

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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА

“ _____ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Інтелектуальна система контролю якості даних з платформи OSDU”

Виконав: студент 4 курсу, групи ТР-24

_____ Дідук Андрій Васильович

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник ст.викладач Ольга БЕСПАЛА

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент доцент каф. ТАЕ, к.т.н., Олександр СІРИЙ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль асистент каф. ЦТЕ, Володимир РУДИК

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

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

Студент _____

Київ - 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)

Спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

_____ Наталія АУШЕВА

“__” _____ 2026 р.

З А В Д А Н Н Я
на дипломну роботу студенту

Дідуку Андрію Васильовичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Інтелектуальна система контролю якості даних з платформи OSDU”

керівник роботи Беспала Ольга Миколаївна

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

затверджені наказом по університету від “28” травня 2026 р. №1992-с

2. Термін подання роботи студентом 04.06.2026р.

3. Вихідні дані до роботи: мови програмування Python та TypeScript, фреймворки FastAPI та React, платформа OSDU, резидентна БД Redis, інструмент фонові обробки Celery, штучний інтелект Gemini, середовище розробки VS Code.

4. Перелік питань, які потрібно розробити: 1) провести аналіз серед наявних аналогів; 2) визначити головні функції веб-застосунку, що керує послугами та ресурсами; 3) аргументувати обрані інструменти, алгоритми та технології для реалізації веб-системи; 4) побудувати архітектуру програмного забезпечення та бази даних; 5) створити прикладний програмний веб-інтерфейс; 6) представити процес застосування веб-інтерфейсу.

5. Орієнтовний перелік ілюстративного матеріалу: схема, що показує архітектуру застосунку, діаграма прецедентів (ролей), блок-схема алгоритму валідації даних, знімки екрана роботи користувачів з веб-застосунком.

6. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	15.05.2025	Виконано
2.	Аналіз методів та засобів розв'язання задачі	14.04.2026-20.04.2026	Виконано
3.	Розробка архітектури та загальної структури системи	21.04.2026-27.04.2026	Виконано
4.	Розробка окремих підсистем	28.04.2026-04.05.2026	Виконано
5.	Програмна реалізація системи	05.05.2026-11.05.2026	Виконано
6.	Оформлення пояснювальної записки	12.05.2026-28.05.2026	Виконано
7.	Захист програмного забезпечення	13.05.2026	Виконано
8.	Передзахист	03.06.2026	Виконано
9.	Захист	16.06.2026	Виконано

Студент

(підпис)

Керівник

(підпис)

Андрій ДІДУК

(ім'я, ПРІЗВИЩЕ)

Ольга БЕСПАЛА

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 76 сторінках, складається з п'яти розділів, містить 34 рисунки, 1 додаток та 30 джерел у переліку посилань.

Мета роботи — розробка та впровадження інтелектуальної системи контролю якості геопросторових даних для галузевої платформи OSDU (Open Subsurface Data Universe), яка поєднує детерміновані бізнес-правила з алгоритмами штучного інтелекту для глибокого семантичного аналізу та інтерактивного виправлення даних.

Методи та засоби: мікросервісна архітектура, мова програмування Python, вебфреймворк FastAPI для побудови REST API, система розподіленої фонові обробки Celery, резидентна база даних Redis, бібліотека React з TypeScript для клієнтського застосунку, велика мовна модель Gemini API для семантичного аналізу, протокол авторизації OAuth2 на базі Microsoft Azure Active Directory, технологія контейнеризації Docker.

Основний зміст роботи: проведено аналіз предметної галузі та порівняльний огляд існуючих систем контролю якості геоданих; спроектовано клієнт-серверну архітектуру з трирівневим конвеєром валідації (синтаксичний контроль, перевірка за бізнес-правилами, семантичний аналіз III); реалізовано модуль інтеграції з API платформи OSDU та підсистему безпеки на основі RBAC; розроблено інтерактивний вебінтерфейс із картографічними модулями, аналітичними дашбордами, графом зв'язків між сутностями та інструментом автоматизованого виправлення аномалій засобами III.

Результат — готовий до розгортання програмний комплекс, що автоматизує виявлення та виправлення семантичних і синтаксичних аномалій у масивах геопросторових даних платформи OSDU з генерацією аудит-звітів у форматі PDF.

Ключові слова: OSDU, КОНТРОЛЬ ЯКОСТІ ДАНИХ, ГЕОПРОСТОРОВІ ДАНІ, ШТУЧНИЙ ІНТЕЛЕКТ, ВЕЛИКІ МОВНІ МОДЕЛІ, ВАЛІДАЦІЯ, МІКРОСЕРВІСНА АРХІТЕКТУРА, FASTAPI, REACT, REDIS.

ABSTRACT

The diploma thesis consists of 76 pages, five chapters, contains 34 illustrations, 1 appendix and 30 references.

The objective of the work is the design and implementation of an intelligent data quality control system for the OSDU (Open Subsurface Data Universe) industry platform, combining deterministic business rules with artificial intelligence algorithms for deep semantic analysis and interactive data correction.

Methods and tools: microservice architecture, Python programming language, FastAPI web framework for REST API development, Celery distributed task queue, Redis in-memory database, React with TypeScript for the client application, Gemini API large language model for semantic analysis, OAuth2 authorization protocol based on Microsoft Azure Active Directory, and Docker containerization technology.

Main content: the subject domain and existing geospatial data quality management systems were analyzed and compared; a client-server architecture with a three-stage validation pipeline (syntactic control, business rule verification, AI semantic analysis) was designed; an integration module for the OSDU platform API and an RBAC-based security subsystem were implemented; an interactive web interface was developed, featuring mapping modules, analytical dashboards, an entity relationship graph, and an AI-powered automated data remediation tool.

Result: a deployment-ready software system that automates the detection and correction of semantic and syntactic anomalies in OSDU geospatial data arrays, with PDF audit report generation.

Keywords: OSDU, DATA QUALITY CONTROL, GEOSPATIAL DATA, ARTIFICIAL INTELLIGENCE, LARGE LANGUAGE MODELS, VALIDATION, MICROSERVICE ARCHITECTURE, FASTAPI, REACT, REDIS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ.....	7
ВСТУП.....	10
1 ЗАДАЧА РОЗРОБКИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ КОНТРОЛЮ ЯКОСТІ ГЕОПРОСТОВОРИХ ДАНИХ НА ПЛАТФОРМІ OSDU.....	13
1.1 Призначення програмного забезпечення та його потенційні користувачі	13
1.2 Задачі, які розв’язуються програмним забезпеченням, та засоби розробки	14
1.3 Архітектура підсистем.....	16
1.4 Схема інформаційних потоків та взаємодії програмних модулів.....	17
2 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ІНФОРМАЦІЙНИХ СИСТЕМ КОНТРОЛЮ ЯКОСТІ ГЕОДАНИХ.....	19
2.1 Особливості предметної області та стандарти зберігання інформації на платформі OSDU	19
2.2 Огляд літератури щодо методів валідації та інтелектуального аналізу даних..	21
2.3 Порівняльний аналіз існуючих систем управління якістю даних	23
2.4 Обґрунтування необхідності розробки запропонованої системи	25
3 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	27
3.1 Вибір мов програмування та базових вебфреймворків.....	27
3.2 Вибір системи управління базами даних та механізмів кешування	28
3.3 Технології фоновної обробки та інтеграції зі штучним інтелектом	29
3.4 Засоби контейнеризації та інфраструктурного розгортання	30
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА АРХІТЕКТУРИ СИСТЕМИ.....	31
4.1 Моделювання функціональних вимог та діаграма прецедентів	31
4.2 Реалізація підсистеми управління доступом та безпекою	33
4.3 Архітектура бази даних та NoSQL-моделювання у Redis	35
4.4 Розробка API-шлюзу та маршрутизації запитів.....	37
4.5 Інтеграція з мікросервісами платформи OSDU	40

