерів традиційно орієнтоване на виконання базових вимірювальних та фільтраційних функцій. У більшості промислових пристроїв воно містить попередньо визначені алгоритми цифрової обробки

сигналів тензометричних датчиків, компенсаційні механізми, функції автоматичного нульового балансу та калібрування, а також засоби визначення потоку матеріалів для конвеєрних систем. Таке ПЗ відзначається високою стабільністю та надійністю роботи в реальному часі, оскільки його основним завданням є безперервне отримання та обробка сигналів із мінімальною затримкою. Проте функціонал, що надається такими системами, зазвичай обмежений рамками вимірювання і не передбачає розширеної інтеграції або можливостей адаптації алгоритмів до специфічних потреб конкретного виробництва. У багатьох випадках розробники спеціалізованих вагових контролерів надають лише мінімальні інструменти для конфігурації, що знижує гнучкість системи, проте гарантує її метрологічну стабільність.

У противагу обмеженням вбудованих рішень програмне забезпечення, орієнтоване на ПЛК та промислові IoT-платформи, характеризується значно більшою свободою конфігурації. У системах, побудованих на основі Siemens S7 та модулів Siwarex, логіка обробки вимірювань реалізується переважно на рівні середовища програмування TIA Portal. Це відкриває можливості створення складних алгоритмів обробки, використання структурованого програмування, реалізації паралельних процесів, а також глибокої інтеграції із сигналами конвеєрної автоматики. У таких системах програмне забезпечення фактично виконує роль посередника між первинним вимірювальним процесом та промисловою логікою, при цьому ступінь адаптивності алгоритмів значно вищий, ніж у традиційних вагових контролерів. Проте висока гнучкість супроводжується збільшенням залежності якості роботи системи від кваліфікації інженера, адже розробник ПЗ не лише формує алгоритми, а й відповідає за дотримання метрологічних вимог, стабільність циклічної роботи та коректність реалізованих фільтраційних механізмів.

Ще одним важливим аспектом є програмне забезпечення вищих рівнів автоматизації, що забезпечує моніторинг, архівування та аналітичну інтерпретацію зважувальних даних. Сучасні SCADA-системи, зокрема WinCC, Ignition, Zenon або інші промислові платформи [11], надають можливості

безперервного збору телеметрії, візуалізації трендів, генерування звітності та реалізації систем контролю якості. У контексті конвеєрних ваг такі функції стають критично важливими, оскільки обсяг переданих даних значний, а зміни у потокових значеннях маси потребують безперервної візуалізації та збереження історії. Програмні комплекси SCADA дозволяють створювати багаторівневу систему сигналізації, виявляти відхилення від технологічних норм, а також інтегрувати результати вимірювань у корпоративні інформаційні системи. Водночас використання таких платформ потребує розробки додаткових комунікаційних модулів, налаштування протоколів обміну та забезпечення відповідності вимогам кібербезпеки.

Широкого поширення набули також IoT-орієнтовані програмні додатки, що функціонують на шлюзах типу Siemens IoT2040, Advantech або Beckhoff IoT. Подібні системи використовують Node-RED, контейнеризовані сервіси або Python-движки для потокової обробки та передавання даних через MQTT, OPC UA або REST API. На відміну від класичних SCADA-рішень, IoT-платформи орієнтовані не лише на візуалізацію, а й на гнучку агрегацію даних, їхню трансформацію та подальшу передачу у хмарні системи аналітики. У середовищах, де важливо реалізувати інтеграцію вагових систем зі службами прогнозної діагностики, машинного навчання або цифрових двійників, саме програмне забезпечення IoT-шлюзів забезпечує найкращі можливості для адаптації. Проте такі системи характеризуються меншою метрологічною визначеністю, а їх ефективність значною мірою залежить від стабільності промислових мереж і якості програмної реалізації потокових сценаріїв.

Важливим трендом є поширення універсальних серверів даних, зокрема рішень на базі OPC UA [1-3]. Програмне забезпечення таких серверів дозволяє структурувати дані з різних частин системи, реалізувати єдину модель інформації та забезпечити незалежний від платформи доступ до вимірювань. У вагових системах OPC UA стає ключовим елементом для побудови архітектур, що підтримують промислову інтероперабельність, особливо коли використовується комбінація ПЛК, вагових модулів і IoT-шлюзів. Програмна

реалізація OPC UA-серверів або клієнтів дозволяє досягти високого рівня гнучкості, але потребує чіткого дотримання стандартів щодо структури адресного простору та обробки подій, а також достатньої продуктивності обладнання.

Таким чином, програмне забезпечення сучасних вагових систем формує складний багаторівневий технологічний контур, у якому кожний рівень виконує власну функцію — від високоточного збору сигналів до глибокої цифрової інтеграції та аналітики. Спеціалізовані вбудовані системи гарантують стабільність вимірювання, ПЛК-орієнтовані рішення забезпечують гнучкість і просторову масштабованість, SCADA-платформи – повноцінну інтеграцію в автоматизовані виробничі процеси, а IoT-шлюзи – можливість включення вагових систем у глобальні цифрові інфраструктури. Разом вони формують сучасний підхід до організації промислових зважувальних процесів, у якому на перший план виходять не лише точність та надійність вимірювання, але й забезпечення комплексної інформаційної прозорості виробництва.

1.4 Обмеження існуючих програмних рішень і передумови створення власного програмного забезпечення

Попри значний прогрес у сфері автоматизації процесів неперервного зважування, сучасні промислові вагові комплекси й надалі характеризуються низкою суттєвих обмежень, пов’язаних насамперед з особливостями реалізації програмного забезпечення. Виробники прагнуть створювати універсальні програмні платформи, однак така універсальність часто досягається ціною зниження гнучкості та можливостей адаптації алгоритмів до специфічних технологічних умов конкретного підприємства. Саме програмний компонент є визначальним для ефективності вимірювального комплексу, оскільки саме він забезпечує логіку обробки сигналів, передачу технологічних даних, інтеграцію з іншими елементами автоматизованої системи та можливість масштабування в майбутньому.

Одним із центральних обмежень, властивих більшості комерційних вагових рішень, є закритість програмного середовища. Виробники контролерів або спеціалізованих вагових модулів надають користувачам лише мінімально необхідний функціонал, який дозволяє виконувати налаштування, калібрування та базову діагностику. За межами цього набору функцій модифікація алгоритмів стає неможливою. Будь-яке розширення інтерфейсів, додавання нових каналів обміну даними чи інтеграція із сучасними промисловими платформами потребує використання дорогих ліцензій або спеціальних модулів розширення, що суттєво збільшує сумарну вартість системи. Через це промислові підприємства часто стикаються з проблемою обмеженої адаптивності, коли ваговий комплекс стає жорстко прив’язаним до конкретної апаратної конфігурації та не може бути модернізований відповідно до поточних потреб виробництва.

Ще одним аспектом, який обмежує потенціал існуючих рішень, є фрагментованість архітектур на рівні обміну даними. Значна частина сучасних вагових комплексів оперує протоколами, що забезпечують лише низькорівневий доступ до вимірювальних даних. Це можуть бути сторонні API, внутрішні комунікаційні протоколи виробника або базова реалізація Modbus/RTU чи Modbus/TCP. У такому підході дані передаються у вигляді числових регістрів без структурування інформаційної моделі. Інтеграція з системами верхнього рівня зазвичай вимагає додаткового перетворення даних, створення проміжних програмних прошарків або застосування сторонніх шлюзів, що не лише збільшує складність рішення, але й ускладнює забезпечення його довгострокової технічної підтримки.

Незважаючи на наявність рішень, які підтримують сучасніші протоколи, такі як OPC UA, і тут виявляється низка практичних обмежень. Реалізації OPC UA у спеціалізованих вагових контролерах часто мають мінімальну інформаційну модель, обмежену передаванням лише найпростіших параметрів, без можливості доступу до службових даних, діагностики або внутрішніх станів алгоритмів. Унаслідок цього навіть за наявності сучасного протоколу інтеграція

з системами промислового інтернету речей або з цифровими платформами промислової аналітики залишається неповною і вимагає додаткових програмних засобів.

Також важливо зазначити, що більшість готових вагових систем створювалася у парадигмі централізованого керування, орієнтованого переважно на локальне виконання технологічних функцій. У сучасних умовах така модель втрачає ефективність, оскільки виробничі процеси дедалі частіше потребують розподіленої та масштабованої архітектури, у якій кожний елемент обладнання взаємодіє не лише з локальним контролером, але й з периферійними IoT-вузлами, хмарними сервісами, цифровими двійниками та системами прогнозної діагностики. Існуюче програмне забезпечення у більшості випадків не забезпечує автоматичної публікації даних у розподілене середовище, не підтримує сучасних протоколів безпечної аутентифікації та не відповідає вимогам щодо модульності й розширюваності.

Окремим викликом є питання довгострокового зберігання та аналітичної обробки даних. Переважна частина комерційних програмних продуктів надає лише мінімальні функції архівування, які здебільшого обмежуються локальним збереженням даних у пропрієтарному форматі. Такий підхід значно ускладнює інтеграцію з корпоративними системами, а також створення історичних моделей для подальшої оптимізації виробничих процесів. В умовах сучасного промислового виробництва, де аналітика є ключовим інструментом підвищення продуктивності, подібне обмеження стає критичним.

Суттєве значення має також проблема кастомізації користувацького інтерфейсу та логіки взаємодії оператора з системою. Комерційні програмні продукти часто мають жорстко визначений графічний інтерфейс, який не враховує особливості конкретного виробничого процесу. Неможливість модифікації або розширення інтерфейсу призводить до появи додаткового навантаження на оператора і знижує ефективність роботи вагового комплексу, особливо у випадках, коли система експлуатується в складних умовах або вимагає швидкого реагування на зміну технологічної ситуації.

У сукупності зазначені обмеження створюють об’єктивні передумови для розроблення власного програмного забезпечення, яке має забезпечити високу гнучкість, модульність і відкритість архітектури. Перехід до використання сучасних методів обміну даними, таких як OPC UA, MQTT або REST API, відкриває можливість інтеграції вагової системи з широким спектром цифрових платформ. Застосування ASP.NET та сервісно-орієнтованих принципів розробки дозволяє створити систему, адаптовану до потреб конкретного підприємства та здатну працювати у масштабованому середовищі [12]. Власне програмне забезпечення також дає можливість реалізувати гнучкі механізми аналітики, адаптувати структуру бази даних, створити інтуїтивний інтерфейс користувача та забезпечити розширюваність рішення у відповідності до стратегічних потреб виробництва.

Таким чином, аналіз існуючих програмних рішень демонструє, що їх обмеження стають ключовим чинником, який стримує розвиток сучасних вагових систем у напрямку цифрової трансформації. Розроблення власного програмного забезпечення є не лише способом усунути наявні недоліки, але й стратегічною необхідністю, що дозволяє створити інноваційну інфраструктуру збору та обробки даних, орієнтовану на майбутні вимоги промислового виробництва.

1.5 Формування функціональних та нефункціональних вимог до програмного забезпечення системи неперервного зважування

Розробка програмного забезпечення для системи неперервного зважування гірничої маси потребує чіткого визначення функціональних та нефункціональних вимог, що забезпечують коректне відтворення усіх стадій життєвого циклу даних — від їх одержання та первинної обробки до довгострокового зберігання, аналітичного опрацювання та інтеграції з іншими елементами інформаційної інфраструктури підприємства. Сучасні вагові системи, що працюють у режимі реального часу, пред’являють високі вимоги

до точності, швидкодії, відмовостійкості та масштабованості програмних компонентів, тому формування вимог має спиратися на системну оцінку технологічних, апаратних та експлуатаційних особливостей такого обладнання.

Функціональні можливості програмного забезпечення мають забезпечувати безперервний прийом вимірювальних даних від промислового контролера або спеціалізованого вагового модуля з урахуванням необхідності стабільної роботи у режимі постійного потоку інформації. Завданням програмного забезпечення є не лише прийом "живих" показників, але і їх перетворення у структурований формат, придатний для тривалого зберігання та подальшого аналізу. Саме тому передбачається функціональна підтримка збереження вимірювань у реляційній або часовій базі даних, де кожне значення доповнюється часовою міткою та контекстною інформацією, що дозволяє відтворювати історичні стани процесу зважування.

Важливою складовою системи є механізм доступу до архівних даних, який реалізується через WebAPI. Цей інтерфейс забезпечує стандартизовану взаємодію між клієнтськими застосунками різного типу, зокрема веб-клієнтами, мобільними застосунками або зовнішніми корпоративними інформаційними системами. Сервіс має підтримувати можливість запиту історичних показників за певний часовий інтервал, що дозволяє реалізувати базові аналітичні сценарії, зокрема побудову тенденцій, виявлення відхилень, оцінювання продуктивності та контроль технологічних режимів. У свою чергу, клієнтські застосунки повинні мати інструменти візуалізації отриманих даних, що забезпечує формування графіків, таблиць та інших інформативних представлень архівної інформації.

Окремо слід підкреслити важливість надання користувачеві доступу до даних у режимі реального часу. Розв’язання цього завдання передбачає застосування веб-сокетів, які забезпечують двосторонній канал зв’язку між сервером та клієнтом і дозволяють миттєво передавати нові значення вимірювань без потреби періодичного опитування сервера. Використання цієї технології не лише зменшує навантаження на мережеву інфраструктуру, але і

суттєво покращує оперативність реагування оператора, що особливо важливо у випадках контролю технологічних параметрів у виробничих процесах із високою динамікою.

Важливим елементом функціональної моделі є підтримка фільтрації історичних даних за часовими параметрами. Система має забезпечувати коректну обробку запитів на отримання даних за довільні часові відрізки, що може охоплювати як короткі проміжки для локального аналізу, так і тривалі періоди для стратегічної оцінки продуктивності. Механізм фільтрації повинен враховувати можливі нерівномірності у надходженні вимірювань та гарантувати правильну вибірку навіть у випадках тимчасових пропусків або нестачі даних.

Наведені функціональні вимоги було формалізовано у вигляді діаграми варіантів використання UML, яку було візуалізовано з використанням онлайн-редактору PlantUML (рис. 1.6).

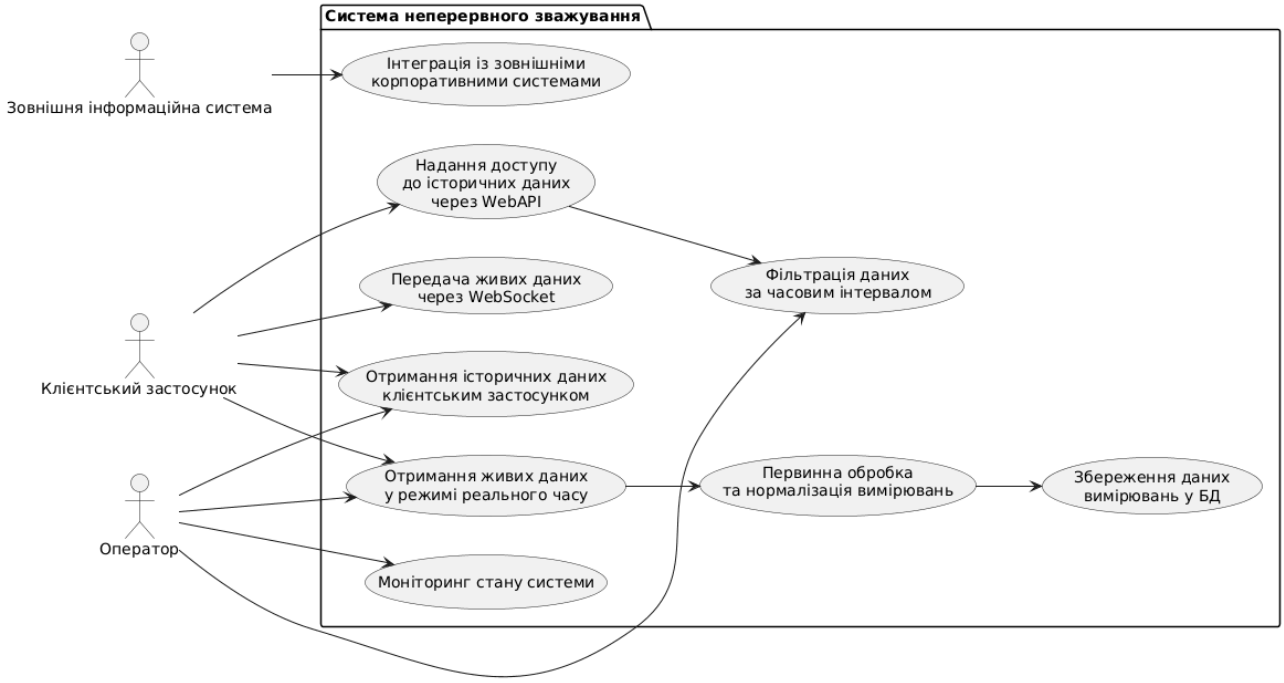

Рисунок 1.6 – Діаграма використання програмного забезпечення системи неперервного зважування

Поряд із функціональними можливостями, не менш важливими є нефункціональні вимоги, що визначають якість роботи програмного забезпечення. До них належать вимоги до продуктивності, які мають гарантувати коректну обробку високочастотних потоків даних, а також вимоги до масштабованості, що забезпечують можливість адаптації програмної архітектури до збільшення кількості вимірювальних каналів або зростання навантаження. Низький рівень затримки при передачі вимірювань, стійкість до збоїв та здатність до автоматичного відновлення після переривань зв'язку є критичними характеристиками для систем, що працюють у умовах реального виробництва.

Суттєву роль відіграють вимоги до надійності та безпеки, оскільки система працює з технологічними параметрами, що можуть впливати на виробничі процеси. Необхідно забезпечити контрольоване керування доступом, захищені канали передачі даних, механізми журналювання та моніторингу стану системи. Забезпечення безпеки має поєднуватися з вимогою до простоти інтеграції, що значною мірою визначається відкритими інтерфейсами взаємодії та модульністю архітектури.

У підсумку вимоги до програмного забезпечення системи неперервного зважування формують цілісну модель, що охоплює всі аспекти опрацювання вимірювань — від їх прийому та збереження до аналізу, візуалізації та інтеграції у корпоративні інформаційні процеси. Чітке визначення цих вимог забезпечує передумови для побудови стійкої, масштабованої та функціонально повноцінної програмної системи, здатної працювати у промислових умовах та задовольняти потреби сучасної промислової автоматизації.

Висновки до розділу:

У межах проведеного дослідження було здійснено комплексний огляд сучасних технічних та програмних рішень, що застосовуються у системах неперервного зважування гірничої маси на конвеєрних лініях. Аналіз поширених апаратних платформ засвідчив, що на ринку домінують дві стратегії

побудови вимірювальних систем: використання спеціалізованих вагових контролерів, орієнтованих на виконання вузькоспеціалізованих алгоритмів високої точності, та застосування модульних рішень на базі ПЛК загального призначення, доповнених ваговими модулями типу Siemens SIWAREX і промисловими IoT-шлюзами для розширення комунікаційних можливостей.