	6
4.6 Організація асинхронної фонові обробки даних	41
4.7 Програмна реалізація гібридного конвеєра валідації.....	43
4.8 Архітектура клієнтського застосунку та управління безпекою	45
4.9 Реалізація функціональних модулів користувацького інтерфейсу.....	48
5 ІНСТРУКЦІЯ КОРИСТУВАЧА ТА ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМНОЇ СИСТЕМИ.....	50
5.1 Вимоги до апаратного та програмного забезпечення	50
5.2 Процес інсталяції та розгортання системи	51
5.3 Авторизація та навігація системою	52
5.4 Аналітичний дашборд та загальний огляд стану системи	54
5.5 Навігація бізнес-одинацями та геопросторова візуалізація	55
5.6 Візуалізація та фільтрація геофізичних даних	57
5.7 Візуалізація об'єктів видобутку: Карта свердловин.....	58
5.8 Робота з каталогом даних	59
5.9 Управління бізнес-правилами.....	61
5.10 Запуск та контроль конвеєра валідації.....	62
5.11 Менеджер звітів та експорт результатів	64
5.12 Детальний аналіз записів та інтелектуальне виправлення	67
5.13 Візуалізація зв'язків OSDU	69
5.14 Інтерфейс профілю користувача.....	71
5.15 Управління користувачами та ролями	72
ВИСНОВКИ	75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	77
ДОДАТОК А	80

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

ACL (Access Control List) — список контролю доступу, що визначає права читання та запису для конкретного об'єкта або ресурсу.

API (Application Programming Interface) — прикладний програмний інтерфейс, набір визначених правил і протоколів для взаємодії між програмними компонентами.

Azure AD / Microsoft Entra ID (Azure Active Directory) — хмарна служба управління ідентифікацією та доступом від Microsoft, яка використовується для корпоративної автентифікації та авторизації.

BU (Business Unit) — бізнес-одиниця, організаційно-географічний підрозділ компанії, у розрізі якого структуруються та аналізуються геопросторові дані.

CRUD (Create, Read, Update, Delete) — базовий набір операцій над даними: створення, читання, оновлення та видалення.

DOM (Document Object Model) — об'єктна модель документа, програмний інтерфейс для динамічної взаємодії зі структурою HTML-сторінки.

DQ / DQR (Data Quality Rule) — правило контролю якості даних, що визначає критерії перевірки коректності конкретного поля або групи полів об'єкта.

GeoJSON — відкритий формат для кодування різноманітних географічних структур даних на основі JSON.

HOC (Higher-Order Component) — компонент вищого порядку у React, що обгортає інший компонент для розширення його функціональності.

HTTP / HTTPS (HyperText Transfer Protocol / Secure) — протокол передачі гіпертексту та його захищена версія, що використовуються для обміну даними у веб.

I/O (Input/Output) — введення-виведення, операції обміну даними між програмою та зовнішніми ресурсами (мережею, файловою системою тощо).

JSON (JavaScript Object Notation) — текстовий формат обміну даними, що базується на підмножині синтаксису мови JavaScript і широко використовується для передачі структурованої інформації.

LLM (Large Language Model) — велика мовна модель, нейронна мережа, навчена на великих масивах текстових даних для розуміння та генерації природної мови.

MSAL (Microsoft Authentication Library) — бібліотека автентифікації Microsoft, що реалізує протокол OAuth2 для отримання токенів доступу від Azure AD у клієнтських застосунках.

NLP (Natural Language Processing) — обробка природної мови, розділ штучного інтелекту, що займається взаємодією комп'ютерів із людською мовою.

NoSQL (Not Only SQL) — клас систем управління базами даних, що не використовують виключно реляційну модель і підходять для зберігання неструктурованих або слабоструктурованих даних.

OAuth2 (Open Authorization 2.0) — відкритий протокол авторизації, що дозволяє застосунку отримати обмежений доступ до облікового запису користувача без передачі пароля.

OSDU (Open Subsurface Data Universe) — галузева хмарно-орієнтована платформа для централізованого зберігання, управління та обміну геопросторовою інформацією у нафтогазовій промисловості.

PDF (Portable Document Format) — формат файлів для представлення документів незалежно від програмного забезпечення, апаратного забезпечення та операційної системи.

PPDM (Professional Petroleum Data Management) — міжнародна асоціація та стандарт управління даними у нафтогазовій галузі, що регламентує моделі даних і правила їх іменування.

RBAC (Role-Based Access Control) — управління доступом на основі ролей, модель безпеки, за якої права користувача визначаються його роллю в системі.

Redis (Remote Dictionary Server) — резидентна база даних типу «ключ-значення», що зберігає дані в оперативній пам'яті для забезпечення надвисокої швидкості доступу.

REST (Representational State Transfer) — архітектурний стиль проектування розподілених гіпермедійних систем, що використовується для побудови вебслужб та API.

SPA (Single Page Application) — односторінковий застосунок, вебдодаток, який динамічно оновлює вміст сторінки без її повного перезавантаження.

SQL (Structured Query Language) — мова структурованих запитів, стандартна мова для управління реляційними базами даних.

TTL (Time-To-Live) — час життя, параметр, що визначає тривалість зберігання об'єкта в кеші або мережевого пакету до його автоматичного видалення.

URL (Uniform Resource Locator) — уніфікований локатор ресурсів, стандартизована адреса для визначення місцезнаходження ресурсу в мережі.

WSL2 (Windows Subsystem for Linux 2) — підсистема Windows для Linux другого покоління, що забезпечує запуск середовища Linux безпосередньо на Windows.

ШІ — штучний інтелект, сукупність технологій та алгоритмів, що дозволяють комп'ютерним системам виконувати завдання, які зазвичай потребують людського інтелекту.

ВСТУП

У сучасній нафтогазовій галузі ефективне управління геолого-геофізичною інформацією є критичним чинником для успішного пошуку, розвідки та розробки родовищ вуглеводнів. Галузевий стандарт OSDU (Open Subsurface Data Universe) став глобальним орієнтиром для побудови хмарних сховищ, що дозволяють об'єднати розрізнені дані свердловин та сейсморозвідки в єдину екосистему. Проте геонаукові дані є надзвичайно гетерогенними, мають глибоко вкладену структуру та змінну схему JSON-документів. Основна проблема полягає в тому, що класичні інструменти контролю якості перевіряють лише синтаксичну відповідність даних JSON-схемам OSDU, але повністю пропускають семантичні та логічні аномалії — такі як хронологічні суперечності дат буріння, просторові невідповідності координат ліцензійним блокам або нереалістичні одиниці вимірювання фізичних величин.

Традиційно ці проблеми вирішуються або тривалим і дорогим ручним аудитом, або впровадженням жорстких детермінованих бізнес-правил, які не здатні адаптуватися до змінних контекстів чи аналізувати природно-мовні описи. Вони досі не вирішені через складність інтелектуальної обробки гнучких JSON-документів та відсутність гнучких інструментів, що поєднують швидкість роботи з розумінням змісту даних. Розробка інтелектуальної системи допомагає вирішити цю проблему завдяки впровадженню двохетапного конвеєра: миттєвого пошуку математичних помилок за допомогою NoSQL-правил у Redis та глибокого логіко-контекстного аналізу за допомогою великих мовних моделей штучного інтелекту.

У цьому контексті актуальним є створення веб-застосунку, який забезпечує автоматизований інтелектуальний контроль якості геонаукових даних на платформі OSDU, поєднуючи швидкі бізнес-правила із семантичною перевіркою за допомогою штучного інтелекту, та надає інтерактивні інструменти геопросторової візуалізації та аудиту зв'язків.

Метою даної роботи є розробка та впровадження інтелектуальної програмної системи контролю якості геонаукових даних на базі платформи OSDU для

автоматичного виявлення, візуалізації та усунення синтаксичних і семантичних аномалій у корпоративних інформаційних сховищах.