Встановлено, що перша група рішень забезпечує високий ступінь точності, низький рівень затримок і передбачуваність вимірювальних процедур завдяки глибокій апаратній оптимізації. Водночас така архітектура обмежує можливості інтеграції з корпоративними IT-ландшафтами та модернізації функціоналу. Платформи на базі ПЛК і IoT-шлюзів, навпаки, пропонують більшу гнучкість, розширюваність і адаптивність до складної інфраструктури підприємства. Вони забезпечують підтримку стандартів OPC UA, MQTT та промислових протоколів, що істотно спрощує інтеграцію з сучасними інформаційними системами, але їх точність та швидкодія можуть залежати від параметрів конфігурації та якості апаратної інтеграції.

Окрему увагу було приділено огляду програмного забезпечення вагових систем, яке охоплює як локальні SCADA-орієнтовані рішення, так і багаторівневі клієнт-серверні застосунки з підтримкою аналітики і потокової обробки. Показано, що актуальний тренд розвитку галузі полягає у поступовому переході від монолітних локальних систем до відкритих, мультиплатформних, API-орієнтованих архітектур, у яких особливе місце займають веб-технології, потокові протоколи та стандартизовані інтерфейси обміну даними.

На основі узагальнення цих результатів було сформовано функціональні та нефункціональні вимоги до майбутнього програмного забезпечення системи неперервного зважування. Визначено ключові функції: збирання, попередня обробка, нормалізація та зберігання вимірювальних даних; надання історичних записів через WebAPI з можливістю фільтрації за часовими критеріями; передавання «живих» даних у режимі реального часу через веб-сокети; підтримка інтеграції зі сторонніми системами; а також забезпечення зручного доступу до даних через клієнтський застосунок. До нефункціональних

характеристик віднесено вимоги до надійності, безпеки, масштабованості, продуктивності та розширюваності системи.

Проведений аналіз дозволив сформувати обґрунтовану архітектурну концепцію майбутнього рішення. З огляду на потребу у стандартизованому та безпечному обміні даними було прийнято рішення щодо використання OPC UA як основного і найсучаснішого промислового протоколу для взаємодії з вимірювальним обладнанням [13-15], а також розробки серверної частини на базі серверних технологій (ASP.NET, PHP, Node.jS) із застосуванням WebAPI та SignalR для забезпечення доступу як до архівних, так і до потокових даних. Такий підхід гарантує сумісність з існуючими промисловими системами, достатній рівень гнучкості для подальшого масштабування та можливість інтегрування у корпоративні інформаційні середовища.

Загалом аналітичний розділ створив цілісне підґрунтя для вибору архітектури програмного забезпечення, визначив ключові технологічні орієнтири та дозволив перейти до проєктування й реалізації системи неперервного зважування із чітко сформульованими цілями та вимогами.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ НЕПЕРЕРВНОГО ЗВАЖУВАННЯ

2.1 Вибір та обґрунтування технологій реалізації програмного забезпечення системи неперервного зважування

Проєктування програмного забезпечення для систем неперервного зважування гірничої маси потребує ретельного вибору технологічного стеку, здатного забезпечити надійну інтеграцію з промисловим обладнанням, ефективну обробку та зберігання великих обсягів даних, а також зручний та високопродуктивний доступ до вимірювальної інформації в реальному часі. Вибір технологій має ґрунтуватися не лише на їх технічних характеристиках, але й на відповідності вимогам промислових систем, серед яких провідне місце займають стабільність, масштабованість, підтримка стандартів, розширюваність і довготривала підтримка.

У межах цього проєкту було проаналізовано сучасні рішення для серверної, клієнтської та інфраструктурної частин системи, що дозволило сформувати оптимальну технологічну архітектуру, яка включає ASP.NET як основу серверної логіки, MS SQL Server як систему керування базами даних, React як клієнтську платформу, а також низку спеціалізованих бібліотек для взаємодії із промисловими протоколами, потоковим передаванням даних та роботою з базою даних.

Під час вибору серверної технології ключовими критеріями були підтримка високопродуктивних веб-API, можливість оброблення потокових даних, сумісність із технологією OPC UA, а також здатність забезпечувати стабільну роботу в умовах промислової експлуатації. Було розглянуто кілька сучасних платформ, серед яких Node.js, Python (FastAPI, Django), Java (Spring Boot) та ASP.NET на базі .NET 8.

Порівняльний аналіз показав, що платформа ASP.NET має низку переваг, важливих саме для промислових застосунків [13]. Завдяки високопродуктивному виконанню на основі JIT-компіляції, ASP.NET забезпечує мінімальні затримки у роботі серверної частини, що особливо важливо для систем неперервного моніторингу. Крім того, вбудована підтримка асинхронності дозволяє оптимально реалізувати обробку великої кількості одночасних клієнтських запитів, уникаючи блокування потоків і зберігаючи високу пропускну здатність. Суттєвим аргументом є також наявність нативних механізмів масштабування в рамках екосистеми .NET, що дозволяє розгортати застосунок у хмарних середовищах або корпоративній інфраструктурі без суттєвих модифікацій коду.

Окрему увагу слід приділити підтримці технології SignalR [16], інтегрованої в ASP.NET, яка забезпечує ефективне реалізування передавання даних у режимі реального часу, що є критично важливим для відображення «живих» вагових показників на клієнтській стороні. Зручні інструменти для розробки WebAPI, розвинена система типізації та високий рівень безпеки також вплинули на остаточний вибір на користь ASP.NET як основної серверної платформи.

У проєкті було також розглянуто широкий спектр систем керування базами даних, серед яких PostgreSQL, MySQL/MariaDB, SQLite, а також MS SQL Server. Вибір MS SQL Server ґрунтувався на необхідності забезпечення високої надійності, ефективних механізмів транзакційної обробки та можливості працювати з великими обсягами даних, характерних для систем неперервного моніторингу.

MS SQL Server забезпечує стійку продуктивність у випадках інтенсивного запису, що є типовою операцією для системи неперервного зважування, де кожні декілька секунд надходять нові вимірювання. Важливим фактором є розвинена підтримка індексування та оптимізатор запитів, який здатний швидко обробляти запити на вибірку історичних даних з фільтрацією за часовими параметрами.

Ще одним вагомим аргументом є органічна інтеграція MS SQL Server із технологіями .NET та Entity Framework Core. Саме завдяки цьому програмне забезпечення отримує можливість ефективно взаємодіяти з базою даних через ORM-рівень, що підвищує надійність, прискорює розробку та зменшує кількість можливих помилок, пов’язаних з ручним створенням SQL-запитів. Також слід врахувати підтримку потужних засобів резервного копіювання, відновлення та моніторингу, що є важливими складовими експлуатації промислових інформаційних систем.

Також ваговим фактором вибору MS SQL Server стало те, що дана СУБД активно використовуються у діючих інформаційних системах на гірничо-збагачувальних та металургійних підприємствах регіону,

Для створення клієнтського застосунку було обрано React – сучасну JavaScript-бібліотеку для побудови інтерфейсів користувача. Аналіз доступних фреймворків (Angular, Vue.js, Svelte) показав, що React є найбільш гнучким і поширеним рішенням для систем, які потребують високої продуктивності та гнучкої архітектури компонентів.

Ключовою особливістю React є декларативний підхід до побудови інтерфейсу, що значно спрощує синхронізацію візуального відображення з поточним станом даних, особливо у випадку роботи з потоковими значеннями, що надходять із серверу через SignalR. Можливість використання хуків і контекстів забезпечує ефективну організацію логіки стану застосунку, що є важливою характеристикою для реалізації модулів перегляду історичних та реальних даних.

Окрім цього, React має широку екосистему бібліотек – від інструментів побудови графіків до засобів маршрутизації, що дозволяє швидко реалізувати інтерфейс з високим рівнем інтерактивності. Потужна підтримка з боку спільноти, велика база документованих практик та тривалий життєвий цикл технології додатково обґрунтовують її використання у проєкті.

У системі неперервного зважування вагому роль відіграють не лише базові технології, але й додаткові бібліотеки, що забезпечують інтеграцію із

промисловим обладнанням, передавання даних у реальному часі та доступ до бази даних.

Особливе значення має бібліотека OPCFoundation.NetStandard.Opc.Ua [14], яка забезпечує повну підтримку протоколу OPC UA – сучасного стандарту промислових комунікацій. Саме через цей протокол відбувається взаємодія серверної частини з вимірювальним обладнанням, а також гарантоване отримання валідованих і структурованих даних про стан системи.

Бібліотека SignalR використовується для реалізації каналу передавання даних у реальному часі між сервером та клієнтським застосунком. Завдяки нативній інтеграції з ASP.NET SignalR дозволяє мінімізувати затримки та забезпечити стабільну роботу при великій кількості одночасних підключень, що є важливим у випадках моніторингу динамічних технологічних процесів.

Для роботи з базою даних використовується Microsoft.EntityFrameworkCore.SqlServer, що забезпечує ORM-рівень взаємодії між застосунком та MS SQL Server. Це дозволяє мінімізувати кількість рутинного коду для управління даними, уникати типових помилок у SQL-запитах, а також підвищує масштабованість та переносимість застосунку за рахунок раціонального розділення бізнес-логіки і механізмів доступу до даних.

2.2 Проєктування архітектури програмного забезпечення системи неперервного зважування

Архітектура розробленого програмного забезпечення системи неперервного зважування ґрунтується на клієнт-серверному підході з чітким розділенням відповідальності між рівнями збору даних, їх зберігання, обробки та візуалізації. З одного боку, система інтегрується з промисловим обладнанням через протокол OPC UA, що забезпечує стандартизований доступ до вимірювальних змінних конвеєрних ваг та швидкості стрічки. З іншого боку, реалізовано веб-орієнтований інтерфейс користувача, який забезпечує відображення як поточних («живих»), так і історичних даних із використанням

сучасних веб-технологій.

Архітектурно система складається з кількох взаємопов’язаних підсистем: серверної частини на базі ASP.NET Core, що включає OPC UA-клієнтський сервіс, WebAPI та хаб SignalR; рівня зберігання даних на основі MS SQL Server та ORM-бібліотеки Entity Framework Core; а також клієнтської частини, реалізованої на React з використанням SignalR-клієнта та HTTP-клієнта axios. Такий поділ дозволяє забезпечити масштабованість системи, ізоляцію промислової логіки від рівня представлення та гнучкість подальшої модернізації.

2.2.1 Загальна компонентна структура системи

Загальну структуру компонентів програмного забезпечення системи моніторингу конвеєрних ваг відображено у діаграмі компонентів, описаній у синтаксисі PlantUML. На діаграмі (рис. 2.1) виділено промислове обладнання із OPC UA-сервером, серверну частину (backend), базу даних, а також клієнтський застосунок (frontend).

На концептуальному рівні OPC UA-сервер виступає джерелом технологічних даних (маса матеріалу на конвеєрі, швидкість конвеєрної стрічки), до якого підключається фоновий сервіс OPC UA-клієнта на стороні ASP.NET. Далі дані потрапляють до рівня доступу до даних, де зберігаються в базі даних MS SQL Server, а також одночасно транслюються у хаб SignalR для організації потокової взаємодії з клієнтськими застосунками. Історичні дані надаються через REST API, а клієнт на React споживає як поточні значення через SignalR, так і архівні через HTTP-запити.

2.2.2 Серверна частина та служба OPC UA

Серверна частина реалізована на базі ASP.NET Core та організована як набір взаємодіючих сервісів і контролерів, зареєстрованих у контейнері залежностей у файлі Program.cs. Скріншот вікна Solution Explorer у Visual Studio, що відображає структуру проєкту серверної частини, наведено на рис. 2.2.

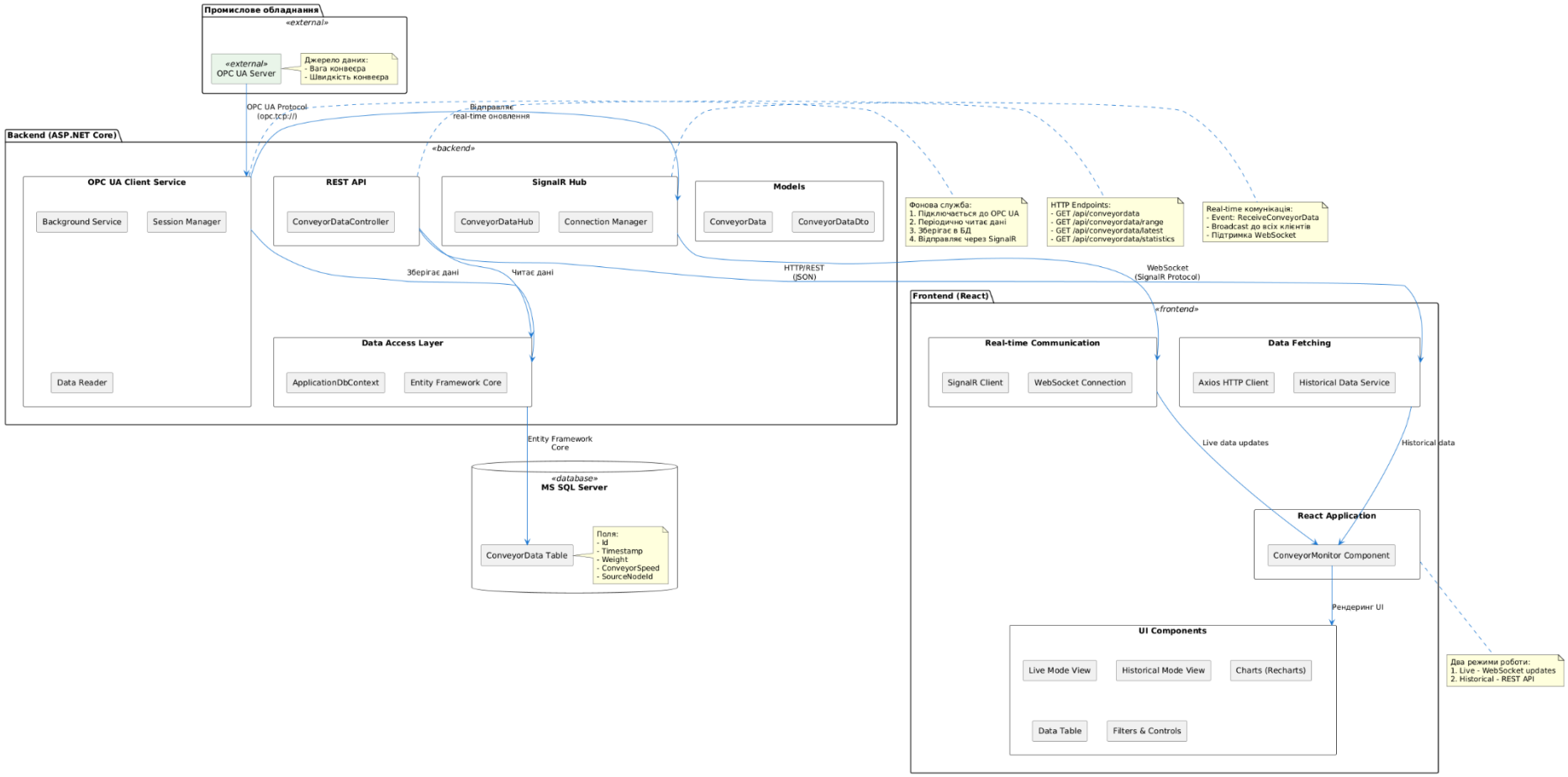

Рисунок 2.1 – Діаграма компонентів UML програмного забезпечення системи неперервного зважування

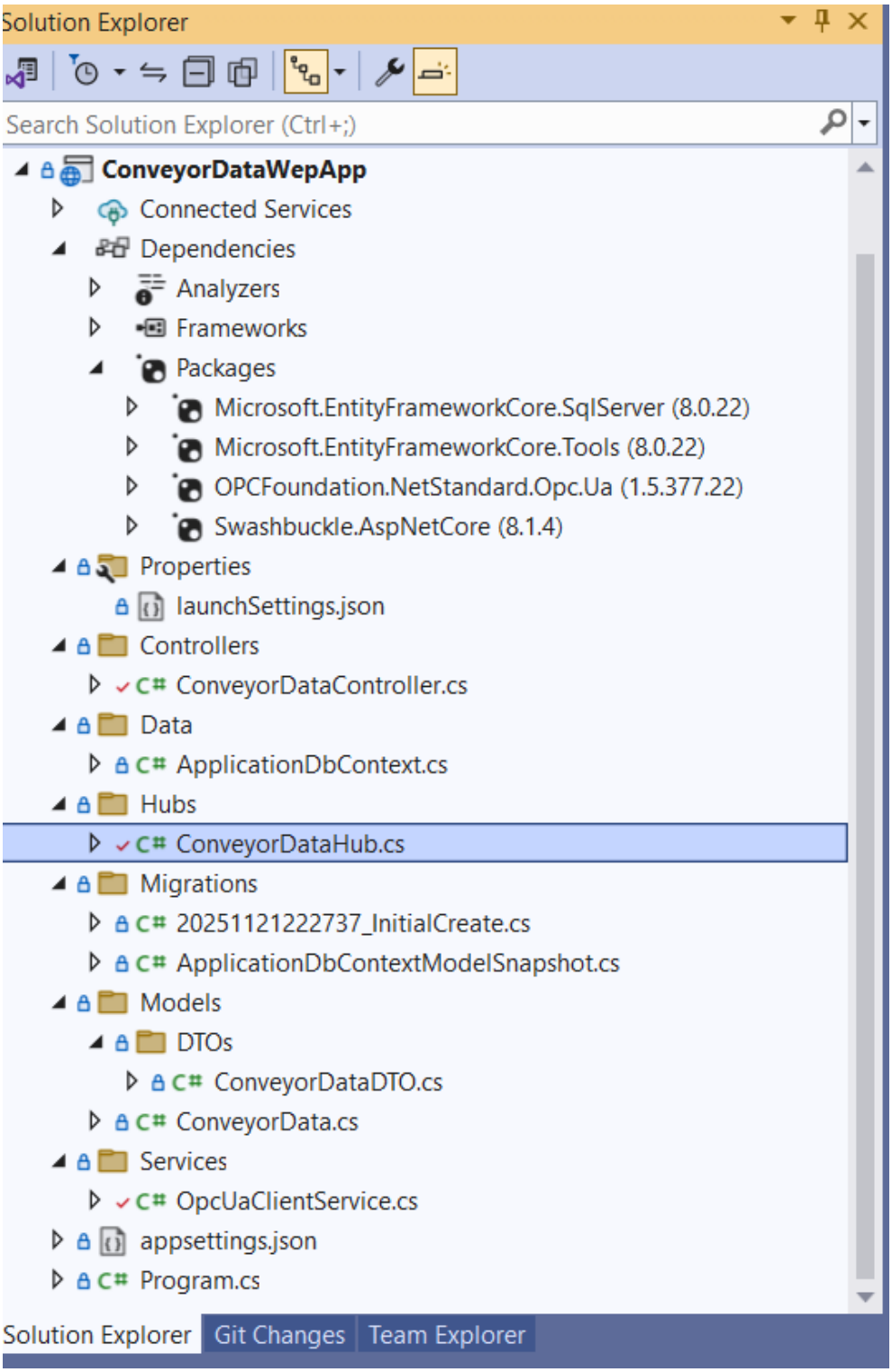

Рисунок 2.2 – Структура проєкту серверної частини додатку

Центральне місце займає фоновий сервіс OpcUaClientService, який успадковує клас BackgroundService та відповідає за організацію життєвого циклу підключення до OPC UA-сервера, періодичне зчитування даних, їх збереження в БД та трансляцію у реальному часі через SignalR.