Для досягнення поставленої мети були сформовані наступні завдання:

- 1) проаналізувати наявні рішення та сервіси для управління якістю даних у нафтогазовій галузі та специфіку стандартів OSDU;
- 2) обрати технології для створення високопродуктивної асинхронної мікросервісної архітектури, NoSQL-моделювання та модулів інтелектуального аналізу;
- 3) створити серверну частину додатку, яка забезпечує виконання двохетапного конвеєра валідації, асинхронну чергу задач, розмежування доступу на основі ролей та інтеграцію з великими мовними моделями через API;
- 4) розробити структуру та дизайн користувацького інтерфейсу для картографічної візуалізації просторових об'єктів, інтерактивного аналізу зв'язків між сутностями у вигляді графа, управління бізнес-правилами та генерації аналітичних звітів.

Дана дипломна робота складається з 5 розділів. Загальна кількість рисунків — 34, найменувань у списку використаних джерел — 30, основна частина викладена на 74 сторінках, повний обсяг роботи — 87 сторінок.

У першому розділі проведено аналіз предметної області, досліджено специфіку та домени даних хмарного стандарту OSDU, сформульовано задачу розробки інтелектуальної системи контролю якості геонаукових даних з детальним описом вхідної/вихідної інформації та ролей користувачів.

У другому розділі здійснено аналіз наукових методів валідації, проведено порівняльний аналіз існуючих комерційних платформ та open-source інструментів управління якістю даних, виділено їхні недоліки та обґрунтовано необхідність створення гібридного рішення.

У третьому розділі наведено обґрунтування вибору засобів розробки, мов програмування, асинхронного мікрофреймворку FastAPI, системи керування чергами Celery, NoSQL бази даних Redis, середовища контейнеризації Docker та сервісів Azure Active Directory і Gemini AI.

У четвертому розділі детально описано програмну реалізацію та архітектуру системи, NoSQL-моделювання об'єктів у Redis, розробку API-шлюзу, побудову конвеєра інтелектуальної валідації та реалізацію підсистеми управління доступом.

У п'ятому розділі представлено процес розгортання та інсталяції системи за допомогою Docker Compose, детально описано користувацький інтерфейс, картографічні модулі візуалізації, панелі керування правилами, граф зв'язків та результати функціонування системи у вигляді аналітичних звітів.

1 ЗАДАЧА РОЗРОБКИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ КОНТРОЛЮ ЯКОСТІ ГЕОПРОСТОВОРИХ ДАНИХ НА ПЛАТФОРМІ OSDU

Задачею даного дослідження є розробка програмного комплексу для автоматизованого контролю якості геоданих на базі платформи OSDU. Основна проблема, яку вирішує система, полягає у відсутності надійних механізмів виявлення прихованих семантичних аномалій та логічних суперечностей у масивах складних об'єктів (таких як свердловини, каротажі чи сейсмічні зйомки). Для розв'язання цієї проблеми необхідно створити систему, яка інтегрується з OSDU API, здійснює багаторівневу валідацію записів за допомогою користувацьких бізнес-правил та алгоритмів штучного інтелекту, а також надає зручний інтерфейс.

1.1 Призначення програмного забезпечення та його потенційні користувачі

Основне призначення програмного забезпечення полягає в комплексній автоматизації процесів контролю якості геоданих (насамперед таких сутностей як свердловини, стовбури свердловин, сейсмічних зйомків тощо), які зберігаються в індустріальному середовищі OSDU [1]. Розроблювана система покликана виявляти як формальні синтаксичні порушення, так і приховані семантичні суперечності в масивах даних за допомогою алгоритмів штучного інтелекту. Крім того, програмний продукт забезпечує візуалізацію результатів перевірки, побудову інтерактивних графів зв'язків та формування конкретних рекомендацій щодо виправлення знайдених помилок із можливістю їх застосування через інтерфейс.

Потенційними користувачами розроблюваної системи є фахівці нафтогазової галузі, геонауковці та інженери з управління даними. Їх можна розділити на три основні цільові групи з відповідними ролями:

— кінцеві користувачі. Геологи, геофізики та бізнес-аналітики, які використовують систему переважно в режимі читання для перегляду аналітичних дашбордів, графів залежностей та результатів перевірки якості даних перед тим, як використовувати ці дані у своїх дослідженнях;

— експерти предметної області та власники даних. Спеціалісти, які несуть відповідальність за достовірність інформації. Вони мають права на створення та редагування правил перевірки, ініціювання масових перевірок якості, а також прийняття або відхилення пропозицій штучного інтелекту щодо виправлення аномалій у даних;

— адміністратори даних. Користувачі з максимальними привілеями, які, окрім безпосереднього управління даними, здійснюють адміністрування профілів користувачів, розподіл ролей та загальний контроль за функціонуванням інформаційного середовища.

1.2 Задачі, які розв’язуються програмним забезпеченням, та засоби розробки

Для досягнення загальної мети програмне забезпечення автоматизує виконання низки функціональних задач, кожна з яких характеризується специфічними потоками вхідної та вихідної інформації (рис. 1.1). Першою базовою задачею є синхронізація та високошвидкісне кешування об’єктів OSDU [2]. Вхідною інформацією для неї виступають запити користувача на пошук певних типів даних, параметри пагінації, фільтрації та облікові токени авторизації. На виході система генерує структуровані JSON-об’єкти, які надійно зберігаються у швидкісній базі даних [3]. Це дозволяє суттєво знизити навантаження на хмарну платформу та забезпечити миттєвий доступ до інформації під час подальшого аналізу. Наступною важливою задачею є формальна валідація даних на основі настроюваних бізнес-правил. На вхід цієї підсистеми подаються масиви даних із локального сховища та активні набори правил якості, що описують допустимі числові діапазони, типи даних, обов’язковість полів і

хронологічну логіку. Вихідною інформацією слугують статуси проходження перевірки для кожного запису окремо. [4–5].

Третьою, більш складною задачею є інтелектуальний семантичний аналіз та генерація рекомендацій. Вхідними даними тут виступають записи, що потребують експертної перевірки, та контекстні запити, згенеровані системою. Результатом виконання цієї задачі є детальні текстові описи семантичних аномалій, виявлених моделлю ШІ, й структуровані рекомендації для виправлення значень (додаток А) [6]. Завершальною задачею є візуалізація зв'язків та результатів валідації. Вона приймає на вхід масиви із вкладеними посиланнями та статистику звітів, формуючи графічний інтерфейс із графами залежностей між сутностями та аналітичними таблицями.

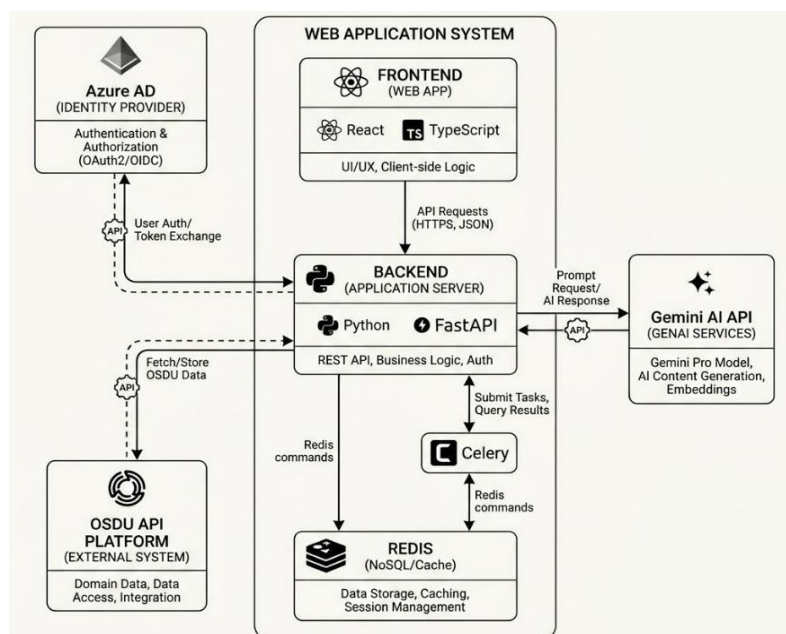


Рисунок 1.1 — Діаграма засобів розробки

Для реалізації цієї комплексної архітектури було обрано сучасний стек засобів розробки. Серверна частина створена мовою програмування Python з використанням фреймворку FastAPI для побудови високопродуктивного REST API [7]. Фонова обробка тривалих процесів забезпечується бібліотекою Celery у зв'язці з брокером повідомлень [8]. Зберігання та швидкий доступ до даних організовано за допомогою резидентної бази даних Redis. Клієнтську частину побудовано на базі бібліотеки React

та мови TypeScript для створення надійного односторінкового застосунку, а для візуалізації статистичної аналітики інтегровано бібліотеку Recharts [9–11]. Функції штучного інтелекту реалізовані через безпосередню взаємодію з сучасними хмарними LLM-моделями Google (Gemini). Загальна інфраструктура розгорнута з використанням Docker для контейнеризації та забезпечення платформонезалежності, тоді як управління доступом здійснюється через механізми Azure AD [12–13].

1.3 Архітектура підсистем

Програмне забезпечення побудовано на мікросервісних принципах і складається з кількох взаємопов'язаних підсистем, кожна з яких має свою визначену мету. Підсистема веб-інтерфейсу призначена для забезпечення користувача зручним візуальним інструментом для взаємодії з даними. Вона приймає дії користувача, відправляє відповідні запити до серверної частини та перетворює отримані відповіді у зрозумілі графічні елементи. Центральним ядром системи виступає підсистема API-шлюзу та маршрутизації. Її головна мета полягає у прийомі запитів від клієнтської частини, здійсненні жорсткої перевірки прав доступу та розподілі завдань. Задачі, що потребують значних обчислювальних ресурсів або тривалого часу на виконання, делегуються підсистемі фонові обробки [14].

Мета підсистеми фонові обробки полягає у забезпеченні асинхронного виконання процесів без блокування основного інтерфейсу користувача. Вона відповідає за пакетне завантаження масивів з OSDU та їх збереження у локальному кеші, паралельно розподіляючи завдання між чергами з різним пріоритетом. Безпосередня перевірка даних покладена на підсистему інтелектуального аналізу та валідації. Вона зіставляє поля об'єктів з бізнес-правилами, а також формує звернення до мовних моделей штучного інтелекту, передаючи їм контекст запису. Після отримання відповіді від алгоритмів, модуль адаптує її у внутрішній формат, перетворюючи на звіти про проблеми та готові пропозиції для їх усунення.

1.4 Схема інформаційних потоків та взаємодії програмних модулів

Функціонування комплексу забезпечується послідовною взаємодією його складових модулів, що утворюють єдиний інформаційний конвеєр. Процес розпочинається з того, що клієнтський модуль генерує запит на аналіз набору даних та надсилає його до модулю API-шлюзу. Отримавши запит, шлюз здійснює валідацію токенів доступу через зовнішній сервіс авторизації та визначає рівень привілеїв користувача.

Якщо ініційована дія передбачає масову перевірку або масштабну синхронізацію з хмарним середовищем, API-шлюз не блокує роботу системи в очікуванні завершення процесу, а натомість публікує відповідне завдання у брокері повідомлень.

Далі підсистема фонові обробки перехоплює це завдання, звертається до зовнішніх пошукових та файлових сервісів платформи OSDU, витягує необхідні масиви записів і зберігає їх у локальному швидкісному сховищі. Після наповнення локальної бази активується модуль валідації. Він вичитує збережені правила перевірки та самі записи, автоматично зіставляє їх і формує первинний звіт про синтаксичну цілісність даних.

Для тих записів, що потребують глибшого експертного аналізу, ініціюється обмін контекстною інформацією між модулем інтеграції та зовнішніми мовними моделями, в результаті якого повертаються текстово-структуровані результати експертизи.

На фінальному етапі всі згенеровані дані, включно зі статусами проходження перевірок, побудованими зв'язками графів та сформованими пропозиціями щодо виправлення помилок, агрегуються в єдиний інформаційний пакет (рис 1.2). Застосування такого багаторівневого конвеєра обробки дозволяє не лише забезпечити автоматизований контроль якості великих обсягів даних, а й інтегрувати інтелектуальні компоненти для підтримки прийняття рішень.

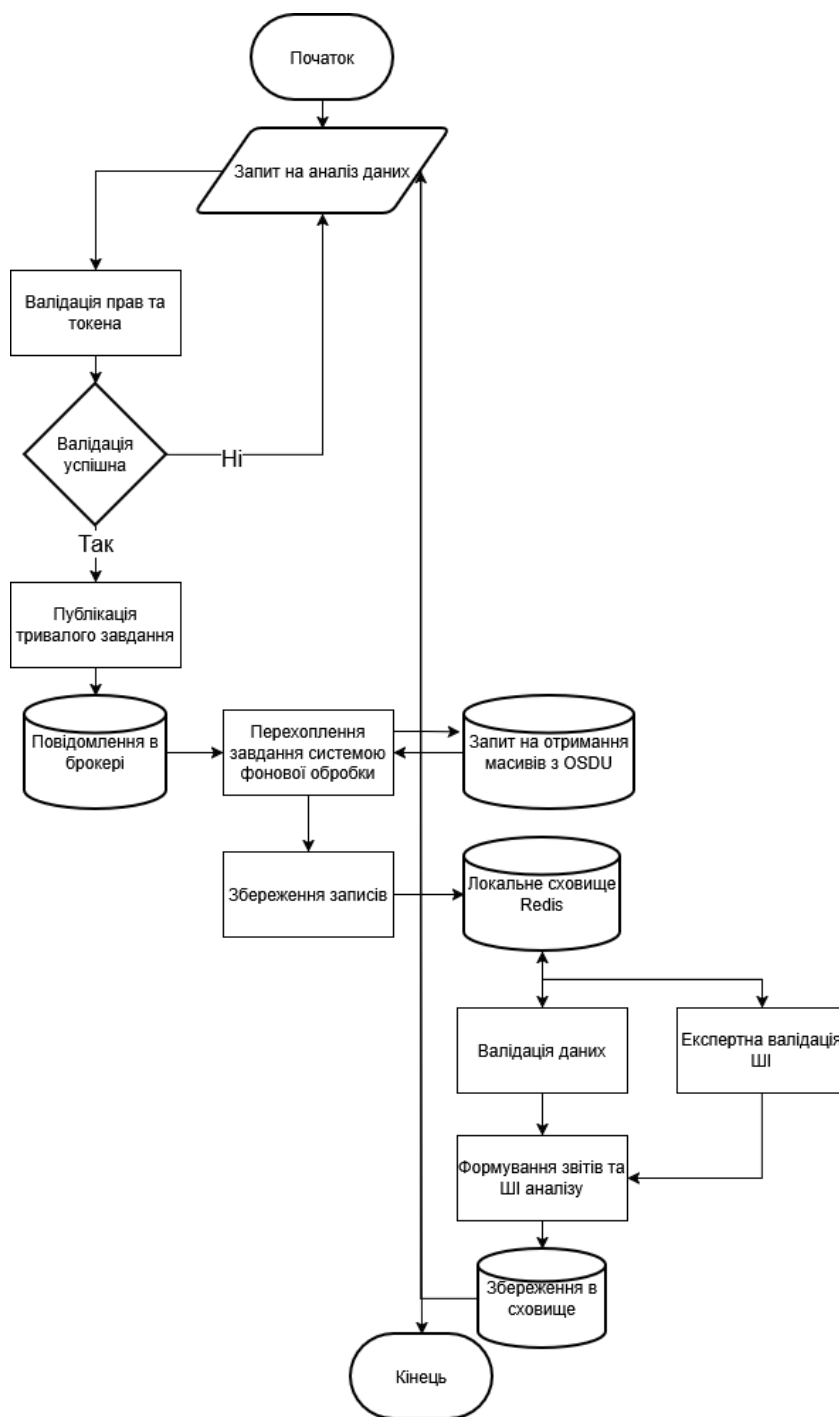


Рисунок 1.2 — Блок-схема алгоритму взаємодії модулів

Цей пакет фіксується у кеші та повертається через API-шлюз до веб-інтерфейсу, де інформація остаточно візуалізується для кінцевого користувача.