У методі ExecuteAsync (рис. 2.3) сервісу OpcUaClientService реалізовано нескінченний цикл, у межах якого відбувається перевірка стану сесії OPC UA, спроби підключення у випадку її відсутності, а також виклик операції читання й обробки даних із заданим інтервалом сканування, що налаштовується через параметр конфігурації OpcUa:ScanIntervalMs. Такий підхід дозволяє підтримувати стійкий робочий цикл служби, враховуючи можливі розриви зв'язку з промисловим обладнанням.

```
35      protected override async Task ExecuteAsync(CancellationToken stoppingToken)
36      {
37          _logger.LogInformation("===>>> OPC UA Client Service starting...");
38          await Task.Delay(2000, stoppingToken);
39          _logger.LogInformation("===>>> Starting main loop...");
40          while (!stoppingToken.IsCancellationRequested)
41          {
42              _logger.LogInformation("===>>> Loop iteration started");
43              try
44              {
45                  if (_session == null || !_session.Connected)
46                  {
47                      _logger.LogWarning("OPC UA session not connected, attempting to connect...");
48                      await ConnectAsync(stoppingToken);
49                  }
50                  if (_session != null && _session.Connected)
51                  {
52                      _logger.LogInformation("===> Reading OPC UA data...");
53                      await ReadAndSaveDataAsync(stoppingToken);
54                  }
55                  else
56                  {
57                      _logger.LogWarning("OPC UA session still not connected after connection attempt");
58                  }
59                  _logger.LogInformation($"Waiting {_scanInterval}ms before next scan...");
60                  await Task.Delay(_scanInterval, stoppingToken);
61              }
62              catch (Exception ex)
63              {
64                  _logger.LogError(ex, "Error in OPC UA service");
65                  await Task.Delay(5000, stoppingToken);
66              }
67          }
68          _logger.LogInformation("OPC UA Client Service stopped.");
69      }
```

Рисунок 2.3 – Лістинг реалізації методу ExecuteAsync сервісу OpcUaClientService

Метод ConnectAsync (рис. 2.4) сервісу OpcUaClientService виконує ініціалізацію конфігурації застосунку OPC UA-клієнта, налаштування сертифікатів, побудову і валідацію параметрів безпеки, вибір кінцевої точки підключення CoreClientUtils.SelectEndpointAsync, створення сесії за допомогою фабрики TraceableSessionFactory.Instance та встановлення анонімної ідентичності для підключення до сервера. Таким чином, у межах одного сервісу

реалізовано повний життєвий цикл OPC UA-сесії від формування конфігурації до контролю стану підключення.

```
71      private async Task ConnectAsync(CancellationToken cancellationToken)
72      {
73          try
74          {
75              var endpointUrl = _configuration["OpcUa:EndpointUrl"];
76              _logger.LogInformation($"==1== Connecting to OPC UA server: {endpointUrl}");
77
78              var pkiRoot = Path.Combine(
79                  Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData),
80                  "OPC Foundation",
81                  "pki");
82
83              _appConfig = new ApplicationConfiguration...;
124
125             Directory.CreateDirectory(Path.Combine(pkiRoot, "own"));
126             Directory.CreateDirectory(Path.Combine(pkiRoot, "issuers"));
127             Directory.CreateDirectory(Path.Combine(pkiRoot, "trusted"));
128             Directory.CreateDirectory(Path.Combine(pkiRoot, "rejected"));
129
130             _logger.LogInformation("==2== Before Validate");
131             await _appConfig.Validate(ApplicationType.Client);
132             _logger.LogInformation("==3== After Validate");
133
134             _logger.LogInformation("==4== Before SelectEndpointAsync");
135             var selectedEndpoint = await CoreClientUtils.SelectEndpointAsync(
136                 _appConfig,
137                 endpointUrl,
138                 useSecurity: false);
139             _logger.LogInformation("==5== After SelectEndpointAsync");
140
141             var endpointConfiguration = EndpointConfiguration.Create(_appConfig);
142             var configuredEndpoint = new ConfiguredEndpoint(
143                 null,
144                 selectedEndpoint,
145                 endpointConfiguration);
```

Рисунок 2.4 – Фрагмент лістингу реалізації методу ConnectAsync сервісу OpcUaClientService

Безпосереднє читання змінних ваги та швидкості реалізовано в методі ReadAndSaveDataAsync (рис. 2.5). У цьому методі із файлу конфігурації зчитуються ідентифікатори вузлів OPC UA (WeightNodeId, SpeedNodeId), формується колекція ReadValueIdCollection, а потім викликається асинхронний метод _session.ReadAsync. Отримані значення перевіряються на коректність через коди статусу, конвертуються до типів .NET, і на їх основі формується сутність ConveyorData з часовою міткою та джерелом даних. Далі дані зберігаються в базі даних методом SaveToDatabase через створення нового DI-області (CreateScope), отримання контексту ApplicationDbContext та виконання операцій Add і SaveChangesAsync.

```
1 reference
private async Task ReadAndSaveDataAsync(CancellationToken cancellationToken)
{
    try
    {
        var weightNodeId = _configuration["OpcUa:WeightNodeId"];
        var speedNodeId = _configuration["OpcUa:SpeedNodeId"];

        var nodesToRead = new ReadValueIdCollection
        {
            new ReadValueId { NodeId = new NodeId(weightNodeId), AttributeId = Attributes.Value },
            new ReadValueId { NodeId = new NodeId(speedNodeId), AttributeId = Attributes.Value }
        };

        var response = await _session!.ReadAsync(
            null,
            0,
            TimestampsToReturn.Both,
            nodesToRead,
            cancellationToken);

        var results = response.Results;

        if (results.Count >= 2 &&
            StatusCode.IsGood(results[0].StatusCode) &&
            StatusCode.IsGood(results[1].StatusCode))
        {
            _logger.LogInformation("=== OPC UA DATA READ SUCCESS ===");

            var conveyorData = new ConveyorData
            {
                Timestamp = DateTime.UtcNow,
                Weight = Convert.ToDecimal(results[0].Value),
                ConveyorSpeed = Convert.ToDecimal(results[1].Value),
                SourceNodeId = weightNodeId
            };

            _logger.LogInformation($"Weight: {conveyorData.Weight}, Speed: {conveyorData.ConveyorSpeed}");

            await SaveToDatabase(conveyorData);
            _logger.LogInformation($"Saved to DB with ID: {conveyorData.Id}");

            var dto = new ConveyorDataDto
            {
                Id = conveyorData.Id,
                Timestamp = conveyorData.Timestamp,
                Weight = conveyorData.Weight,
                ConveyorSpeed = conveyorData.ConveyorSpeed,
                SourceNodeId = conveyorData.SourceNodeId
            };

            _logger.LogInformation($"=== SENDING TO SIGNALR === ID: {dto.Id}");

            await _hubContext.Clients.All.SendAsync("ReceiveConveyorData", dto, cancellationToken);

            _logger.LogInformation("=== SIGNALR SENT SUCCESSFULLY ===");
        }
        else
        {
            _logger.LogWarning($"OPC UA read failed. Count: {results.Count}, " +
                $"Status: {results[0].StatusCode}");
        }
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "!!! EXCEPTION IN ReadAndSaveDataAsync !!!");
    }
}
```

Рисунок 2.5 – Лістинг реалізації методу ReadAndSaveDataAsync сервісу OpcUaClientService

Після збереження формується DTO-об’єкт ConveyorDataDto, який передається у SignalR-хаб через _hubContext.Clients.All.SendAsync("ReceiveConveyorData", dto, cancellationToken). Таким чином, в одному логічному блоці виконуються всі вимоги щодо збирання, зберігання та потокової публікації даних.

2.2.3 Модель даних та рівень доступу до бази даних

Рівень доступу до даних реалізовано за допомогою бібліотеки Entity Framework Core та контексту ApplicationDbContext.

Модель доменних даних описується класом ConveyorData (рис. 2.6).

```
1   using System.ComponentModel.DataAnnotations.Schema;
2   using System.ComponentModel.DataAnnotations;
3
4   namespace ConveyorDataWepApp.Models
5   {
        11 references
6       public class ConveyorData
7       {
8           [Key]
            4 references
9           public int Id { get; set; }
10
11          [Required]
            13 references
12          public DateTime Timestamp { get; set; }
13
14          [Column(TypeName = "decimal(18,2)")]
            10 references
15          public decimal Weight { get; set; }
16
17          [Column(TypeName = "decimal(18,2)")]
            8 references
18          public decimal ConveyorSpeed { get; set; }
19
            4 references
20          public string? SourceNodeId { get; set; }
21      }
22  }
23
```

Рисунок 2.6 – Лістинг класу ConveyorData для збереження у БД результатів вимірювання даних з конвеєрних ваг

Клас ConveyorData містить наступні властивості:

– Id (Primary Key) – унікальний ідентифікатор запису;

– Timestamp – мітка часу збору даних;

– Weight – вага на конвеєрі (decimal 18,2);

– ConveyorSpeed – швидкість руху конвеєра (decimal 18,2);

– SourceNodeId – ідентифікатор вузла OPC UA, що є джерелом даних.

Використання атрибутів Column(TypeName = "decimal(18,2)") для полів ваги та швидкості дозволяє забезпечити точне зберігання вимірювальних значень у базі даних, що є важливим з погляду подальшої аналітики.

Рівень доступу до даних реалізований з використанням Object-Relational Mapping (ORM) фреймворку Entity Framework Core. Компонент ApplicationDbContext (рис. 2.7) успадковується від DbContext та визначає:

– набір сутностей – колекція DbSet<ConveyorData> для роботи з таблицею телеметричних даних;

– конфігурація моделі – створення індексу на полі Timestamp для оптимізації запитів з фільтрацією за часом;

– міграції схеми бази даних – автоматична генерація та застосування змін структури БД.

```
1   using ConveyorDataWepApp.Models;
2   using Microsoft.EntityFrameworkCore;
3
4   namespace ConveyorDataWepApp.Data
5   {
        8 references
6       public class ApplicationDbContext : DbContext
7       {
            0 references
8           public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
9               : base(options)
10          {
11          }
12
            8 references
13          public DbSet<ConveyorData> ConveyorData { get; set; }
14
            0 references
15          protected override void OnModelCreating(ModelBuilder modelBuilder)
16          {
17              base.OnModelCreating(modelBuilder);
18
19              modelBuilder.Entity<ConveyorData>(entity =>
20              {
21                  entity.HasIndex(e => e.Timestamp);
22              });
23          }
24      }
25  }
```

Рисунок 2.7 – Лістинг класу контексту бази даних ApplicationDbContext

Особливістю даного класу є те, що у методі OnModelCreating задається індекс по полю Timestamp, що оптимізує вибірку даних за часовими критеріями. Це безпосередньо пов’язано з функціональними вимогами до фільтрації історичних даних та побудови статистичних запитів за період часу.

Реєстрація контексту відбувається в Program.cs з використанням SQL Server як СУБД: builder.Services.AddDbContext<ApplicationDbContext>(…);. Такий підхід забезпечує гнучкість налаштування під час розгортання системи, оскільки фактичний рядок підключення зберігається у конфігураційних файлах.

2.2.4 Реалізація хабу SignalR

SignalR Hub забезпечує двосторонню комунікацію в реальному часі між сервером та клієнтськими застосунками [16]. Компонент ConveyorDataHub (рис. 2.8) наслідує базовий клас Hub з бібліотеки Microsoft.AspNetCore.SignalR та реалізує наступний функціонал:

– широкомовна розсилка даних – метод SendConveyorData забезпечує відправку телеметричних даних всім підключеним клієнтам через метод Clients.All.SendAsync;

– управління підключеннями – відстеження підключення та від'єднання клієнтів з логуванням ідентифікаторів сесій;

– підтримка множинних транспортних протоколів – автоматичний вибір оптимального транспорту (WebSocket, Server-Sent Events або Long Polling).

Хаб ConveyorDataHub також містить перевизначені методи OnConnectedAsync та OnDisconnectedAsync, що використовуються для журналювання підключень клієнтів. Основний потік «живих» даних надходить сюди із OpcUaClientService, після чого розсилається до всіх підключених клієнтів методом Clients.All.SendAsync("ReceiveConveyorData", dto).

SignalR використовує подієву модель взаємодії, де сервер ініціює відправку даних клієнтам за допомогою іменованих подій (в даному випадку "ReceiveConveyorData").

```
6   public class ConveyorDataHub : Hub
7   {
8       private readonly ILogger<ConveyorDataHub> _logger;
            0 references
9       public ConveyorDataHub(ILogger<ConveyorDataHub> logger)
10      {
11          _logger = logger;
12      }
            0 references
13      public async Task SendConveyorData(ConveyorData data)
14      {
15          _logger.LogInformation($"Hub: SendConveyorData called with Weight={data.Weight}");
16          await Clients.All.SendAsync("ReceiveConveyorData", data);
17      }
            0 references
18      public override async Task OnConnectedAsync()
19      {
20          await base.OnConnectedAsync();
21          _logger.LogInformation($"*** CLIENT CONNECTED *** ID: {Context.ConnectionId}");
22      }
            0 references
23      public override async Task OnDisconnectedAsync(Exception? exception)
24      {
25          await base.OnDisconnectedAsync(exception);
26          _logger.LogWarning($"*** CLIENT DISCONNECTED *** ID: {Context.ConnectionId}, " +
27              $"Exception: {exception?.Message}");
28      }
29  }
```

Рисунок 2.8 – Лістинг класу ConveyorDataHub, що реалізує функціонал хабу SignalR

2.2.5 Контролер REST API

REST API реалізований за допомогою класу ConveyorDataController, який надає кінцеві точки (HTTP-endpoints) для доступу до історичних даних та управління системою.

Реалізація REST API зосереджена в контролері ConveyorDataController. У ньому реалізовано кілька ендпоїнтів, які відповідають функціональним вимогам до надання доступу до історичних даних. Метод GetAll (рис. 2.8) забезпечує пагінований доступ до всієї сукупності вимірювань з можливістю вказати номер сторінки та розмір вибірки.

Для підтримки роботи з пагінацією додано Generic-клас PagedResult<T> (рис. 2.10), який спрощує вибірку даних за номером сторінки. У методі GetAll для реалізації фільтрації даних використовуються LINQ-методи OrderByDescending(), Take() та Skip().

```
58          [HttpGet]
            0 references
59          public async Task<ActionResult<PagedResult<ConveyorData>>> GetAll(
60              [FromQuery] int page = 1,
61              [FromQuery] int pageSize = 100)
62          {
63              var totalCount = await _context.ConveyorData.CountAsync();
64
65              var data = await _context.ConveyorData
66                  .OrderByDescending(d => d.Timestamp)
67                  .Skip((page - 1) * pageSize)
68                  .Take(pageSize)
69                  .ToListAsync();
70
71              return Ok(new PagedResult<ConveyorData>
72              {
73                  Data = data,
74                  TotalCount = totalCount,
75                  Page = page,
76                  PageSize = pageSize
77              });
78          }
```

Рисунок 2.9 – Лістинг методу GetAll контролеру ConveyorDataController для отримання усіх записів вимірювань

```
161     public class PagedResult<T>
162     {
            1 reference
163         public List<T> Data { get; set; } = new();
            2 references
164         public int TotalCount { get; set; }
            1 reference
165         public int Page { get; set; }
            2 references
166         public int PageSize { get; set; }
            0 references
167         public int TotalPages => (int)Math.Ceiling(TotalCount / (double)PageSize);
168     }
```

Рисунок 2.10 – Лістинг класу PagedResult<T> для підтримки пагінації результатів HTTP-запитів

Метод GetByDateRange реалізує вибірку даних за часовим діапазоном, що відповідає вимогам до фільтрації історії за періодом (рис. 2.11). Метод GetLatest дозволяє отримати останні N записів, що використовується для оперативного аналізу динаміки без завантаження повного архіву.

```
83          [HttpGet("range")]
            0 references
84          public async Task<ActionResult<IEnumerable<ConveyorData>>> GetByDateRange(
85              [FromQuery] DateTime startDate,
86              [FromQuery] DateTime endDate)
87          {
88              var data = await _context.ConveyorData
89                  .Where(d => d.Timestamp >= startDate && d.Timestamp <= endDate)
90                  .OrderByDescending(d => d.Timestamp)
91                  .ToListAsync();
92
93              return Ok(data);
94          }
```

Рисунок 2.11 – Лістинг методу GetByDateRange контролеру ConveyorDataController для отримання фільтрованих за датою записів вимірювань

Окремий метод GetStatistics (рис. 2.12) формує агреговані статистичні показники за період, такі як середні, мінімальні, максимальні значення ваги та швидкості, а також сумарна вага. Це розширює можливості системи у напрямі аналітики.

```
113         [HttpGet("statistics")]
            0 references
114         public async Task<ActionResult<ConveyorStatistics>> GetStatistics(
115             [FromQuery] DateTime startDate,
116             [FromQuery] DateTime endDate)
117         {
118             var data = await _context.ConveyorData
119                 .Where(d => d.Timestamp >= startDate && d.Timestamp <= endDate)
120                 .ToListAsync();
121
122             if (!data.Any())
123                 return NotFound("No data found for the specified period");
124
125             var statistics = new ConveyorStatistics
126             {
127                 StartDate = startDate,
128                 EndDate = endDate,
129                 TotalRecords = data.Count,
130                 AverageWeight = data.Average(d => d.Weight),
131                 MinWeight = data.Min(d => d.Weight),
132                 MaxWeight = data.Max(d => d.Weight),
133                 AverageSpeed = data.Average(d => d.ConveyorSpeed),
134                 MinSpeed = data.Min(d => d.ConveyorSpeed),
135                 MaxSpeed = data.Max(d => d.ConveyorSpeed),
136                 TotalWeight = data.Sum(d => d.Weight)
137             };
138
139             return Ok(statistics);
140         }
```

Рисунок 2.12 – Лістинг методу GetStatistics контролеру ConveyorDataController для отримання узагальнених статистичних показників

Метод CleanupOldData (рис .2.13) дозволяє видаляти застарілі записи, що є засобом керування обсягом бази даних.

```
145         [HttpDelete("cleanup")]
            0 references
146         public async Task<ActionResult> CleanupOldData([FromQuery] int daysToKeep = 30)
147         {
148             var cutoffDate = DateTime.UtcNow.AddDays(-daysToKeep);
149
150             var oldRecords = await _context.ConveyorData
151                 .Where(d => d.Timestamp < cutoffDate)
152                 .ToListAsync();
153
154             _context.ConveyorData.RemoveRange(oldRecords);
155             await _context.SaveChangesAsync();
156
157             return Ok(new { DeletedRecords = oldRecords.Count, CutoffDate = cutoffDate });
158         }
159     }
160
```

Рисунок 2.13 – Лістинг методу CleanupOldData контролеру ConveyorDataController для видалення застарілих даних

Таким чином, контролер ConveyorDataController експонує наступні кінцеві точки:

1. Отримання даних:

– GET /api/conveyordata – постраничне отримання всіх записів з підтримкою пагінації;

– GET /api/conveyordata/range – вибірка даних за заданий часовий період;

– GET /api/conveyordata/latest – отримання останніх N записів;

2. Аналітика:

– GET /api/conveyordata/statistics – обчислення статистичних показників (середнє, мінімум, максимум, сума) за період;

3. Обслуговування:

– DELETE /api/conveyordata/cleanup – видалення застарілих даних для управління розміром бази даних;

– POST /api/conveyordata/test-signalr – тестовий endpoint для верифікації функціонування SignalR.