2 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ІНФОРМАЦІЙНИХ СИСТЕМ КОНТРОЛЮ ЯКОСТІ ГЕОДАНИХ

Ефективне управління геопросторовими даними є фундаментом для прийняття обґрунтованих стратегічних рішень у нафтогазовій та енергетичній галузях. Тож варто провести детальний аналіз предметної області, розглянути сучасні стандарти зберігання інформації та дослідити науково-практичну літературу щодо існуючих підходів до контролю якості даних. Крім того, здійснити порівняльний аналіз провідних інформаційних систем, що дозволяє виявити їхні недоліки та теоретично обґрунтувати необхідність розробки нової, більш досконалої інтелектуальної системи валідації.

2.1 Особливості предметної області та стандарти зберігання інформації на платформі OSDU

Сучасна нафтогазова галузь оперує безпрецедентними обсягами інформації, яка безупинно генерується на всіх етапах розвідки, розробки та експлуатації родовищ. Ця інформація охоплює різноманітні геофізичні, геологічні та інженерні дані, такі як координати стовбурів свердловин, результати каротажів, об'ємні сейсмічні зйомки та лабораторні аналізи зразків гірських порід. Історично склалося так, що компанії зберігали ці масиви у розрізнених, пропрієтарних форматах та ізольованих базах даних. Такий підхід створював суттєві перешкоди для обміну інформацією між департаментами, надзвичайно ускладнював агрегацію даних для комплексного аналізу та часто призводило до дублювання або навіть втрати критично важливих відомостей під час передачі проєктів між підрядниками.

Для вирішення проблеми інформаційної розрізненості провідні світові енергетичні компанії та технологічні гіганти об'єдналися для створення платформи OSDU (Open Subsurface Data Universe). OSDU являє собою стандартизовану, хмарно-

орієнтовану екосистему, призначену для централізованого зберігання, управління та безпечного обміну геопросторовою інформацією. Платформа встановлює єдині галузеві стандарти для моделей даних і надає набір відкритих прикладних програмних інтерфейсів (API) для пошуку, зчитування та запису об'єктів [15]. Завдяки такому підходу різноманітні програмні додатки можуть безперешкодно взаємодіяти з єдиним корпоративним джерелом істини, незалежно від того, яким саме постачальником вони були розроблені.

Архітектура даних платформи OSDU базується на строго типізованих JSON-схемах та логічному ієрархічному поділі сутностей. Уся інформація концептуально поділяється на кілька ключових категорій (рис. 2.1).

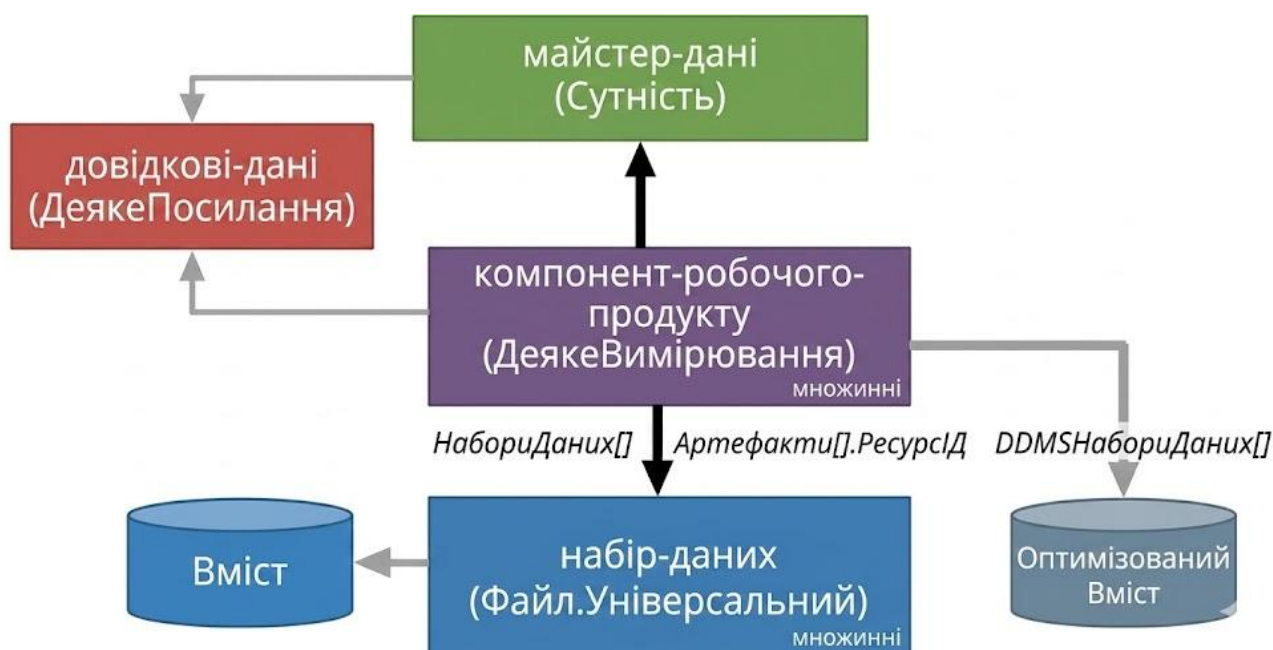


Рисунок 2.1 — Діаграма зв'язків масивів даних платформи OSDU

Основні дані (Master Data) описують фундаментальні бізнес-об'єкти, такі як свердловини, стовбури свердловин або масштабні сейсмічні проекти. Компоненти робочого продукту (Work Product Components) містять більш специфічну, часто аналітичну або просторову інформацію, наприклад, набори маркерів, інтерпретації вимірювань пластового тиску або сейсмічні горизонти. Окремою категорією

виступають довідкові дані (Reference Data), що слугують загальними корпоративними словниками для стандартизації термінології. Кожен запис у системі супроводжується розширеними метаданими, які описують його походження, актуальну версію, права доступу та зв'язки з іншими об'єктами. Саме ці складні багатовимірні зв'язки дозволяють відстежувати залежності, наприклад, визначати чітку приналежність конкретного масиву геофізичних досліджень до певного стовбура свердловини.

Незважаючи на потужні можливості щодо агрегації та бездоганної структурної стандартизації, платформа OSDU за своєю суттю залишається інфраструктурним сховищем, а не інтелектуальним аналізатором. Хоча в її архітектурі передбачено формальні схеми для фіксації результатів перевірки якості, вбудовані механізми платформи обмежуються лише базовим синтаксичним контролем під час завантаження файлів, таким як перевірка наявності обов'язкових полів чи загальна відповідність типів даних. Платформа не здійснює автоматичної глибокої перевірки бізнес-логіки та не здатна самостійно виявляти приховані семантичні суперечності чи пропонувати шляхи їх ефективного виправлення. Цей архітектурний факт перекладає всю відповідальність за кінцеву якість даних на плечі самих користувачів та зовнішніх застосунків. Таким чином, в екосистемі виникає потреба у розробці спеціалізованих надбудов — інтелектуальних інформаційних систем контролю, які здатні безперервно аналізувати дані, зіставляти їх з жорсткими галузевими правилами та залучати алгоритми машинного навчання для забезпечення бездоганної достовірності інформації.