Контролер ConveyorDataController використовує асинхронні методи для неблокуючої обробки запитів та повертає результати у форматі JSON відповідно

до принципів RESTful API.

У файлі Program.cs здійснюється реєстрація всіх необхідних компонентів: додавання контролерів, реєстрація хабу за маршрутом /conveyorHub, налаштування CORS-політики для доступу з фронтенду, реєстрація OpcUaClientService як фонової служби, активація Swagger для документування API. Така конфігурація забезпечує зв’язність усіх компонентів серверної частини.

Для візуалізації функціонування застосунку була розроблена діаграма послідовності UML, що відображає часову взаємодію основних компонентів системи під час її роботи та демонструє повний цикл обробки даних – від моменту відкриття користувачем клієнтського застосунку до отримання ним як поточних, так і архівних вимірювань (рис. 2.14). Це дозволяє сформувати цілісне уявлення про динаміку обміну даними між фронтендом, серверною частиною, базою даних і промисловим обладнанням [17].

Після відкриття оператором клієнтського застосунку, реалізованого на React, ініціюється процес встановлення з'єднання між фронтендом і сервером SignalR. Клиєнтська частина ініціює WebSocket-підключення, після чого SignalR надсилає клієнту підтвердження з унікальним ідентифікатором підключення. У цей момент встановлюється двосторонній канал зв’язку, який забезпечує можливість передачі даних у режимі реального часу без необхідності періодичного опитування сервера [18-20]. Таким чином формується основа для live-режиму роботи, який є критично важливим для системи моніторингу технологічних параметрів.

Паралельно з роботою клієнтської частини у серверному середовищі діє фоновий сервіс OPC UA-клієнта [18]. У межах циклу періодичної роботи він з певним інтервалом ініціює зчитування даних із OPC UA-сервера промислового обладнання. Зазвичай це значення ваги матеріалу на конвеєрі та швидкості руху стрічки, представлені у вигляді вузлів інформаційної моделі OPC UA.

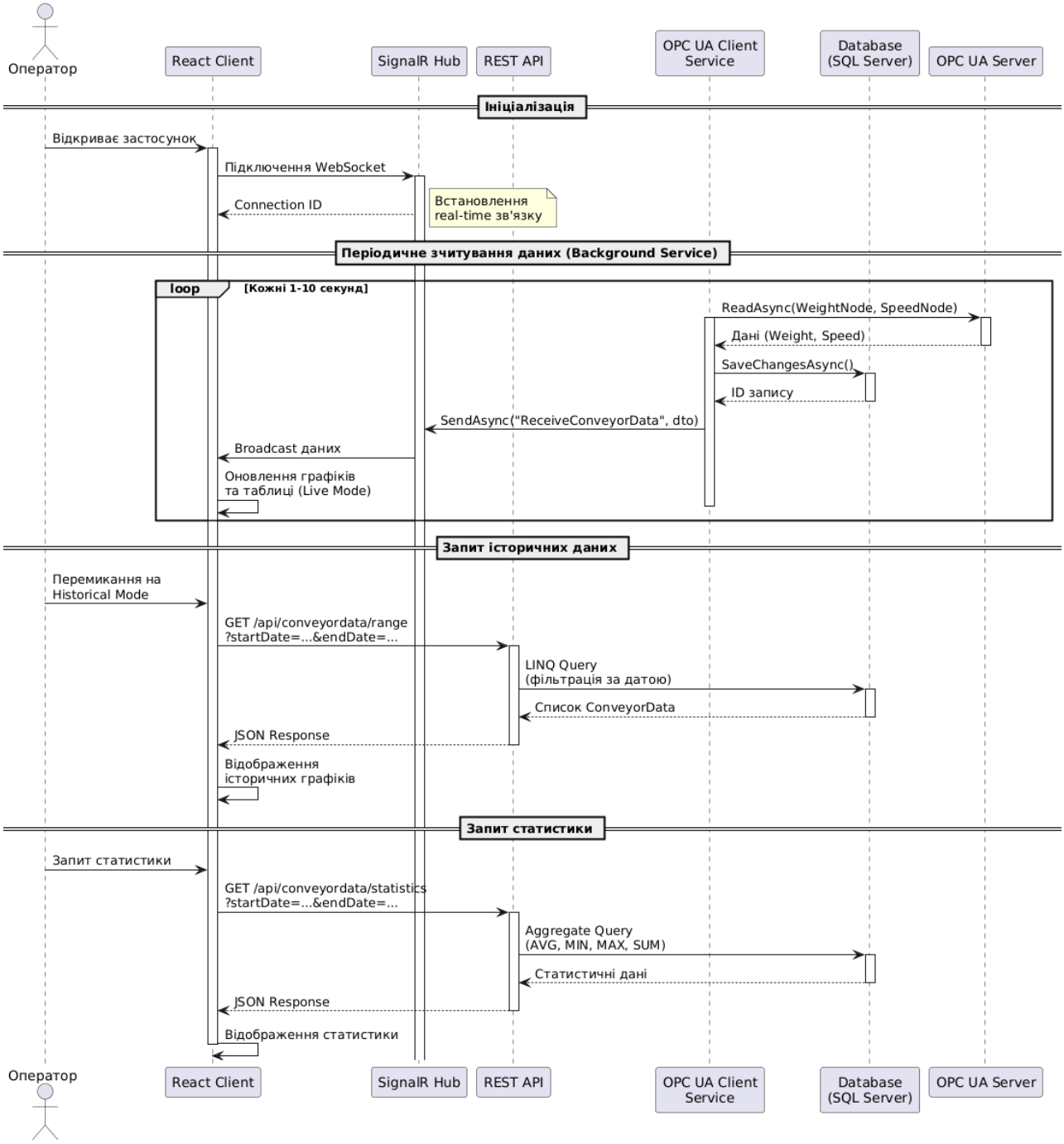

Рисунок 2.14 – Діаграма послідовності UML програмного забезпечення системи неперервного зважування

Після отримання даних сервіс виконує їх обробку та передає до реляційної бази даних для зберігання. У цьому місці відбувається формування нових записів таблиці ConveyorData із відповідними часовими мітками, числовими значеннями та метаданими.

Після успішного збереження даних фоновий сервіс формує DTO-об’єкт і ініціює трансляцію повідомлення через SignalR-хаб [16, 19]. Усі підключені клієнти отримують широкомовне повідомлення «ReceiveConveyorData», що містить найсвіжіші вимірювання. Клієнтський застосунок, отримавши нову порцію даних, оновлює інтерфейс: актуалізує покази у віджетах поточних значень, додає нову точку до набору даних live-графіка та оновлює табличне представлення недавніх вимірювань. Таким чином реалізується безперервний моніторинг конвеєрних ваг із мінімальною затримкою, що робить систему придатною для оперативного реагування на відхилення технологічних параметрів.

Після перемикання оператора в історичний режим клієнтський застосунок припиняє відображення поточних даних і формує HTTP-запит до серверного REST API. Виконується звернення до ендпоїнта /api/conveyordata/range із параметрами початкової та кінцевої дати. На сервері контролер API викликає відповідний метод доступу до даних, який за допомогою LINQ-запиту виконує фільтрацію записів у базі даних за часом. Результатом є колекція об’єктів ConveyorData, що задовольняють умови запиту. Після формування відповіді у форматі JSON дані передаються на клієнт, де виконуються їх узгодження, форматування та візуалізація у вигляді графіків та таблиць.

У випадку формування запиту статистичних даних взаємодія відбувається аналогічно, з тією різницею, що на сервері виконується агрегація даних за допомогою операцій середнього, мінімального, максимального значень, а також обчислення сумарної маси. Після цього агреговані статистичні показники повертаються клієнту у JSON-форматі й відображаються у відповідному інтерфейсному компоненті.

Таким чином діаграма послідовності наочно демонструє, що система функціонує як узгоджений комплекс компонентів, де фоновий OPC UA-сервіс забезпечує постійне отримання промислових даних, серверна частина відповідає за їх збереження та упорядкування, а клієнтський застосунок — за відображення як поточних, так і історичних вимірювань. Інтеграція SignalR створює

ефективний механізм роботи в реальному часі, а REST API дає змогу здійснювати аналітичну та історичну вибірку даних [20]. Узгодженість компонентів у часовому вимірі гарантує коректну роботу системи навіть за умов нестабільних промислових мереж і масштабування навантаження.

2.2.6 Механізми забезпечення надійності програмного забезпечення

Під час проєктування додатку та реалізації програмного коду передбачено декілька механізмів забезпечення надійності, зокрема:

– механізми відновлення з'єднань;

– обробка помилок;

– індексація бази даних.

Програмне забезпечення системи неперервного зважування реалізує механізми відновлення як OPC UA з'єднань, так і з'єднань SignalR.

Сервіс постійно моніторить стан сесії з OPC UA сервером. При виявленні розриву з'єднання (session.Connected == false) ініціюється процес повторного підключення. Між спробами підключення система очікує 5 секунд, що запобігає перевантаженню мережі.

При реалізації зв'язку з використанням SignalR клієнт налаштований на автоматичне відновлення з'єднання з експоненційною затримкою (0, 2, 5, 10 секунд) [21]. Стан з'єднання відображається в UI, інформуючи користувача про проблеми комунікації.

На кожному рівні системи реалізована комплексна обробка виключень:

– серверна частина: всі критичні операції (читання OPC UA, запис до БД, відправка через SignalR) обгорнуті в try-catch блоки з детальним логуванням;

– клієнтська частина: HTTP запити та SignalR комунікація мають обробники помилок з логуванням до консолі браузера;

– база даних: Entity Framework автоматично виконує відкат (rollback) транзакцій при виникненні помилок збереження.

Ще одним механізмом підвищення надійності та продуктивності додатку є створення індексу на полі Timestamp. Дане рішення забезпечує швидкий пошук записів за часовим діапазоном, що критично важливо для запитів історичних

даних. Без індексу складність пошуку була б O(n), з індексом – O(log n).

2.2.7 Використані паттерни проєктування та принципи архітектури

При розробці системи застосовані наступні паттерни та принципи проєктування архітектури:

1. Dependency Injection – використання вбудованого DI-контейнера ASP.NET Core забезпечує [18]:

– слабку зв’язаність компонентів;

– спрощення тестування через можливість підміни залежностей на mock-об'єкти;

– централізоване управління життєвим циклом об'єктів (Singleton, Scoped, Transient).

2. Патерн Репозиторій (Repository Pattern) – опосередковано через DbContext; ApplicationDbContext виступає як репозиторій, інкапсулюючи логіку доступу до даних та надаючи абстракцію над фізичним сховищем [19].

3. Об’єкти передачі даних (Data Transfer Object, DTO) – ConveyorDataDto використовується для передачі даних між рівнями застосунку, що дозволяє [18]:

– відокремити внутрішню модель предметної області від API-контрактів;

– контролювати обсяг даних, що передаються по мережі;

– забезпечити версіонування API без зміни доменних моделей.

4. Background Service Pattern – OpcUaClientService реалізований як IHostedService, що працює в фоновому режимі протягом усього життєвого циклу застосунку [21]. Це забезпечує:

– безперервний збір даних незалежно від активності користувачів;

– елегантне завершення роботи (Graceful shutdown) при зупинці застосунку;

– інтеграцію з системою логування та конфігурації ASP.NET Core.

5. Принцип єдиної відповідальності (Single Responsibility Principle, SRP) – ккожен компонент системи має чітко визначену відповідальність [19]:

– OpcUaClientService – виключно взаємодія з OPC UA;

– SignalRHub – тільки real-time комунікація;

– ConveyorDataController – надання REST API;

– ApplicationDbContext – доступ до бази даних.

Це полегшує розуміння, тестування та модифікацію коду.

6. Розділення проблем (Separation of Concerns) – система чітко поділена на рівні [18]:

– презентаційний рівень (Presentation Layer) – відображення даних (рівень фронтенду на React);

– рівень API (API Layer) – надання інтерфейсів взаємодії (Controllers, Hubs);

– рівень бізнес-логіки (Business Logic Layer) – обробка бізнес-логіки (сервіси, зокрема OpcUaClientService);

– рівень доступу до даних (Data Access Layer) – робота з персистентним сховищем (DbContext).

Підводячи підсумок, запропонована архітектура в повному обсязі відповідає сформульованим раніше функціональним та нефункціональним вимогам. Збирання даних із промислового рівня забезпечується через OPC UA-клієнтський сервіс, що працює у фоновому режимі та враховує особливості промислових мереж. Надійне зберігання даних реалізовано за допомогою MS SQL Server та ORM-рівня Entity Framework Core, що гарантує транзакційну цілісність та оптимізовану вибірку за часовими параметрами. Доступ до історичних даних через WebAPI та їх фільтрація за періодами реалізовані в контролері ConveyorDataController, що дозволяє іншим підсистемам або клієнтам виконувати гнучкі запити до архіву.

Відображення «живих» даних у режимі реального часу забезпечується інтеграцією SignalR на стороні сервера та клієнта, що дозволяє оперативно інформувати оператора про зміну технологічних параметрів без додаткових опитувань сервера. Реалізація на React дає змогу організувати сучасний та інтуїтивний інтерфейс користувача, який підтримує динамічні графіки, таблиці та засоби фільтрації.

Таким чином, розроблена архітектура є логічно цілісною, технологічно

обґрунтованою та орієнтованою на реальні умови експлуатації системи неперервного зважування на промислових конвеєрних лініях. Вона створює надійну основу для подальшої реалізації, тестування та розвитку функціональних можливостей системи.

2.3 Розробка клієнтської частини програмного забезпечення системи неперервного зважування

Для реалізації клієнтського застосунку обрано технологію React. Клієнтська частина реалізована як Single Page Application (SPA) на базі бібліотеки React 18 [22, 23]. Додаток організований за компонентним підходом з використанням функціональних компонентів та React Hooks для управління станом.

Архітектура клієнтського застосунку поєднує два основні механізми отримання даних від серверної частини:

– режим отримання даних реального часу через SignalR;

– режим отримання історичних даних через REST API.

Такий підхід забезпечує реалізацію як оперативного відображення актуальних параметрів технологічного процесу, так і гнучкої аналітичної обробки архівної інформації за довільними часовими періодами. Компонент ConveyorMonitor виступає центральним елементом клієнтського застосунку, що відповідає за ініціалізацію з’єднання, керування режимами роботи, взаємодію з API, а також за візуалізацію графіків, таблиць і поточних значень.

Головний компонент ConveyorMonitor інкапсулює всю логіку взаємодії з з SignalR-хабом та REST API та управління станом інтерфейсу, візуалізацію поточних і історичних даних. Компонент використовує наступні React Hooks:

– useState – для управління локальним станом (дані, підключення, режим відображення, фільтри);

– useEffect – для виконання побічних ефектів (встановлення SignalR з'єднання, cleanup);

– useCallback – для мемоізації функцій завантаження історичних даних.

На початку формуються стани для роботи із живими та історичними даними, параметрами фільтрації і з'єднанням SignalR (рис. 2.15).

Завдяки цьому компонент може гнучко реагувати на зміни вхідних даних, перемикатися між режимами, зберігати історію вимірювань і контролювати кількість точок, що відображаються на графіку.

```
24 const ConveyorMonitor = () => {
25     const [connection, setConnection] = useState(null);
26     const [liveData, setLiveData] = useState([]);
27     const [historicalData, setHistoricalData] = useState([]);
28     const [isConnected, setIsConnected] = useState(false);
29     const [isLiveMode, setIsLiveMode] = useState(true);
30     const [latestValue, setLatestValue] = useState(null);
31
32     // Фільтри
33     const [startDate, setStartDate] = useState('');
34     const [endDate, setEndDate] = useState('');
35     const [recordCount, setRecordCount] = useState(100);
36     const [maxLivePoints, setMaxLivePoints] = useState(50);
```

Рисунок 2.15 – Лістинг структури станів компонента ConveyorMonitor

Модуль real-time комунікації забезпечує постійне з'єднання з сервером для отримання даних телеметрії в режимі реального часу. Реалізація базується на бібліотеці @microsoft/signalr з наступною конфігурацією [24]:

1. Транспортні протоколи:

– пріоритетний транспорт – WebSocket для мінімальної затримки;

– резервні транспорти – Server-Sent Events та Long Polling для забезпечення сумісності.

2. Стратегія відновлення з'єднання:

– автоматичне перепідключення з експоненційною затримкою (0, 2000, 5000, 10000 мс), , що підвищує стійкість клієнта до розривів зв'язку.;

– обробка подій reconnecting, reconnected та close для індикації стану з'єднання користувачу;

3. Обробка вхідних даних:

– підписка на подію "ReceiveConveyorData" для отримання нових вимірювань;

– оновлення локального стану з обмеженням кількості точок на графіку для оптимізації продуктивності рендерингу;

– форматування міток часу для відображення на осі X графіків.