2.2 Огляд літератури щодо методів валідації та інтелектуального аналізу даних

Проблема контролю якості інформації в базах даних має тривалу історію досліджень, еволюціонуючи від простих синтаксичних перевірок до складних інтелектуальних систем. Традиційні підходи до валідації, що широко описуються у науковій літературі, базуються переважно на експертних системах та детермінованих

бізнес-правилах. У нафтогазовій галузі фундаментальну роль відіграють стандарти таких асоціацій як PPDM (Professional Petroleum Data Management) [16], які регламентують допустимі діапазони значень, правила іменування та базові логічні залежності між об'єктами. Системи, побудовані на основі таких жорстких правил, демонструють високу ефективність у виявленні тривіальних помилок, наприклад, від'ємних значень вимірюної глибини, відсутності обов'язкових ідентифікаторів або невідповідності форматів дат [17]. Проте вчені та практики зазначають, що детерміновані підходи мають суттєві обмеження. Вони є негнучкими, вимагають значних ресурсів експертів для підтримки бази правил у актуальному стані та, що найголовніше, не здатні виявляти приховані семантичні аномалії. Такі аномалії можуть формально задовольняти всі числові обмеження, але залишатися позбавленими фізичного чи геологічного сенсу в ширшому контексті [18–19].

З огляду на фундаментальні обмеження традиційних систем, сучасна наукова література все більше фокусується на застосуванні методів машинного навчання та штучного інтелекту для аналізу геопросторових даних. Значна кількість публікацій присвячена використанню алгоритмів пошуку аномалій без вчителя, таких як ізоляційний ліс (Isolation Forest), метод опорних векторів для одного класу (One-Class SVM) та різноманітні архітектури нейромережових автокодувальників (Autoencoders). Ці алгоритми дозволяють автоматизовано виявляти нетипові патерни в багатовимірних масивах даних, які можуть свідчити про помилки вимірювань або некоректну інтерпретацію спеціалістом. Однак, незважаючи на високу точність виявлення відхилень, класичні статистичні моделі машинного навчання страждають від проблеми інтерпретованості («чорна скринька») [20–21]. Вони лише вказують на наявність аномалії, але вкрай рідко здатні пояснити її природу людською мовою або запропонувати конкретний шлях виправлення, що суттєво ускладнює їх інтеграцію в автоматизовані індустріальні конвеєри управління якістю.

Останнім часом у науковій спільноті спостерігається стрімке зростання інтересу до застосування великих мовних моделей (LLM) та методів обробки природної мови (NLP) у задачах валідації складних структурованих і неструктурованих даних. Дослідники підкреслюють, що сучасні генеративні моделі

володіють безпрецедентними можливостями щодо розуміння контексту та здатні ефективно аналізувати складні ієрархічні JSON-схеми, виявляючи тонкі логічні невідповідності між різними полями одного чи кількох пов'язаних об'єктів. На відміну від жорстких алгоритмів, штучний інтелект здатний інтерпретувати специфічні терміни, зіставляти їх із загальновизнаними знаннями предметної області та генерувати змістовні текстові пояснення виявлених суперечностей. Крім того, передові дослідження вказують на великі перспективи використання мовних моделей не лише для виявлення помилок, але й для генерації структурованих пропозицій щодо їх усунення (Data Remediation), що дозволяє перетворити системи валідації з пасивних спостерігачів на активних помічників інженера.

Аналіз сучасної літератури дозволяє зробити висновок, що найбільш ефективним підходом до управління якістю складних корпоративних даних є побудова гібридних систем. Такі комплекси повинні гармонійно поєднувати високу швидкість та абсолютну надійність перевірок за детермінованими бізнес-правилами з глибиною та адаптивністю семантичного аналізу на базі штучного інтелекту. Саме така гібридна парадигма лежить в основі розроблюваної в рамках даного дослідження системи, що дозволить подолати наявний розрив між теоретичними концепціями інтелектуальної валідації та їх практичним впровадженням у реальне хмарне середовище.

2.3 Порівняльний аналіз існуючих систем управління якістю даних

Для обґрунтування технічних та функціональних вимог до розроблюваної системи було проведено порівняльний аналіз існуючих рішень на ринку. Зважаючи на специфіку зберігання геопросторових даних у хмарних середовищах, для аналізу було обрано три різні класи інформаційних систем: нативні базові сервіси екосистеми OSDU, спеціалізовані комерційні платформи від великих галузевих вендорів та універсальні відкриті фреймворки контролю якості.

Першою системою для розгляду є нативний сервіс OSDU Data Quality Services, який постачається як базова частина архітектури самої платформи. Його беззаперечною перевагою є глибока інтеграція з екосистемою OSDU, використання повністю стандартизованих типів даних та висока масштабованість у хмарному середовищі. Платформа дозволяє зберігати результати перевірки безпосередньо у сховищі як компоненти робочого продукту. Однак цей сервіс має критичні недоліки. Він забезпечує лише найпростіший синтаксичний контроль, перевіряючи відповідність JSON-документів затвердженим структурам. У системі відсутній потужний вбудований рушій для обробки складної бізнес-логіки, немає повноцінного графічного інтерфейсу для управління правилами, а також повністю відсутні механізми семантичного аналізу. Крім того, базовий сервіс лише пасивно маркує наявність помилки, не пропонуючи користувачеві жодних шляхів чи рекомендацій щодо її виправлення.

Другим класом рішень є комерційні платформи управління даними від провідних галузевих вендорів, яскравими представниками яких є потужні екосистеми на кшталт Schlumberger Delfi або Halliburton Landmark [22]. Перевагою таких систем є глибока галузева експертиза розробників. Вони містять тисячі попередньо налаштованих бізнес-правил, які повністю відповідають стандартам PPDM, пропонують багаті інструменти для візуалізації результатів у вигляді інтерактивних аналітичних дашбордів та забезпечують професійну технічну підтримку. До суттєвих недоліків комерційних рішень слід віднести їхню надзвичайно високу вартість, закритий сирцевий код та ризик жорсткої прив'язки до одного постачальника (vendor lock-in). Незважаючи на потужність таких експертних систем, вони здебільшого покладаються на детерміновані алгоритми, що обмежує їх гнучкість у виявленні нестандартних семантичних аномалій. До того ж, механізми виправлення даних у таких системах зазвичай зводяться до ручного редагування таблиць користувачем, без використання генеративного штучного інтелекту для формування персоналізованих розумних підказок.

Третім підходом є використання універсальних відкритих фреймворків валідації даних, найпопулярнішим серед яких є система Great Expectations. Головною

перевагою цього інструменту є надзвичайна гнучкість, наявність великої спільноти розробників та можливість безкоштовного використання. Фреймворк дозволяє описувати декларативні очікування щодо структури та значень даних за допомогою коду на мові Python, генеруючи при цьому зрозумілу документацію за результатами перевірок. Проте використання таких систем у контексті OSDU супроводжується значними труднощами. Вони розроблені переважно для табличних даних (реляційних баз або фреймів Pandas), тоді як OSDU оперує складними, глибоко вкладеними графами JSON-об'єктів. Відповідно, використання універсальних фреймворків вимагає написання великої кількості проміжного коду для витягування та трансформації даних (парсингу). Крім того, Great Expectations не володіє функціоналом для автоматичної генерації виправлень і не містить модулів штучного інтелекту для семантичного контролю, залишаючись суто детермінованим інструментом тестування.

2.4 Обґрунтування необхідності розробки запропонованої системи

Проведений огляд літератури та детальний порівняльний аналіз існуючих систем чітко демонструє наявність суттєвої прогалини на ринку програмного забезпечення для контролю якості геоданих. Існуючі рішення або обмежуються примітивними формальними перевітками, не здатними виявити складні логічні помилки, або є закритими комерційними монолітами, які позбавлені інтелектуальної гнучкості та сучасної підтримки великих мовних моделей. Нативні засоби екосистеми OSDU, з їхнього боку, забезпечують лише інфраструктурний базис, залишаючи відкритим питання повноцінної експертизи об'єктів. Жодна з розглянутих систем не пропонує надійного гібридного підходу, у якому швидкість перевірки жорстких бізнес-правил поєднувалася б із контекстним розумінням даних алгоритмами штучного інтелекту.