Ключовою особливістю системи є підтримка live-режиму, який реалізується через підключення до SignalR-хабу серверної частини. Ініціалізація WebSocket-з'єднання відбувається один раз під час монтування компонента ConveyorMonitor (рис. 2.16).

```
38  // Підключення до SignalR
39  useEffect(() => {
40    const newConnection = new signalR.HubConnectionBuilder()
41      .withUrl(SIGNALR_HUB_URL, {
42        transport: signalR.HttpTransportType.WebSockets |
43                   signalR.HttpTransportType.ServerSentEvents |
44                   signalR.HttpTransportType.LongPolling
45      })
46      .withAutomaticReconnect([0, 2000, 5000, 10000])
47      .configureLogging(signalR.LogLevel.Debug)
48      .build();
49    newConnection.on('ReceiveConveyorData', (data) => {
50      setLatestValue(data);
51      setLiveData(prev => {
52        const updated = [...prev, { ...data, time:
53          format(new Date(data.timestamp), 'HH:mm:ss') }];
54        return updated.length > maxLivePoints ?
55        updated.slice(-maxLivePoints) : updated;
56      });
57    });
58
59    newConnection.start().then(() => setIsConnected(true));
60
61    setConnection(newConnection);
62  }, []);
```

Рисунок 2.16 – Лістинг ініціалізації SignalR з автоматичним перепідключенням

Після встановлення з'єднання (рис. 2.17) компонент отримує потоки даних у реальному часі. Дані обробляються локально, форматуються, додаються до

масиву liveData, а останнє значення зберігається у latestValue для оперативного відображення в інтерфейсі. Залишення обмеженого числа точок у масиві запобігає надмірному навантаженню на фронтенд.

```
84      // Запуск підключення
85      const startConnection = async () => {
86        try {
87          await newConnection.start();
88          console.log('SignalR Connected successfully!');
89          console.log('Connection State:', newConnection.state);
90          setIsConnected(true);
91          setConnection(newConnection);
92        } catch (err) {
93          console.error('SignalR Connection Error:', err);
94          setIsConnected(false);
95          // Спроба повторного підключення через 5 секунд
96          setTimeout(() => startConnection(), 5000);
97        }
98      };
99
100     startConnection();
101
102     // Cleanup при розмонтуванні компонента
103     return () => {
104       if (newConnection && newConnection.state ===
105         signalR.HubConnectionState.Connected) {
106         newConnection.stop().then(() => {
107           console.log('SignalR Connection stopped');
108         }).catch(err => {
109           console.error('Error stopping connection:', err);
110         });
111       }
112     };
113   }, []); // Порожній масив залежностей - виконується лише один раз
114
```

Рисунок 2.17 – Лістинг запуску підключення під час монтування компонента ConveyorMonitor

Для завантаження історичних даних використовується бібліотека axios – HTTP-клієнт на основі Promise API. Модуль DataFetch виконує наступні функції:

– параметризовані запити – формування URL з query-параметрами для фільтрації даних за часовим діапазоном або кількістю записів;

– обробка відповідей – десеріалізація JSON та трансформація даних для відображення (форматування дат, округлення числових значень);

– обробка помилок – логування помилок HTTP-запитів та інформування користувача про проблеми з'єднання.

HTTP-клієнт axios звертається до ендпоїнтів conveyordata/latest та conveyordata/range через базову адресу API. Дані, отримані від сервера, форматуються (зокрема час – за допомогою бібліотеки date-fns) та зберігаються у стані historicalData.

Запити виконуються асинхронно з використанням async/await синтаксису, що забезпечує неблокуючу роботу інтерфейсу користувача під час завантаження даних [24].

У режимі історичного перегляду компонент звертається до репрезентативних кінцевих точок серверної частини. Це дає змогу завантажувати архівні значення за період або останні записані вимірювання.

```
114  // Завантаження історичних даних
115  const loadHistoricalData = useCallback(async () => {
116    try {
117      let url = `${API_BASE_URL}/conveyordata/latest?count=${recordCount}`;
118
119      if (startDate && endDate) {
120        url = `${API_BASE_URL}/conveyordata/range?startDate=${startDate}&endDate=${endDate}`;
121      }
122
123      const response = await axios.get(url);
124      const formattedData = response.data.map((item) => ({
125        ...item,
126        time: format(new Date(item.timestamp), 'dd/MM HH:mm:ss')
127      }));
128
129      setHistoricalData(formattedData);
130    } catch (error) {
131      console.error('Error loading historical data:', error);
132    }
133  }, [startDate, endDate, recordCount]);
```

Рисунок 2.18 – Лістинг коду отримання історичних даних з використанням бібліотеки axios

Такий механізм забезпечує точну відповідність функціональним вимогам до перегляду архівних даних, їх фільтрації та обмеження кількості записів.

Рівень представлення складається з набору спеціалізованих UI-компонентів, які умовно можна поділити на групу компонентів відображення метрик, групу компонентів графічної візуалізації, групу компонентів управління та компонент для табличного представлення.

До групи компонентів відображення метрик входять:

– Value Cards – карточки з поточними значеннями ваги, швидкості та часу вимірювання;

– індикатор стану з’єднання – візуальна індикація активності SignalR підключення.

До групи компонентів графічної входять інтерактивні графіки:

– графік зміни ваги в часі;

– графік зміни швидкості конвеєра в часі.

Для побудови інтерактивних графіків використовується бібліотека Recharts, яка надає декларативні React-компоненти для створення діаграм.

Графіки підтримують автоматичне масштабування осей, відображення підказок при наведенні курсора та легенди для ідентифікації метрик.

До групи компонентів управління входять:

– перемикач режимів (Live/Historical) – кнопка для зміни джерела даних;

– панель фільтрів – елементи введення для вибору часового діапазону та кількості записів;

– налаштування відображення – елементи керування для задання максимальної кількості точок на графіках.

Компонент ConveyorMonitor підтримує два логічні режими роботи:

– Live Mode – використання SignalR і потокове оновлення даних;

– Historical Mode – завантаження архіву через REST API.

Перемикання між ними реалізовано без перезавантаження сторінки та без змінення стану з’єднання SignalR (рис. 2.19).

```
135   // Перемикання режимів
136   const toggleMode = () => {
137     const newMode = !isLiveMode;
138     setIsLiveMode(newMode);
139
140     if (!newMode) {
141       // Перехід в історичний режим
142       loadHistoricalData();
143     } else {
144       // Перехід в live режим - очистити історичні дані
145       setHistoricalData([]);
146     }
147   };
```

Рисунок 2.19 – Лістинг обробки перемикання режимів роботи інтерфейсу

Для представлення даних використано бібліотеку Recharts, що дозволяє будувати інтерактивні лінійні графіки з підтримкою автоматичного ресайзу, тултипів та легенд. Графіки побудовані окремо для ваги та швидкості конвеєра, що полегшує сприйняття динаміки зміни параметрів (рис. 2.20).

```
239       {/* Графіки */}
240       <div className="charts">
241         <div className="chart-container">
242           <h3>Вага (кг)</h3>
243           <ResponsiveContainer width="100%" height={300}>
244             <LineChart data={currentData}>
245               <CartesianGrid strokeDasharray="3 3" />
246               <XAxis dataKey="time" />
247               <YAxis />
248               <Tooltip />
249               <Legend />
250               <Line
251                 type="monotone"
252                 dataKey="weight"
253                 stroke="#8884d8"
254                 strokeWidth={2}
255                 dot={false}
256                 name="Вага"
257               />
258             </LineChart>
259           </ResponsiveContainer>
260         </div>
261
262         <div className="chart-container">
263           <h3>Швидкість конвеєра (м/с)</h3>
264           <ResponsiveContainer width="100%" height={300}>
```

Рисунок 2.20 – Лістинг реалізації візуалізації графіків

Дані також відображаються у табличній формі (рис. 2.21), де останні 20 записів подаються у порядку, зручному для оперативного аналізу, з відсортуванням за часом у зворотному порядку (найновіші зверху).

```
284         {/* Таблиця даних */}
285         {currentData.length > 0 && (
286           <div className="data-table">
287             <h3>Дані ({currentData.length} записів)</h3>
288             <table>
289               <thead>
290                 <tr>
291                   <th>Час</th>
292                   <th>Вага (кг)</th>
293                   <th>Швидкість (м/с)</th>
294                 </tr>
295               </thead>
296               <tbody>
297                 {currentData.slice(-20).reverse().map((item, index) => (
298                   <tr key={index}>
299                     <td>{item.time}</td>
300                     <td>{item.weight.toFixed(2)}</td>
301                     <td>{item.conveyorSpeed.toFixed(2)}</td>
302                   </tr>
303                 ))}
304               </tbody>
305             </table>
306           </div>
307         )}
308       </div>
309     );
310   };
```

Рисунок 2.21 – Лістинг реалізації табличного представлення

Поточні значення відображені у вигляді окремих інформаційних карток із підкресленням ключових параметрів технологічного процесу – ваги, швидкості та часу отримання даних. Код реалізації відображення карток з поточними даними наведено на рис. 2.22.

```
{/* Поточні значення */}
{latestValue && isLiveMode && (
  <div className="current-values">
    <div className="value-card">
      <h3>Вага</h3>
      <p className="value">{latestValue.weight.toFixed(2)} кг</p>
    </div>
    <div className="value-card">
      <h3>Швидкість</h3>
      <p className="value">{latestValue.conveyorSpeed.toFixed(2)} м/с</p>
    </div>
    <div className="value-card">
      <h3>Час</h3>
      <p className="value-small">
        {format(new Date(latestValue.timestamp), 'HH:mm:ss')}
      </p>
    </div>
  </div>
)}
```

Рисунок 2.22 – Лістинг реалізації карток з поточними значеннями ваги, швидкості та часу вимірювання

Система забезпечує коректну поведінку навіть у складних умовах нестабільного мережевого з’єднання. Для цього реалізовано:

– автоматичне перепідключення SignalR;

– логування станів з'єднання (onreconnecting, onreconnected, onclose);

– обмеження кількості точок у live-графіку;

– обробку винятків під час HTTP-запитів;

– незалежну роботу live- та historical-режимів.

Таким чином, клієнтська частина є не лише інтерфейсним модулем, але й компонентом, що забезпечує стійку роботу системи та зручність взаємодії оператора з технологічними параметрами.

2.4 Розгортання розробленого програмного забезпечення системи неперервного зважування

Розгортання розробленої системи неперервного зважування передбачає інтеграцію програмного забезпечення з реальним промисловим обладнанням,

налаштування серверного середовища, розміщення бази даних та розгортання клієнтського застосунку у веббраузерах операторів. Особливості розгортання системи визначаються специфікою промислового середовища, що вимагає надійності комунікацій, розмежування мережевих сегментів та відповідності застосованих технологій нормативним вимогам підприємства.

Архітектурно система складається з декількох фізичних вузлів, кожен з яких виконує окрему функцію (рис. 2.23). На рівні промислової мережі функціонує OPC UA-сервер, який є частиною програмно-апаратного комплексу PLC або SCADA-системи. Цей сервер забезпечує публікацію технологічних змінних у мережу та слугує джерелом даних для подальших обчислень. У сегменті серверів підприємства розміщується додаток ASP.NET Core, який виконує роль центрального процесингового вузла: він здійснює періодичне зчитування даних з OPC UA, зберігає їх у базі даних та обслуговує вебзапити клієнтів. Окремо функціонує сервер бази даних MS SQL Server, який забезпечує надійне зберігання та ефективні запити до історичних даних. Клієнтський рівень, що складається з браузерів операторів, підключається до серверної частини через протоколи WebSocket та HTTPS [18, 19].

У фізичній структурі важливою складовою є промислова мережа, в якій розміщено PLC або SCADA-систему з інтегрованим OPC UA-сервером. Він працює в окремому мережевому сегменті, критично важливому з погляду кібербезпеки та вимог надійності. Публікація змінних відбувається через протокол opc.tcp, який забезпечує низьку затримку та стійкість до втрат пакетів. Наявність чіткої структури адресних просторів OPC UA (ns=2;s=...) спрощує інтеграцію із серверною частиною .NET.

На сервері прикладного рівня розгорнуто додаток ASP.NET Core, який запускається у середовищі .NET 8.0 та працює як хост для кількох модулів: фонової служби OPC UA-клієнта, хабу SignalR та REST API. Сервер забезпечує доступність двох основних протоколів – HTTPS та WebSocket – відповідно до потреб клієнтських застосунків.

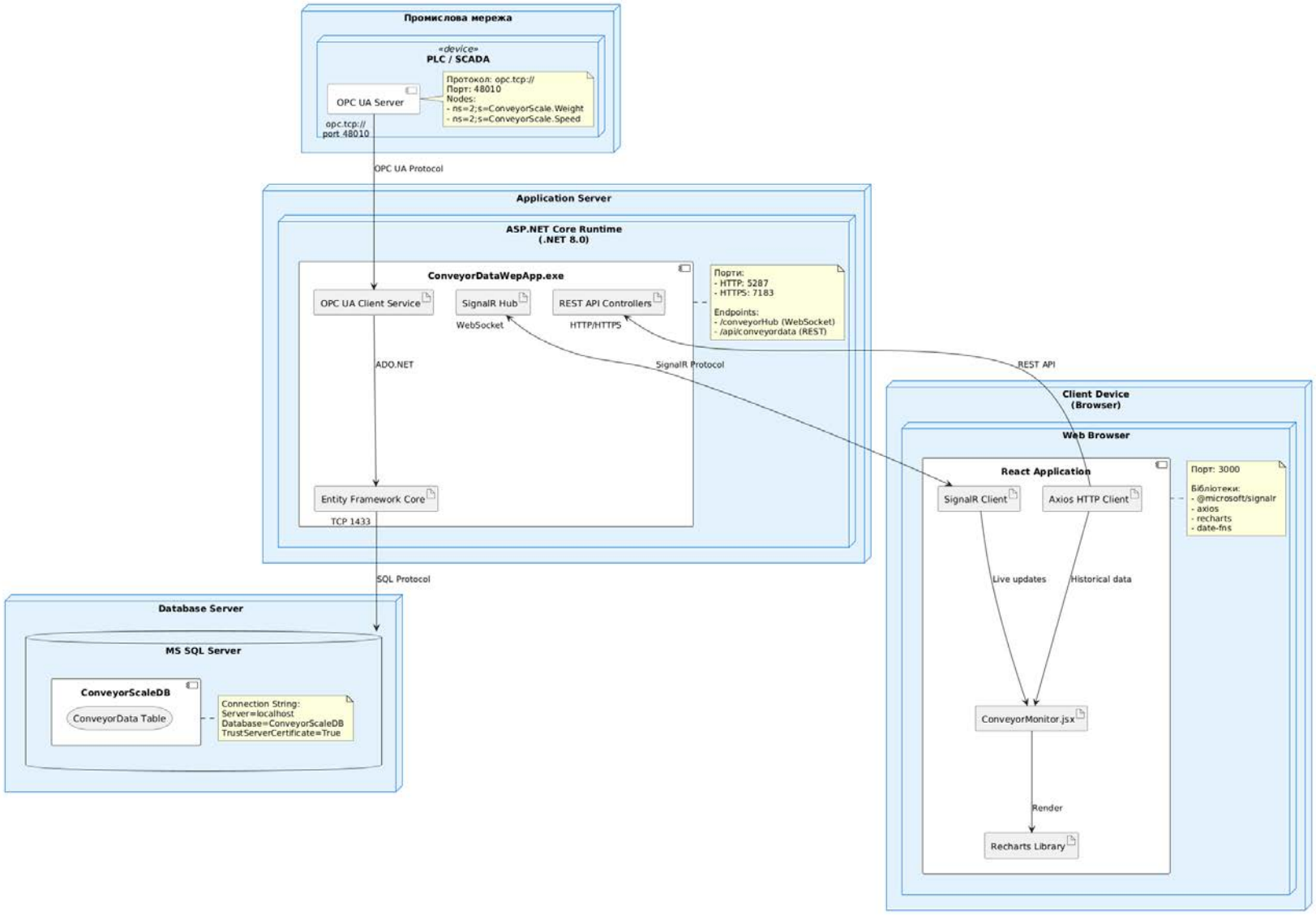

Рисунок 2.23 – Діаграма розгортання UML програмного забезпечення системи неперервного зважування

Фоновий сервіс, запущений у межах цього серверного вузла, забезпечує періодичне зчитування технологічних параметрів і їх подальшу обробку, завдяки чому інформація надходить на клієнтську частину з мінімально можливою затримкою.

У серверній інфраструктурі окремо розміщується база даних MS SQL Server. Такий підхід дає змогу забезпечити чітке розмежування навантаження між веб-сервером і сервером зберігання, що суттєво підвищує масштабованість та надійність. SQL Server приймає запити через стандартний протокол TCP 1433, а використання Entity Framework Core дає змогу формувати оптимізовані SQL-запити і забезпечувати транзакційну цілісність даних.

Фінальним компонентом розгортання є клієнтські пристрої, які отримують доступ до системи через браузер. Усі клієнти працюють у режимі безсерверного рендерингу фронтенду, отримуючи статичні файли React під час першого завантаження, після чого взаємодія з сервером відбувається виключно через WebSocket (SignalR) або HTTPS (REST API). Це дозволяє масштабувати систему практично необмежено щодо кількості одночасних клієнтів, оскільки вся інтенсивна обробка виконується на сервері.

Таким чином, розгортання системи забезпечує чітке розмежування функціональних зон: промислового рівня (OT), серверної інфраструктури (IT), бази даних та клієнтського рівня. Така багаторівнева структура не лише покращує продуктивність і надійність, але й забезпечує відповідність системи вимогам кібербезпеки, прийнятим у промислових підприємствах. Внутрішня логіка комунікацій, відображена на діаграмі розгортання, демонструє узгодженість архітектурних рішень і відображає реальний сценарій роботи системи в умовах промислової експлуатації.

2.5 Забезпечення кібербезпеки системи неперервного зважування

Із розвитком промислової автоматизації та інтеграцією виробничих систем у корпоративні інформаційні інфраструктури питання кібербезпеки набувають особливої актуальності. Сучасні конвеєрні вагові системи, що працюють у

контексті промислового Інтернету речей, оперують критично важливими технологічними даними, які впливають на точність обліку матеріальних потоків, формування звітності та виконання контролю якості процесів. Тому комплекс заходів щодо захисту конфіденційності, цілісності та доступності цих даних є невід'ємною складовою проєктування системи.

Архітектура розробленої системи передбачає розділення мереж на технологічний (OT) і інформаційний (IT) сегменти. Таке розмежування є ключовою умовою мінімізації ризиків несанкціонованого доступу до промислового обладнання. OPC UA-сервер, розміщений у середовищі PLC або SCADA-системи, працює виключно в зоні OT, а доступ до нього здійснюється через контрольовані шлюзи або міжмережеві екрани. Це знижує ймовірність прямого впливу користувачів або зовнішніх мереж на критичні технологічні пристрої. Протокол OPC UA сам по собі містить засоби безпеки, зокрема сертифікацію та шифрування, однак у контексті промислового середовища часто застосовуються спрощені конфігурації з анонімною автентифікацією. Тому захист на рівні сегментації мережі залишається фундаментальним елементом безпеки.

В розробленому програмного забезпеченні реалізовані також наступні базові механізми безпеки:

– HTTPS: Використання TLS для шифрування трафіку між клієнтом і сервером;

– CORS: Конфігурація Cross-Origin Resource Sharing для контролю доступу з різних доменів;

– параметризовані запити: Entity Framework автоматично параметризує SQL запити, запобігаючи SQL-ін'єкціям.

На серверному рівні ASP.NET Core-додаток функціонує в ізольованому середовищі Application Server, що обробляє дані з OPC UA-сервера, зберігає їх у базі даних та надає доступ клієнтам. Важливим аспектом безпеки є використання протоколу HTTPS для всіх REST API-запитів, що усуває ризики перехоплення або модифікації даних у процесі передачі. Хоча дані з вагових пристроїв не

становлять конфіденційної інформації в класичному розумінні, їх цілісність має значний вплив на економічні процеси підприємства. Тому забезпечення криптографічного захисту є обов’язковим.

Особливу увагу слід приділити механізму роботи SignalR. Канал обміну даними в реальному часі використовує WebSocket-з’єднання, яке також має бути захищеним за допомогою HTTPS (WSS). Це забезпечує захист від атак типу «man-in-the-middle», ін’єкцій даних і несанкціонованих підключень [17]. Реалізований компонент клієнтського застосунку підтримує автоматичне перепідключення та обробку подій зміни стану мережі, що сприяє стабільності роботи, однак не замінює належної автентифікації та контролю доступу. У промислових умовах система може бути додатково розширена механізмами авторизації на основі JWT-токенів або інтеграції з корпоративним доменом Active Directory .

База даних MS SQL Server також відіграє ключову роль у забезпеченні безпеки. Оскільки вона зберігає повний історичний масив вимірювань, будь-які порушення цілісності або несанкціоновані модифікації можуть призвести до економічних втрат або помилок у звітності. Для запобігання таким ризикам застосовується обмеження доступу до SQL Server за принципом найменших привілеїв, розмежування ролей, а також шифрування з’єднання між вебдодатком та сервером бази даних. Додатковим засобом захисту є збереження журналів аудиту SQL Server, що дає змогу відстежувати зміни структури бази даних та дії користувачів.

На клієнтському рівні браузерний застосунок працює в контексті, ізольованому засобами браузера, проте він може бути вразливим до атак типу XSS або CSRF у випадку неправильного налаштування серверної частини. Для мінімізації цих ризиків застосовується політика CORS, обмеження походження запитів лише дозволеними доменами та коректна обробка HTTP-заголовків у backend-застосунку. Реалізація політики CORS у Program.cs чітко окреслює, які клієнти можуть встановлювати WebSocket-з’єднання з сервером.

Важливо також зазначити, що загальна модель безпеки системи ґрунтується на припущенні про контрольованість виробничої та серверної інфраструктури. У промислових умовах додатково можуть застосовуватися методи виявлення вторгнень (IDS/IPS), системи сегментації SDN, моніторинг мережевого трафіку та централізоване керування подіями безпеки (SIEM). Це дозволяє оперативно виявляти підозрілі дії, а також забезпечувати відповідність корпоративним політикам безпеки.

Для промислового розгортання рекомендується також впровадити:

– аутентифікацію: зокрема, шляхом інтеграції з Identity Server або Azure AD для управління користувачами;

– авторизацію: реалізація ролей та політик доступу до різних кінцевих точок (endpoints);

– механізми OPC UA Security: увімкнення режиму безпеки OPC UA з взаємною автентифікацією сертифікатів;

– Rate Limiting: обмеження частоти запитів для захисту від DDoS атак;

– аудит: логування всіх дій користувачів для забезпечення покращеного моніторингу.

Узагальнюючи, забезпечення кібербезпеки системи неперервного зважування досягається за рахунок комплексного підходу, що охоплює фізичне розмежування мережевих сегментів, шифрування каналів передачі, контроль доступу до серверів та бази даних, а також мінімізацію поверхні атаки на рівні клієнтських застосунків. Запропонована архітектура відповідає вимогам промислових підприємств та забезпечує збалансовану взаємодію між безпекою, продуктивністю та гнучкістю системи.

Висновки до розділу:

У другому розділі було виконано проєктування програмного забезпечення системи неперервного зважування на конвеєрних лініях, що охоплює вибір технологій, побудову архітектури, опис алгоритмів взаємодії між компонентами та визначення основних механізмів розгортання системи у виробничому

середовищі. Проведений аналіз технологічних платформ дав можливість обґрунтувати застосування ASP.NET Core як базової серверної технології, MS SQL Server як системи керування реляційними даними та React як оптимального інструменту для побудови інтерактивного веб-інтерфейсу. Такий вибір забезпечує надійність, продуктивність, масштабованість та технологічну сумісність програмного забезпечення з промисловими системами.

У межах проєктування архітектури було сформовано багаторівневу модель, що включає серверну частину для збору та обробки даних, базу даних для зберігання історичних вимірювань та клієнтську частину для відображення як поточних, так і архівних параметрів. Архітектура побудована за принципами розподілених систем із чітким розмежуванням відповідальності між компонентами, що сприяє підвищенню надійності та спрощує подальший супровід і масштабування. Окрему увагу було приділено інтеграції з промисловим обладнанням через протокол OPC UA, що забезпечує стандартизований та безпечний спосіб отримання технологічних змінних.

На основі розроблених діаграм компонентів, послідовності та розгортання було детально описано взаємодію між елементами системи. Зокрема, фоновий OPC UA-клієнтський сервіс, що працює в середовищі ASP.NET Core, забезпечує регулярне зчитування технологічних параметрів, їх перевірку, збереження у базі даних та трансляцію в реальному часі через SignalR. Клієнтська частина системи реалізує два режими роботи: live-режим з оновленням даних у потоковому форматі та історичний режим із можливістю фільтрації архіву за довільними часовими інтервалами. Такий підхід забезпечує повноцінну підтримку операційного моніторингу та аналітичної обробки даних.

У розділі також було висвітлено питання кібербезпеки системи. Аналіз показав, що запропонована інфраструктура відповідає вимогам сучасних промислових мереж завдяки розмежуванню IT- та OT-сегментів, застосуванню мережевих політик доступу, використанню захищених протоколів передачі даних та впровадженню механізмів контролю доступу на рівні бази даних і серверної взаємодії. Комплексність запропонованих заходів забезпечує стійкість

системи до типових загроз та сприяє її надійній роботі у виробничих умовах.

Таким чином, результати другого розділу демонструють завершене проєктування програмного забезпечення системи неперервного зважування, що включає обґрунтований вибір технологій, побудову функціональної та технічної архітектури, моделювання ключових процесів та розроблення рішень з розгортання й забезпечення безпеки. Сформована проєктна модель є основою для реалізації програмного комплексу та подальшої експлуатації системи у промисловому середовищі.

РОЗДІЛ 3
ПРАКТИЧНА АПРОБАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ НЕПЕРЕРВНОГО ЗВАЖУВАННЯ

3.1 Тестування серверної частини та API

З метою практичної перевірки працездатності розробленого програмного забезпечення системи неперервного зважування гірничої маси на конвеєрній лінії було виконано два етапи апробації: тестування серверної частини (REST API та механізмів обробки OPC UA-даних) і тестування клієнтського React-застосунку, який забезпечує взаємодію оператора з інформаційною системою. Такий підхід дозволяє оцінити коректність роботи обох частин комплексу, а також перевірити узгодженість між ними.

Перевірку функціональності бекенд-компонентів здійснено на основі Swagger UI, що генерується автоматично в середовищі ASP.NET Core. Це дає змогу досліджувати кінцеві точки API, оцінювати формат вхідних і вихідних даних та перевіряти коректність роботи бізнес-логіки без необхідності використання сторонніх інструментів. Інтерфейс Swagger доступний за адресою:

https://localhost:7183/swagger

На рис. 3.1 бпредставлено вигляд головної сторінки Swagger UI з переліком доступних ендпоїнтів.

Серед основних REST API, що підлягали тестуванню, можна виділити отримання останніх записів, вибірку даних за часовим діапазоном, формування статистики та та тестування SignalR-хабу.

Кінцева точка:

GET /api/conveyordata/latest

призначена для отримання останніх N записів, збережених у базі даних. Під час тестування перевірено, що сервер коректно повертає дані у JSON-форматі,

впорядковує результати за часовою міткою та забезпечує відповідність значень моделі ConveyorData (рис. 3.2).

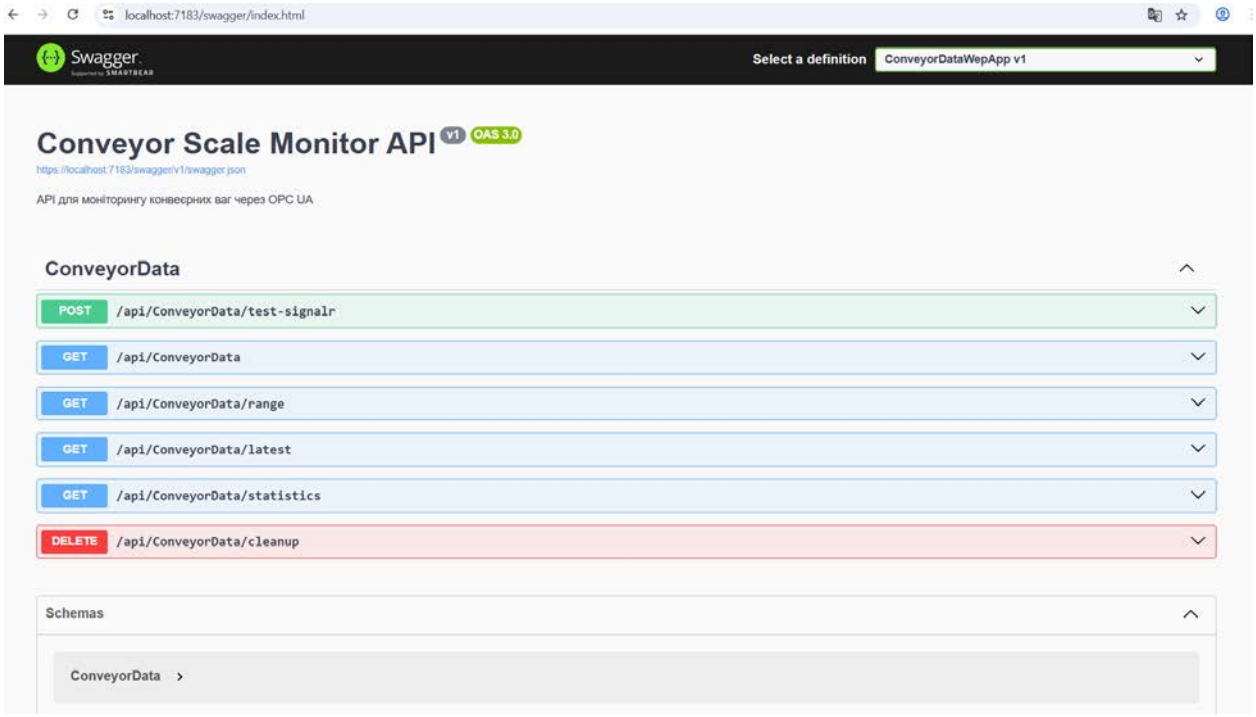

Рисунок 3.1 – Головна сторінка Swagger UI з переліком доступних кінцевих точок API

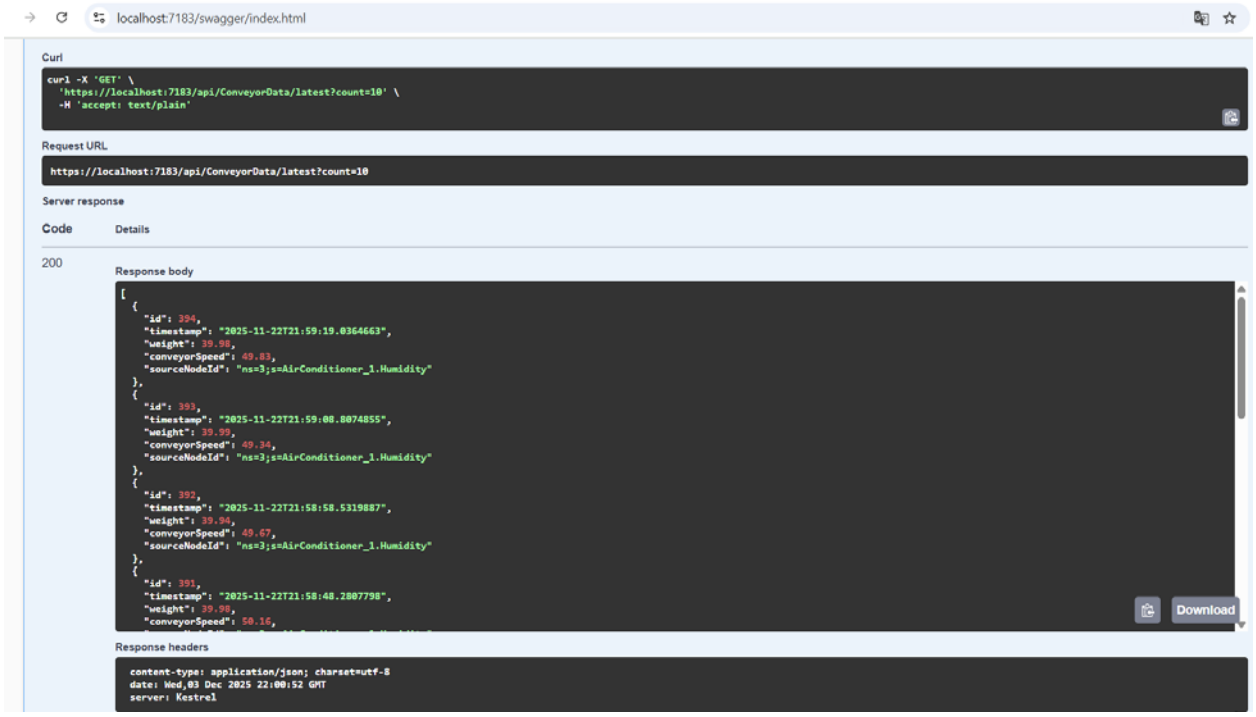

Рисунок 3.2 – Результати тестування кінцевої точки для отримання отримання останніх N записів у БД

Задана кількість точок задається параметром count.

У результатах тестових викликів (див. рисунок 3.2) підтверджено:

– правильність структури відповіді;

– відповідність одиниць вимірювання;

– коректність відображення часових міток у форматі UTC.

Кінцева точка

GET /api/conveyordata/range

використовується для фільтрації даних за часовим діапазоном. Вона є критично важливою для реалізації аналітичних можливостей системи. Під час тестування було виконано вибірку різних інтервалів та перевірено правильність фільтрації, відсутність пропущених записів та коректну роботу з великими наборами даних.

На рисунку 3.3 наведено вікно для введення початку та завершення діапазону фільтрації.

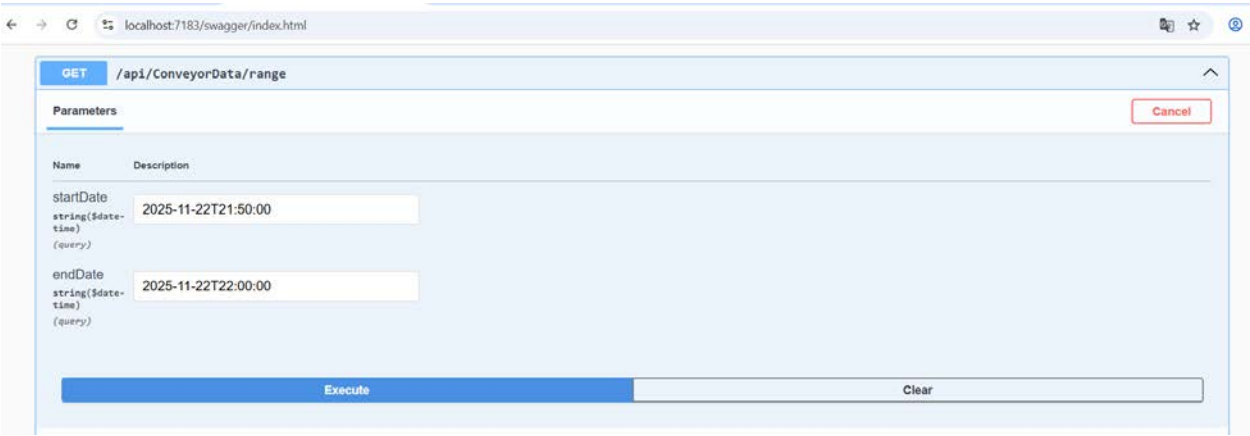

Рисунок 3.3 – Налаштування діапазону фільтрації даних вимірювань за часом у Swagger UI

На рисунку 3.4 наведено формат запиту, код відповіді та приклад вибірки за період між двома датами, сформованої через Swagger. Отриманий результат підтверджує коректність роботи даної кінцевої точки.

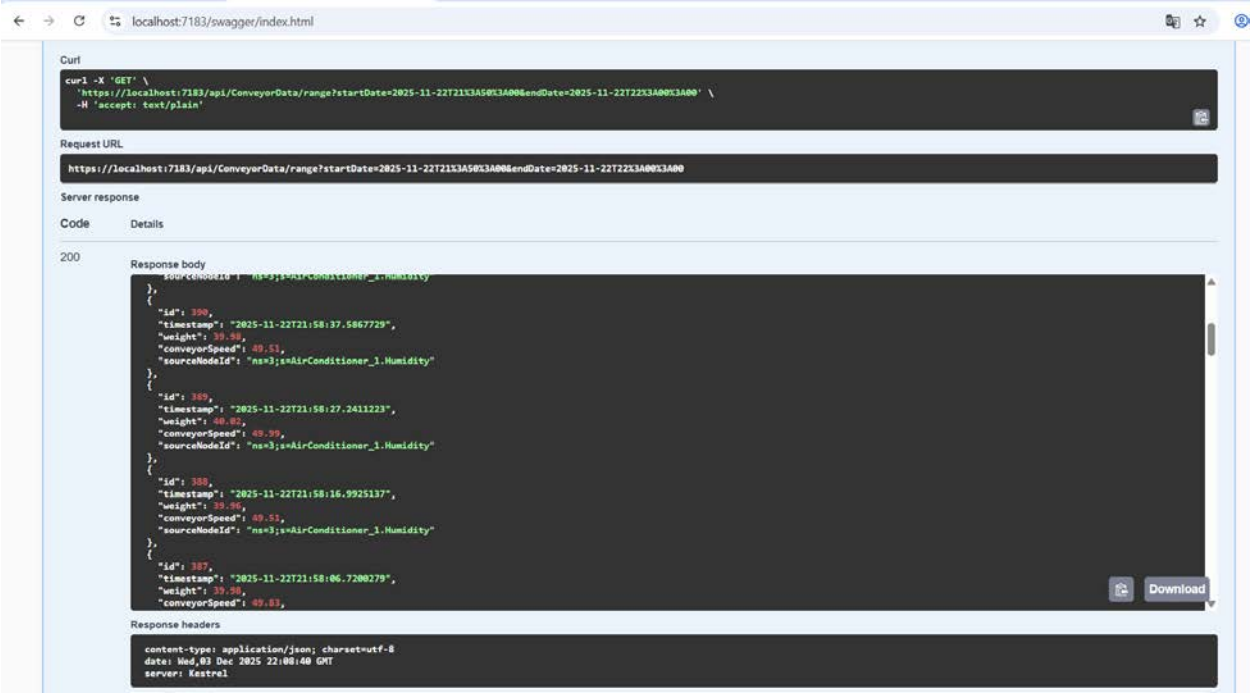

Рисунок 3.4 – Результати тестування кінцевої точки для фільтрації даних за часовим діапазоном

Кінцева точка

GET /api/conveyordata/statistics

реалізує агрегацію даних (мінімум, максимум, середнє значення, сумарна маса) за визначений період (рис. 3.5).