Усвідомлення цих обмежень формує нагальну необхідність розробки нової, спеціалізованої інформаційної системи. Запропоноване програмне забезпечення має

не лише автоматизувати інтеграцію з хмарним API OSDU для безшовного витягування масивів даних, але й забезпечити трирівневий конвеєр валідації. Ключовою інновацією розроблюваної системи є імплементація модулів штучного інтелекту, які виступають не лише аналізаторами прихованих семантичних дефектів, але й активними асистентами, здатними автоматично генерувати структуровані рекомендації для швидкого виправлення помилок користувачами в інтерактивному інтерфейсі. Реалізація такого архітектурного задуму дозволить принципово підвищити достовірність геопросторових досліджень, зменшити час на рутинну роботу інженерів та максимізувати ефективність використання корпоративної платформи даних [19].

3 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Створення сучасної інформаційної системи для обробки великих масивів геоданих вимагає комплексного підходу до проєктування архітектури та ретельного підбору технологічного стеку. Розроблюваний програмний комплекс функціонує як розподілена багаторівнева система. Вона повинна безперервно взаємодіяти з хмарним середовищем OSDU, виконувати ресурсомісткі задачі валідації записів у фоновому режимі та забезпечувати інтерактивну взаємодію з кінцевим користувачем без затримок.

Відповідно, обрані інструменти розробки повинні відповідати суворим критеріям: гарантувати високу продуктивність обчислень, забезпечувати надійність асинхронної обробки завдань, підтримувати високі стандарти безпеки при передачі корпоративних даних та дозволяти безшовну інтеграцію із зовнішніми хмарними сервісами штучного інтелекту.

3.1 Вибір мов програмування та базових вебфреймворків

Основною мовою програмування для розробки серверної частини системи (Backend) було обрано Python [23]. Такий вибір зумовлений його беззаперечним лідерством у сфері обробки даних, широкими можливостями інтеграції з алгоритмами машинного навчання та наявністю потужної екосистеми бібліотек для роботи з JSON-структурами. Оскільки геодані платформи OSDU вимагають складної маніпуляції з багатовимірними ієрархічними об'єктами, синтаксичні можливості Python дозволили реалізувати логіку парсингу та валідації максимально лаконічно. У якості серверного фреймворку було інтегровано FastAPI. На відміну від традиційних громіздких рішень, FastAPI базується на сучасних стандартах асинхронного програмування та забезпечує високу пропускну здатність API-шлюзу. Використання

цього інструменту дозволило системі одночасно приймати десятки клієнтських запитів на аналіз інформації та здійснювати миттєву маршрутизацію без блокування основного потоку виконання.

Для реалізації клієнтського графічного інтерфейсу (Frontend) було обрано бібліотеку React у поєднанні зі строго типізованою мовою TypeScript. Оскільки розроблювана система повинна візуалізувати складні взаємозв'язки між записами, відображати аналітичні дашборди та надавати інтерактивний інтерфейс для швидкого редагування виявлених аномалій, компонентний підхід React став оптимальним рішенням для побудови динамічного односторінкового застосунку. Застосування TypeScript відіграло критичну роль у забезпеченні надійності клієнтського коду. Завдяки статичній типізації контракти обміну даними між сервером та інтерфейсом жорстко контролюються на етапі компіляції, що дозволило уникнути критичних помилок під час відображення вкладених структур даних OSDU.

3.2 Вибір системи управління базами даних та механізмів кешування

Специфіка взаємодії з корпоративною платформою OSDU вимагає постійного завантаження величезних масивів геопросторових метаданих через мережу, що зазвичай супроводжується значними часовими затримками на рівні інфраструктури. Для вирішення проблеми продуктивності та забезпечення миттєвої швидкодії аналітичних алгоритмів було прийнято рішення інтегрувати систему управління базами даних резидентного класу, а саме Redis. У даному архітектурному рішенні Redis виконує роль не просто допоміжного транзитного кешу, а виступає ключовим локальним сховищем високої доступності. Після того, як система витягує тисячі записів з хмарного сховища OSDU, вони серіалізуються та розміщуються в оперативній пам'яті сервера бази даних. Це дозволяє модулю валідації отримувати доступ до необхідних об'єктів за долі мілісекунди. Крім того, завдяки використанню спеціалізованого розширення для об'єктного моделювання, Redis дозволив

ефективно зберігати та швидко вилучати налаштовані користувачами правила контролю якості та згенеровані масиви звітів про перевірку.

3.3 Технології фонові обробки та інтеграції зі штучним інтелектом

Процеси масової синхронізації записів з OSDU та проведення складного багаторівневого семантичного аналізу потребують значних обчислювальних ресурсів і можуть тривати від кількох хвилин до годин. Їх виконання у синхронному режимі неминуче призвело б до блокування роботи користувача.

Для організації асинхронної фонові обробки було застосовано розподілену систему черг завдань Celery. Завдяки цій технології ресурсомісткі задачі відкріплюються від основного веб-сервера і делегуються спеціальним процесам-робітникам. Це забезпечило високу масштабованість архітектури, дозволивши розпаралелити перевірку якості тисяч геопросторових об'єктів на кілька незалежних обчислювальних потоків із різними рівнями пріоритету.

Окремим архітектурним викликом стала реалізація інтелектуальної валідації, що виходить за межі детермінованих бізнес-правил. Для вирішення цієї задачі було здійснено програмну інтеграцію з хмарним інтерфейсом великих мовних моделей, зокрема Gemini API. Замість того, щоб розробляти та навчати власні нейромережі з нуля, система експлуатує вже існуючі потужності передового штучного інтелекту.

Модуль інтеграції формує спеціалізовані контекстні запити, що містять фрагменти видобутих даних та специфіку предметної області, після чого направляє їх до мовної моделі.

Отримана відповідь автоматично розбирається сервером і перетворюється на структуровані рекомендації щодо виправлення значень, роблячи LLM ключовим аналітичним рушієм системи.

3.4 Засоби контейнеризації та інфраструктурного розгортання

Для забезпечення абсолютної стабільності, ізоляції та платформонезалежності роботи програмного комплексу було використано технологію контейнеризації Docker. Враховуючи, що архітектура проєкту складається з великої кількості мікрокомпонентів, а саме API-сервера, кількох фонових робітників, брокера повідомлень та резидентної бази даних - налаштування такого середовища вручну було б надзвичайно трудомістким і загрожувало б конфліктами версій. Завдяки інструментарію Docker Compose, уся серверна інфраструктура розгортається автоматизовано в межах ізольованих контейнерів, що гарантує ідентичність функціонування системи як на локальному комп'ютері розробника, так і на кінцевому продуктовому хмарному сервері.

Враховуючи високу комерційну чутливість геопросторової інформації, особлива увага при проєктуванні була приділена механізмам безпеки. Оскільки платформа OSDU тісно інтегрована з сучасними корпоративними стандартами ідентифікації, для управління доступом користувачів до розробленого застосунку було імплементовано протоколи авторизації на базі Microsoft Azure Active Directory (Azure AD) [24]. Ця інфраструктурна складова дозволила не лише реалізувати безпечний безпарольний вхід за допомогою єдиних корпоративних облікових записів, але й забезпечила генерацію токенів доступу. Використовуючи ці токени, серверна частина програмного комплексу може легітимно та безпечно взаємодіяти з API платформи OSDU від імені поточного користувача, суворо дотримуючись його індивідуальних привілеїв та корпоративних політик.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА АРХІТЕКТУРИ СИСТЕМИ

Практична реалізація цього дослідження базується на створенні гнучкої архітектури інтелектуальної системи. Відштовхуючись від теоретичних напрацювань та обраного технологічного стеку, було послідовно спроектовано програмні механізми, які гарантують безперебійність усіх етапів: від інтеграції та витягування даних до їх глибокої верифікації та автоматичного виправлення виявлених невідповідностей.

Робота над системою охопила повний цикл проєктування: від високорівневого моделювання прецедентів використання до низькорівневої реалізації NoSQL-структур. Ключовим елементом рішення став трирівневий конвеєр валідації, доповнений модулем штучного інтелекту, що генерує динамічні виправлення даних. Паралельно було розроблено інтерактивний інтерфейс, призначений для візуалізації багатовимірних графів зв'язків між сутностями платформи OSDU. У результаті, комплексне поєднання діаграм, оптимізованих структур даних та програмних компонентів демонструє зрілість розробленого рішення та його цілковиту готовність до впровадження у реальне індустріальне середовище.