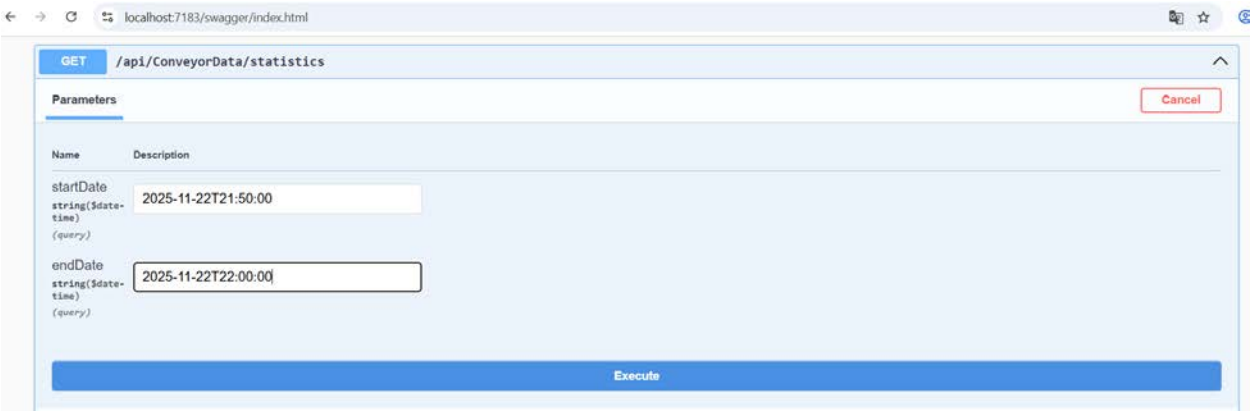

Рисунок 3.5 – Налаштування періоду фільтрації даних вимірювань за часом для отримання статистики у Swagger UI

Під час тестування перевірено коректність математичних обчислень та відповідність результатів розрахунків значенням, отриманим вручну із вибірки (рис. 3.6).

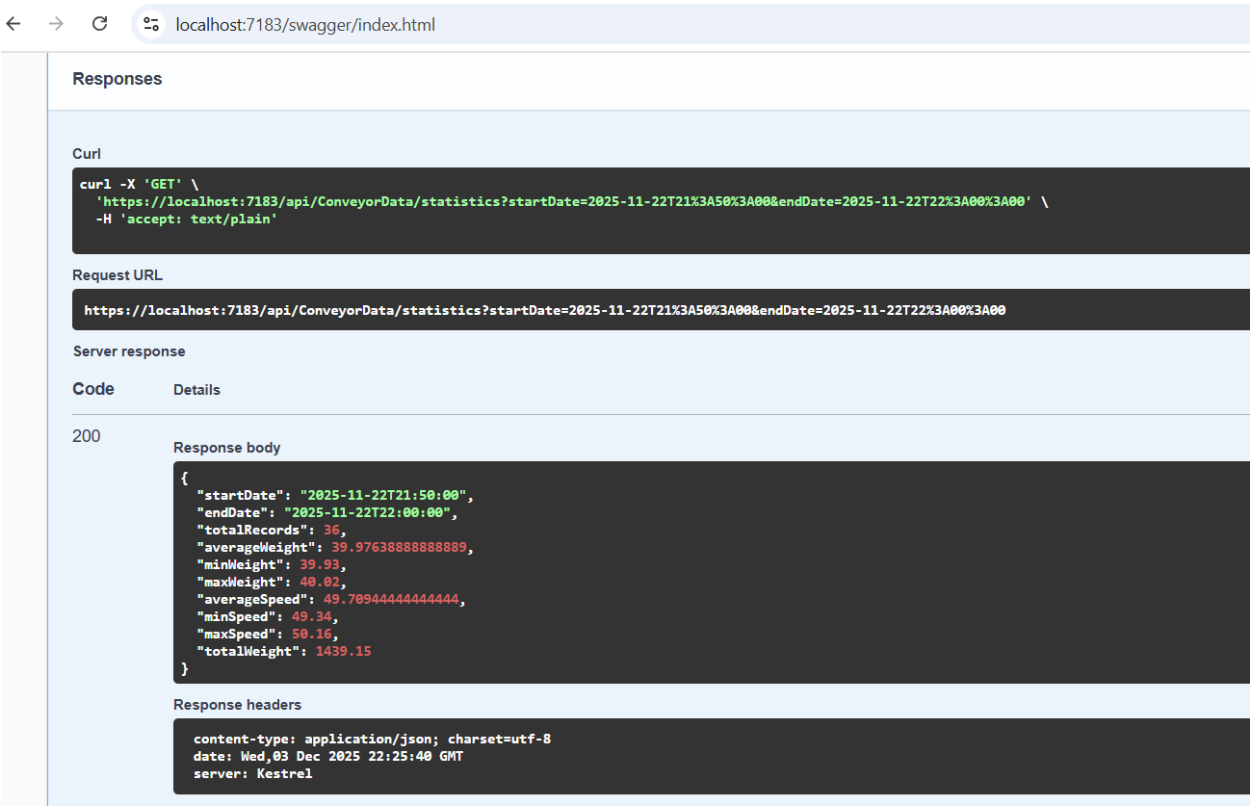

Рисунок 3.6 – Результати тестування кінцевої точки для отримання статистики по виміряним значенням

Окремо протестовано коректність роботи WebSocket-з’єднання за допомогою тестової кінцевої точки

POST /api/conveyordata/test-signalr

Виклик цього методу формує тестове повідомлення з випадковими значення ваги та швидкості конвеєра і транслює його всім активним клієнтам через хаб ConveyorDataHub. На рисунку 3.7 наведено відповідь при виконанні даного тестового методу, а на рис. 3.8 – результат відображення тестових даних у клієнтському застосунку.

Як видно з порівняння даних на рис. 3.7 та 3.8, згенеровані під час тестування дані було успішно передані до клієнту через SignalR та відображені на інтерфейсі клієнтського додатку.

Рисунок 3.7 – Результати тестування кінцевої точки для перевірки зв’язку через SignalR

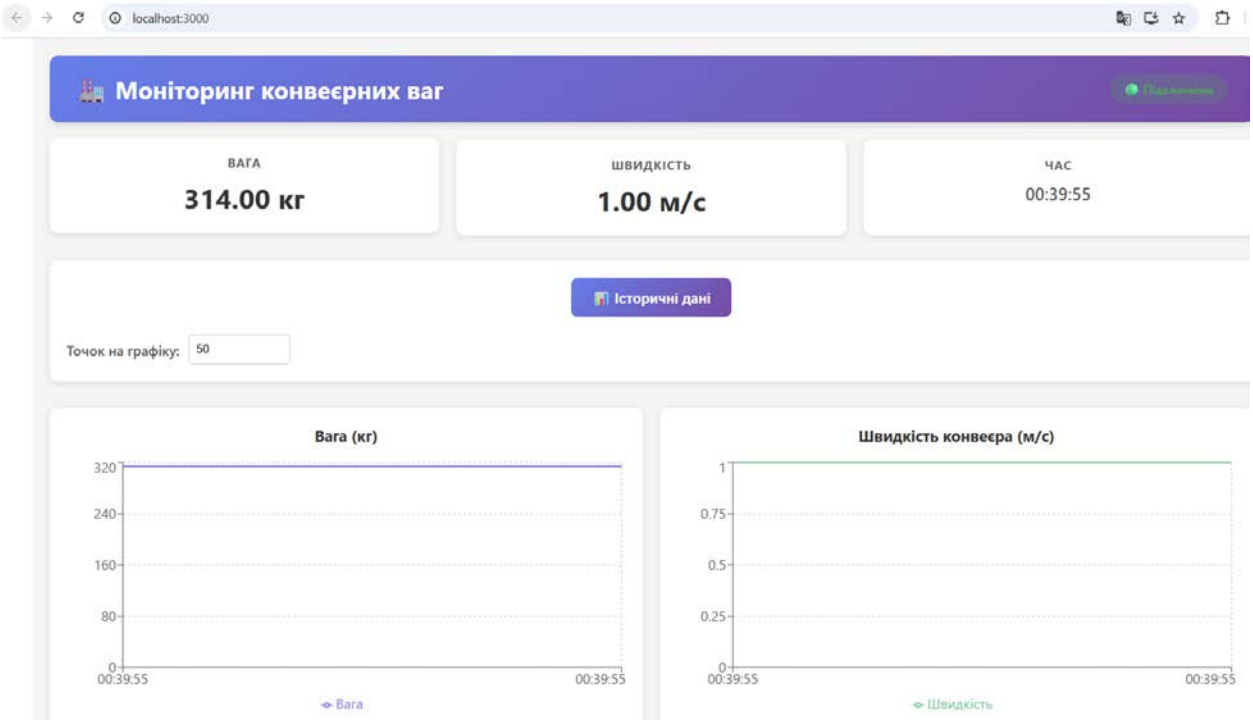

Рисунок 3.8 – Результати тестування зв’язку з сервером з боку клієнтського застосунку

Результати тестування показали, що серверна частина функціонує стабільно, коректно обробляє запити та забезпечує надійний канал передачі даних у режимі реального часу.

3.2 Практичне тестування клієнтського React-застосунку

Клієнтський застосунок забезпечує оператора зручним інтерфейсом для моніторингу показників конвеєрних ваг у двох режимах: відображення живих даних та аналіз історичних значень. Інтерфейс розроблено з урахуванням вимог ергономіки та простоти використання, що дозволяє мінімізувати час навчання персоналу.

Для початку роботи користувачу необхідно відкрити вебзастосунок у браузері за адресою:

http://localhost:3000

Після завантаження інтерфейсу відображається головна сторінка системи, що включає: індикатор стану підключення SignalR, панель керування режимами, блок поточних значень, графіки та таблицю останніх записів. На рис. 3.9 наведено приклад інтерфейсу у режимі реального часу з оновленням графіка.

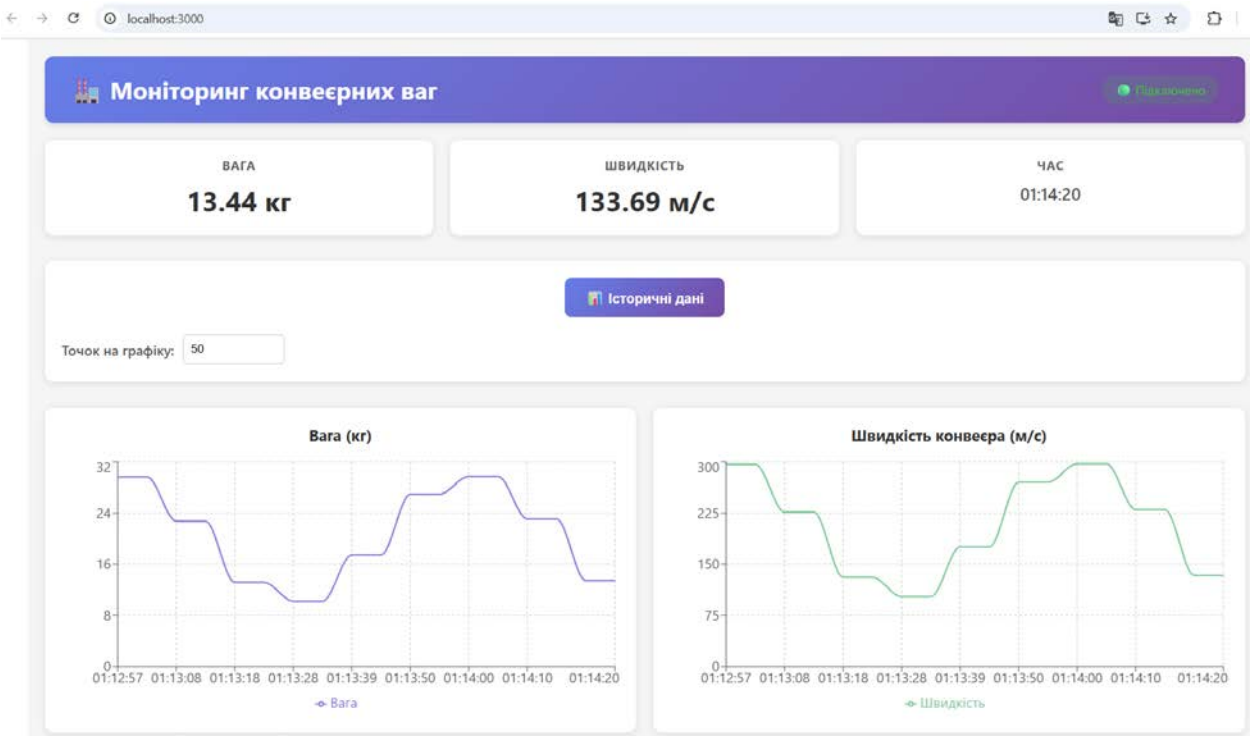

Рисунок 3.9 – Головна сторінка клієнтської частини програмного забезпечення неперервного зважування

У верхній частині сторінки розташовано індикатор з’єднання. Зелене маркування означає, що встановлено активне WebSocket-підключення з сервером, і система готова отримувати дані в реальному часі. У разі втрати з’єднання індикатор змінює стан на червоний, а застосунок автоматично робить спроби перепідключення.

Режим Live є основним режимом моніторингу. Він використовується для оперативного спостереження за технологічними параметрами, що надходять із конвеєрних ваг у реальному часі.

Після встановлення WebSocket-з’єднання на екран починають надходити дані, сформовані серверною частиною та передані через SignalR. Інтерфейс відображає:

1. Поточні значення – у верхній частині екрана користувач бачить останнє виміряне значення ваги, швидкості конвеєра та час надходження даних (див. рисунок 3.11).

2. Графік ваги – відображає залежність ваги від часу за останні N значень. Оновлюється автоматично з кожним оновленням даних.

3. Графік швидкості конвеєра – аналогічно демонструє динаміку змін швидкості у реальному часі (див. рис. 3.9).

4. Таблиця останніх записів – містить інформацію про останні надходження даних, що дає змогу оператору швидко відстежити динаміку без перегляду графіка (рис. 3.10).

Користувач може змінювати кількість точок, які відображаються на графіку, за допомогою відповідного параметра «Точок на графіку». Це впливає лише на локальне відображення без зміни серверних налаштувань.

Під час експлуатації у виробничому середовищі можливі короткочасні втрати зв’язку. Застосунок забезпечує автоматичне перепідключення до SignalR-хабу у Live-режимі, про що сигналізується відповідним станом індикатора (див. рисунок 3.18).

Тестування live-режиму показало, що застосунок стійко переносить втрату з’єднання, автоматично перепідключається і коректно продовжує отримання даних.

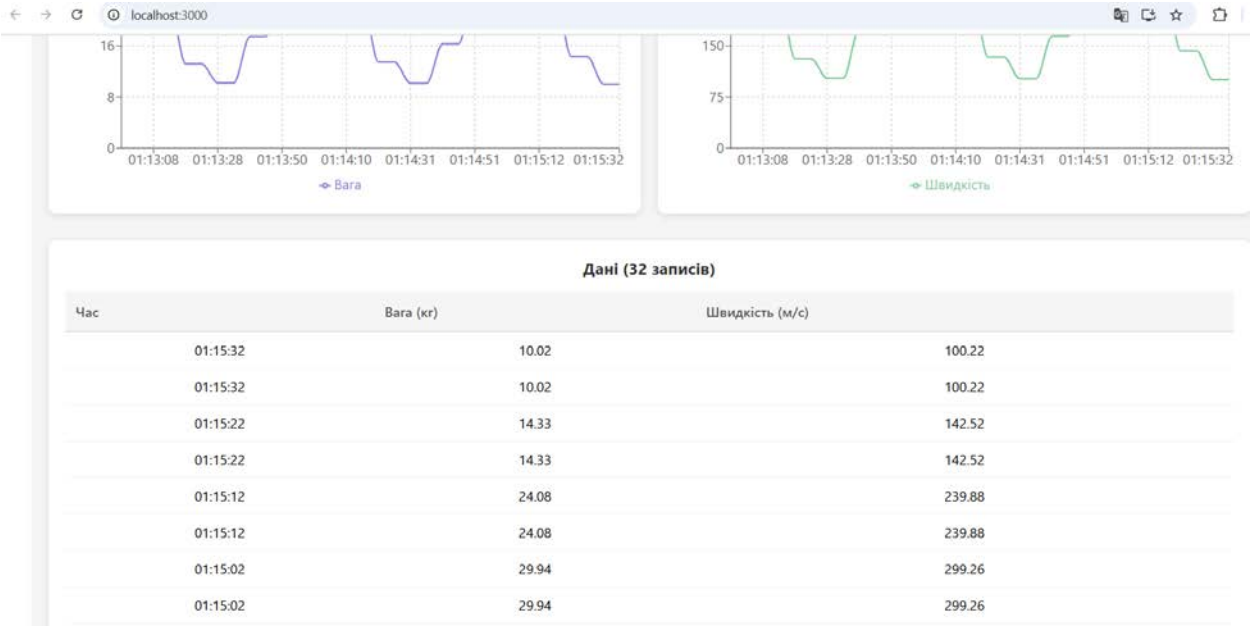

Рисунок 3.10 – Відображення останніх отриманих виміряних значень у вигляді таблиці

Для переходу від live-моніторингу до аналізу історичних даних користувачу потрібно натиснути кнопку «Історичні дані». Після цього застосунок припиняє оновлення інформації в реальному часі і переходить до роботи через REST API (рис. 3.11). Оператор має можливість:

– завантажити останні N значень;

– виконати вибірку даних за часовим діапазоном;

– переглянути графічне та табличне представлення результатів;

– порівняти динаміку показників за різні періоди.

Функціональність реалізована через REST API, що було перевірено на коректність роботи з різними інтервалами часу.

Фільтри призначені для отримання структурованих вибірок історичних вимірювань. Користувач може налаштувати такі параметри (рис. 3.11):

1. Початок періоду – часова позначка, з якої слід почати вибірку. Формат введення відповідає datetime-local.

2. Кінець періоду – час, до якого слід включати дані у вибірку.

3. Кількість записів.

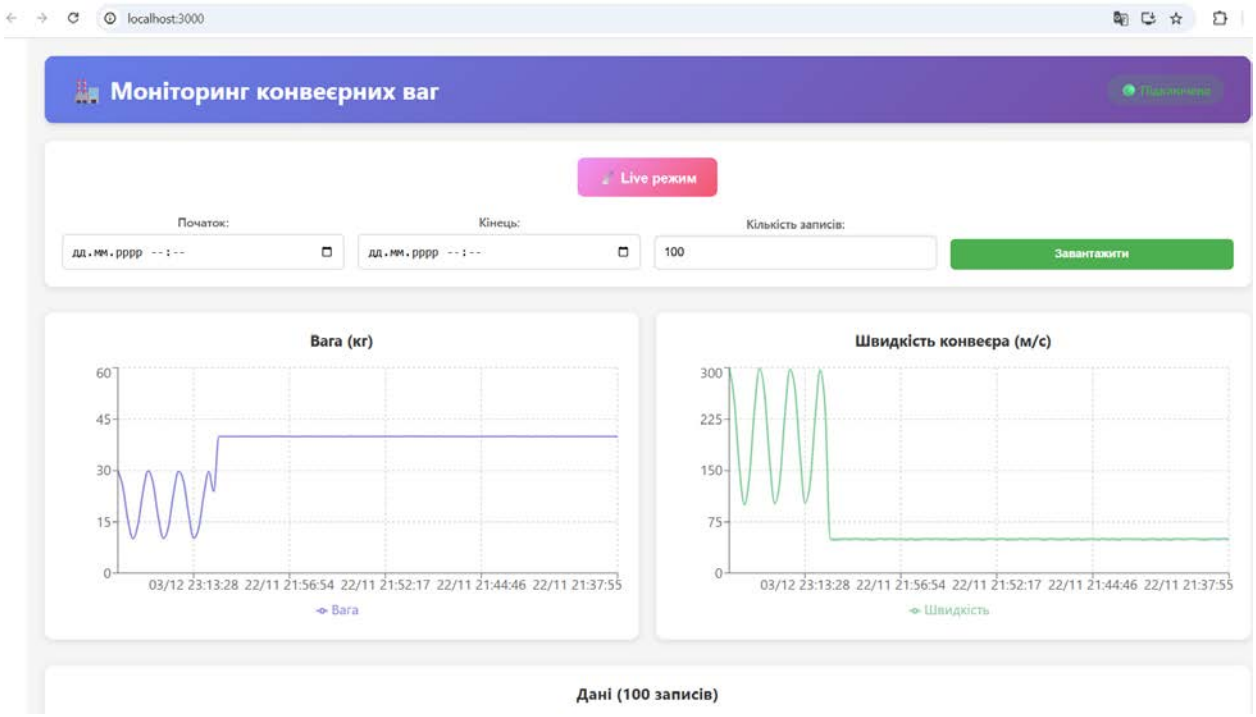

Рисунок 3.11 – Сторінка відображення історичних даних програмного забезпечення неперервного зважування

У разі відсутності діапазону даних система може завантажити останні N значень, використовуючи ендпоїнт:

GET /api/conveyordata/latest.

Після налаштування параметрів користувач повинен натиснути кнопку «Завантажити», після чого застосунок звертається до відповідного API-методу:

GET /api/conveyordata/range?startDate=...&endDate=...

Результати вибірки відображаються у вигляді графіка та таблиці (рис. 3.12), причому дані формуються незалежно від потоку SignalR. Це дає змогу використати застосунок для аналітичних цілей, порівняння періодів або ретроспективного аналізу.

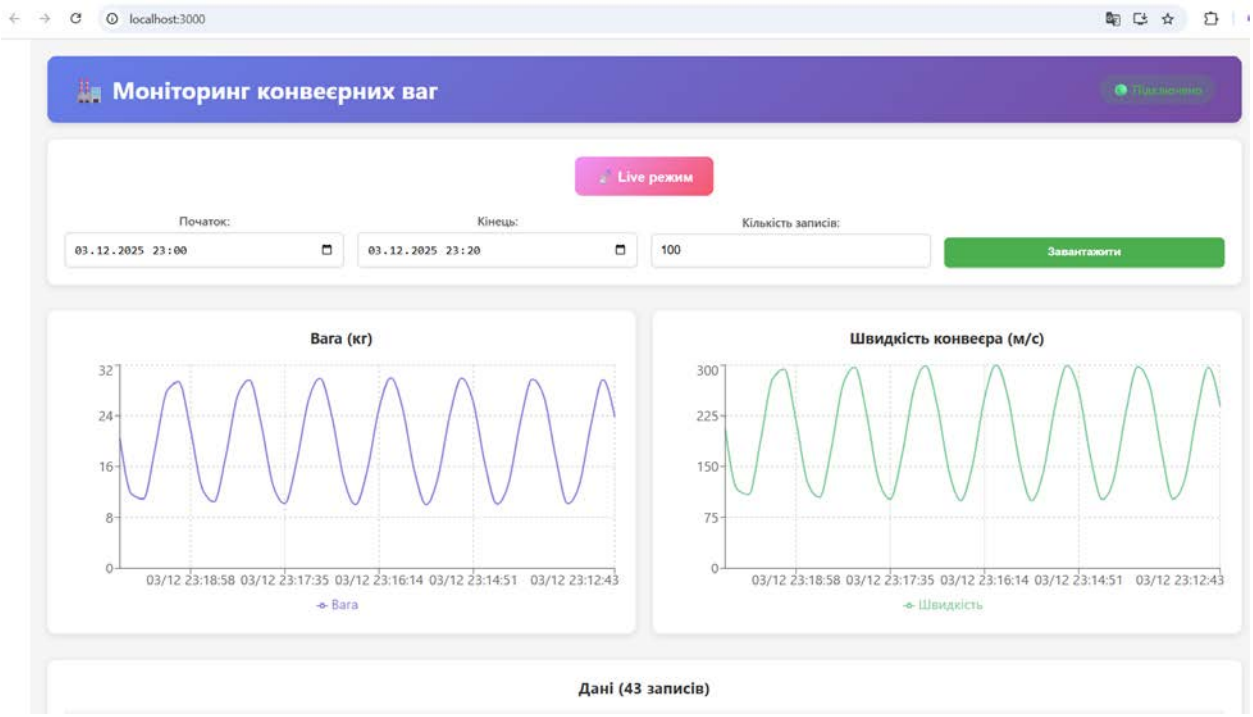

Рисунок 3.12 – Відображення результатів фільтрації вимірювань за датою

Проведена апробація розробленого програмного забезпечення демонструє, що система є зручною у використанні, підтримує інтуїтивні механізми перемикання між режимами та дозволяє виконувати як оперативний моніторинг, так і історичний аналіз. Структура інтерфейсу забезпечує чіткий поділ функцій, а наявність фільтрів і графічної візуалізації робить застосунок придатним для практичного використання у виробничих умовах.

Висновки до розділу:

У межах третього розділу було здійснено всебічну практичну апробацію розробленого програмного забезпечення системи неперервного зважування, що включала тестування серверних компонентів, оцінювання коректності роботи REST API, перевірку механізмів потокової передачі даних через SignalR, а також повну перевірку функціональності клієнтського веб-інтерфейсу на основі React. Проведене тестування підтвердило працездатність і відповідність створеної системи вимогам, визначеним у попередніх розділах.

На етапі перевірки серверної частини було встановлено, що реалізований бекенд на базі ASP.NET забезпечує стабільне зчитування даних із промислового OPC UA-сервера, їх коректне збереження у базі даних Microsoft SQL Server та надання структурованих даних через набір REST-ендпоїнтів. Тестування API засвідчило відсутність збоїв у реалізації основних операцій — формування історичних вибірок, обробки параметрів фільтрації, обчислення статистичних показників та отримання останніх значень. Окремо було перевірено поведінку системи при навмисних помилкових запитах і нестандартних параметрах – усі ендпоїнти демонструють коректне оброблення виняткових ситуацій.

Значну увагу приділено випробуванню механізмів реального часу. Використання технології SignalR дало змогу забезпечити стабільне двостороннє WebSocket-з'єднання між клієнтом і сервером, автоматичне відновлення зв'язку після розривів та гарантовану доставку даних кожного циклу зчитування OPC UA. Практичні експерименти засвідчили, що система демонструє мінімальну затримку при передачі даних та здатна підтримувати регулярне оновлення інформації з інтервалами до однієї секунди, що відповідає вимогам режиму оперативного моніторингу.

Окремий блок практичної апробації стосувався роботи клієнтського застосунку, реалізованого засобами React. У ході тестування підтверджено, що інтерфейс коректно реагує на зміну режимів, забезпечує плавне перемикання між live-моніторингом та режимом історичних даних, а також правильно відображає інформацію у графічній та табличній формах. Механізм фільтрації за часовими параметрами працює правильно, що дозволяє виконувати ретроспективний аналіз виробничих процесів. Структура інтерфейсу виявилася зручною для користувача, а візуальні засоби Recharts забезпечили інформативне графічне подання технологічних параметрів.

Узагальнюючи результати апробації, слід зазначити, що розроблена система продемонструвала високу надійність, коректність роботи та відповідність функціональним вимогам. Система забезпечує повний цикл оброблення даних – від отримання сигналів із промислового обладнання до їх візуалізації в режимі реального часу. Створений програмний продукт може

бути інтегрований у промислове середовище та застосований у виробничих умовах для моніторингу параметрів конвеєрних ваг, що підтверджує успішність розробки та придатність системи до практичного використання.

ВИСНОВКИ

У ході виконання дипломної роботи було здійснено повний цикл розроблення, проєктування та практичної апробації програмного забезпечення для системи неперервного зважування гірничої маси на конвеєрних вагових установках. Систему було створено з урахуванням сучасних вимог до промислових інформаційних платформ, включаючи надійність, масштабованість, інтегрованість із промисловими протоколами та зручність користувацької взаємодії. Результати роботи свідчать про досягнення мети дослідження та виконання всіх поставлених завдань.

У аналітичній частині здійснено огляд апаратних рішень для вимірювання масових потоків, включаючи спеціалізовані вагові контролери, системи на основі ПЛК загального призначення та промислові IoT-шлюзи. Проаналізовано їх переваги, обмеження та особливості застосування у гірничо-збагачувальній промисловості. Досліджено сучасні програмні продукти для моніторингу конвеєрних ваг та визначено ключові недоліки існуючих систем – закритість архітектур, обмежені можливості інтеграції, недостатня гнучкість у візуалізації та відсутність механізмів роботи з даними в реальному часі на відкритих стандартних протоколах. На основі аналізу обґрунтовано необхідність створення відкритої модульної системи, що поєднує OPC UA як промисловий протокол та сучасні веб-технології.

У другому розділі розроблено функціональні й нефункціональні вимоги до системи, побудовано її архітектурну модель та обґрунтовано вибір технологій: ASP.NET Core – як серверної платформи з високою продуктивністю, MS SQL Server – як надійної реляційної СУБД, React – для реалізації інтерактивного клієнтського застосунку, SignalR – для каналів реального часу, а також OPCFoundation.NetStandard.Opc.Ua – для реалізації промислового протоколу взаємодії із обладнанням. Архітектурна модель включає фонову службу OPC UA, модуль потокової передачі даних, REST API, шар доступу до даних, базу даних та веб-клієнт. Сформовано UML-

діаграми варіантів використання, компонентів, послідовності та розгортання, що забезпечило всебічне документування артефактів, які виникали у процесі проєктування системи.

У практичній частині було реалізовано програмне забезпечення відповідно до запропонованої архітектури. Розроблений сервер здійснює стабільне підключення до OPC UA-сервера, періодичне зчитування ваги та швидкості конвеєра, їх збереження у базі даних та трансляцію оновлень через канал реального часу. REST API забезпечує доступ до історичних даних, статистики та вибірок з фільтрами. Клієнтський застосунок React забезпечує відображення live-параметрів, побудову інтерактивних графіків та виконання аналітичних запитів до історичних даних. Проведена практична апробація підтвердила стабільну роботу всіх компонентів, відповідність функціональним вимогам та високу швидкодію системи.

Окрему увагу приділено питанням кібербезпеки. У роботі проаналізовано потенційні загрози, притаманні системам реального часу з використанням OPC UA та веб-технологій. Запропоновано комплекс рекомендацій, що включають захищене розгортання серверів, використання HTTPS, контроль сертифікатів OPC UA, застосування політик CORS, аудит доступу та впровадження зонованої мережевої моделі.

Узагальнюючи результати роботи, можна стверджувати, що створена система неперервного зважування здатна забезпечити надійний моніторинг технологічних параметрів, має відкриту та масштабовану архітектуру, підтримує роботу в реальному часі та надає оператору зручні засоби для перегляду й аналізу як поточних, так і історичних даних. Отримані результати підтверджують практичну цінність розробленого програмного забезпечення та його можливість інтеграції у виробничі процеси підприємств гірничо-збагачувальної промисловості.

Робота має подальший потенціал розвитку: інтеграцію алгоритмів прогнозного аналізу, підключення мобільних клієнтів, розширення набору промислових датчиків, автоматизацію формування звітів та впровадження

засобів машинного навчання для оцінки ефективності транспортування матеріалу. У сукупності це робить розроблену систему платформою, придатною для масштабування та подальшого розвитку у напрямку Індустрії 4.0.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

-

1. OPC Foundation. OPC UA Online Reference [Електронний ресурс]. – URL: https://reference.opcfoundation.org (дата звернення: 27.11.2025).

2. Mahnke W., Leitner S.-H., Damm M. OPC Unified Architecture. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. URL: https://doi.org/10.1007/978-3-540-68899-0 (дата звернення: 27.11.2025).

3. OPC UA – Unified Architecture: The Everyman's Guide to the Most Important Information Technology in Industrial Automation. Createspace Independent Publishing Platform, 2016. 170 с.

4. Siemens Milltronics BW500 and BW500/L Belt Scale Intergrator - Process Instruments. Process Instruments. URL: https://www.processinstrumentsolutions.co.uk/product/siemens-integrator-milltronics-bw500-and-bw500-l/ (дата звернення: 24.11.2025).

5. Siemens AG. SIWAREX WP231 Operating Instructions. – Digital documentation. – Siemens, 2022. URL: https://cache.industry.siemens.com/dl/files/056/90229056/att_70860/v1/A5E31238908A-02_SIWAREX_WP231_GHB_en_en-US.pdf (дата звернення: 27.11.2025).

6. Static & dynamic scales. Qlar. URL: https://www.qlar.com/technologies/weighing/static-and-dynamic-scales (дата звернення: 27.11.2025).

7. Ramsey™ Micro-Tech 9000 Electronics Platform Ramsey Micro-Tech 9100 Static Weight Integrator. Thermo Fisher Scientific - DE. URL: https://www.thermofisher.com/order/catalog/product/RAMSEY9100 (дата звернення: 27.11.2025).

8. Wägetechnik, Durchsatzmessung und Metallsuchtechnik in Thüringen – Hoferick Engineering GmbH. URL: http://www.he-gmbh.info/media/pdf/1912603_mt3100_manual_ben.pdf (дата звернення: 27.11.2025).

9. WY15 Belt Scale Series | Tecweigh. Tecweigh. URL: https://www.tecweigh.com/products/belt-scales/wy15 (дата звернення: 27.11.2025).

10. TIA Portal Information System. TIA Portal Information System. URL: https://docs.tia.siemens.cloud/r/simatic_s7_1200_manual_collection_enus_20/introduction/maintaining-operational-safety-of-your-plant (дата звернення: 23.11.2025).

11. Introducing Ignition. Ignition User Manual. URL: https://www.docs.inductiveautomation.com/docs/8.1/getting-started/introducing-ignition (дата звернення: 28.11.2025).

12. Mezei R. A. Introduction to the Development of Web Applications Using ASP .Net (Core) MVC. Cham : Springer Nature Switzerland, 2024. URL: https://doi.org/10.1007/978-3-031-30626-6 (дата звернення: 27.11.2025).

13. UA-.NETStandard/Docs at master · OPCFoundation/UA-.NETStandard. GitHub. URL: https://github.com/OPCFoundation/UA-.NETStandard/tree/master/Docs#opc-ua-net-standard-stack-documentation (дата звернення: 28.11.2025).

14. PLCcom Opc Ua Client Sdk – Opc Ua Client for .net C# Visual Basic VB Java Developers. Indi.An. URL: https://www.indi-an.com/en/plccom/opc-ua-sdk/opcua-overview (дата звернення: 28.11.2025).

15. .NET Based OPC UA Client SDK - Unified Automation. Home - Unified Automation. URL: https://www.unified-automation.com/products/client-sdk/net-ua-client-sdk.html (дата звернення: 28.11.2025).

16. Overview of ASP.NET Core SignalR. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-10.0 (дата звернення: 28.11.2025).

17. Giretti A. Coding Clean, Reliable, and Safe REST APIs with ASP. NET Core 8: Develop Robust Minimal APIs With . NET 8. Apress L. P., 2023.

18. Cosentino N. Architecting ASP. NET Core Applications: An Atypical Design Patterns Guide for . NET 8, C# 12, and Beyond. Packt Publishing, Limited, 2024.

19. Patterns of Enterprise Application Architecture. Addison-Wesley, 2007. 533 с.

20. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Incorporated, 2020. 250 с

21. Chawla H., Kathuria H. Building Microservices Applications on Microsoft Azure: Designing, Developing, Deploying, and Monitoring. Apress, 2019. 262 с.

22. React Team. React Documentation. URL: https://uk.react.dev/ (дата звернення: 28.11.2025).

23. GitHub – softchris/react-book: Free book on React. Beginner to intermediate. *GitHub*. URL: https://github.com/softchris/react-book (дата звернення: 27.11.2025).

24. Full-Stack React Projects: Modern web development using React 16, Node, Express, and MongoDB. Packt Publishing, 2018. 470 с.

25. Free React JS Book. Free Programming Books – *GoalKicker.com*. URL: https://books.goalkicker.com/ReactJSBook/ (дата звернення: 28.11.2025).

26. React Router Official Documentation. React Router Official Documentation. URL: https://reactrouter.com/ (дата звернення: 27.11.2025).

27. Моркун Н. В., Маринич І. А. Методичні рекомендації до виконання кваліфікаційної роботи бакалавра для студентів спеціальності 122 - “Комп’ютерні науки”. Кривий Ріг, Видавничий центр ДВНЗ «КНУ». 2017.

28. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

29. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

30. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

31. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).