4.1 Моделювання функціональних вимог та діаграма прецедентів

Процес розробки складної інформаційної системи вимагає чіткого моделювання функціональних вимог та визначення меж взаємодії між системою та її користувачами. Для формалізації цих взаємодій було застосовано методологію об'єктно-орієнтованого проєктування. В архітектурі розробленої системи передбачено три основні ролі користувачів, права яких суворо розмежовані на рівні серверної частини за допомогою модуля управління доступом (рис. 4.1) [25].

Базовою роллю є «Кінцевий користувач». Ця категорія охоплює геологів, аналітиків та інших фахівців, чия робота полягає переважно у споживанні готової інформації. Кінцевий користувач має права виключно на читання: він може переглядати загальну статистику на дашбордах, здійснювати навігацію каталогом даних, будувати інтерактивні графи зв'язків між об'єктами, а також переглядати звіти та експортувати у PDF-формат результати вже виконаних системних перевірок якості.

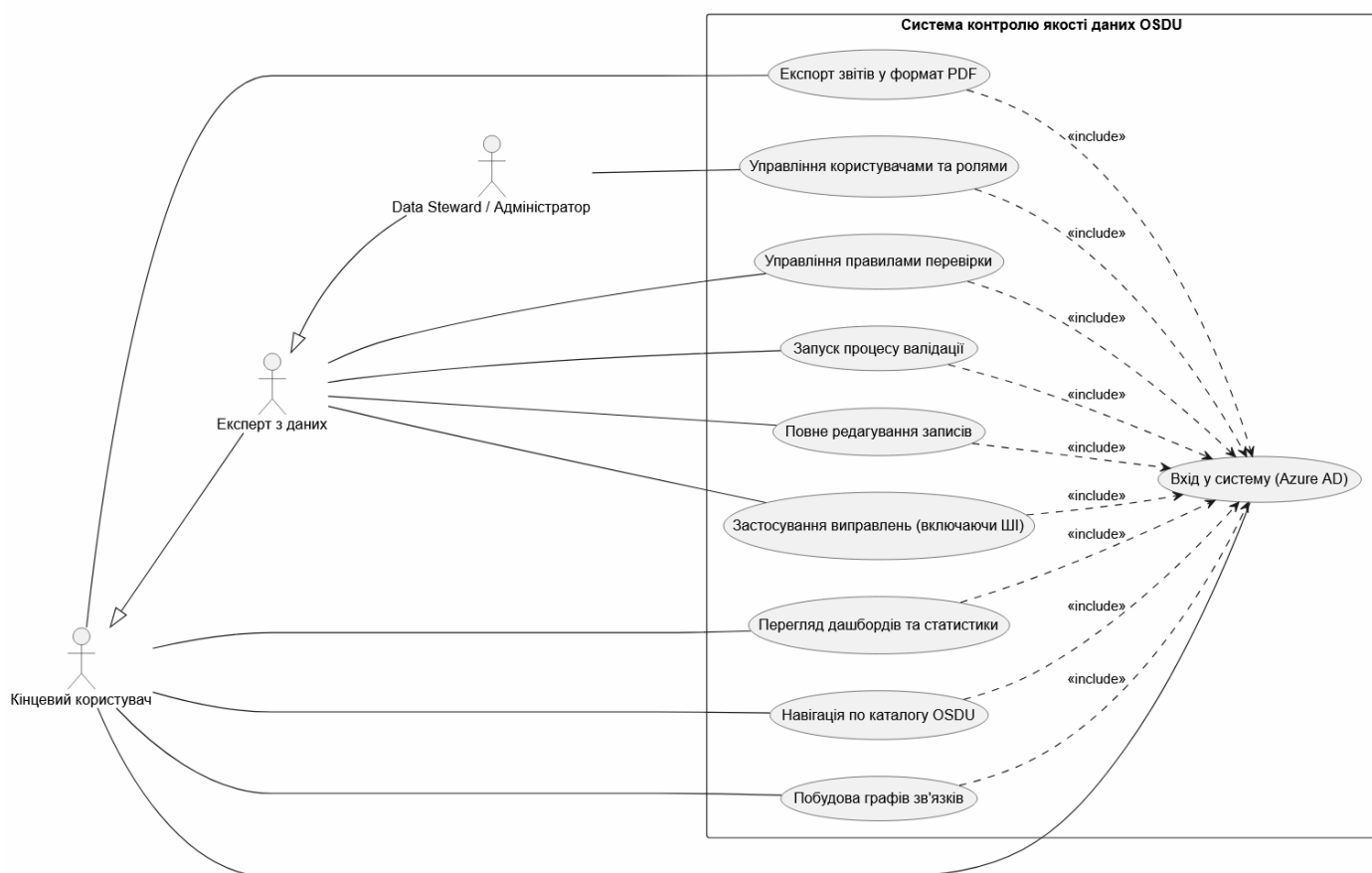


Рисунок 4.1 — Діаграма прецедентів

Наступним рівнем привілеїв наділений «Експерт з даних» (Data Custodian / Data Owner). Окрім повного базового функціоналу, цей актор несе відповідальність за безпосереднє управління життєвим циклом даних. Експерт має повноваження ініціювати нові фонові процеси валідації масивів інформації, а також створювати, редагувати або повністю деактивувати користувацькі правила перевірки (Data Quality

Rules). Найважливішою функцією експерта є взаємодія з модулем штучного інтелекту: він отримує згенеровані системою семантичні звіти, аналізує запропоновані мовною моделлю рекомендації щодо виправлення аномалій та приймає фінальне рішення про їх застосування або відхилення.

Найвищий рівень доступу в ієрархії системи має «Адміністратор даних» (Data Steward). Ця роль повністю успадковує всі права експерта, але додатково наділяється виключними адміністративними повноваженнями. Адміністратор відповідає за загальну цілісність середовища і має доступ до окремого модуля управління профілями, де він може самостійно призначати, змінювати або відкликати ролі доступу для інших користувачів корпоративної системи.

4.2 Реалізація підсистеми управління доступом та безпекою

Забезпечення безпеки та суворого розмежування прав доступу є критично важливою вимогою для будь-якої інформаційної системи, що взаємодіє з корпоративною геопросторовою інформацією. Оскільки екосистема OSDU тісно інтегрується з хмарними провайдерами ідентифікації (зокрема, Microsoft Azure Active Directory), розроблений програмний комплекс свідомо не використовує власну ізольовану систему збереження паролів. Натомість автентифікація користувачів відбувається шляхом перехоплення, розшифрування та криптографічної валідації делегованих токенів доступу стандарту OAuth2 [24]. Для забезпечення гнучкого гранулярного контролю безпосередньо на рівні API-маршрутів серверної частини, було спроектовано та програмно реалізовано спеціалізований модуль безпеки, що отримав назву Role Guard.

Архітектура цього модуля базується на індустріальній концепції управління доступом на основі ролей (Role-Based Access Control, RBAC). У програмному коді чітко задекларовано базові корпоративні ролі, які повністю відповідають архітектурній номенклатурі платформи OSDU.

З метою уникнення дублювання коду та спрощення подальшого масштабування, базові ролі логічно агрегуються у спеціалізовані набори дозволів (Permission Sets) за допомогою заморожених множин.

Наприклад, набір привілеїв `ROLES_CAN_EVALUATE` динамічно об'єднує користувачів, які мають право ініціювати ресурсомісткі фонові процеси валідації даних. Набір `ROLES_CAN_WRITE_RULES` жорстко контролює доступ до створення та модифікації користувацьких правил якості, унеможливаючи несанкціоноване втручання в бізнес-логіку системи рядовими користувачами.

Абсолютний рівень привілеїв, визначений у наборі `ROLES_CAN_WRITE_RECORDS`, забезпечує право безпосередньо модифікувати та перезаписувати оригінальні записи OSDU (зокрема, застосовувати згенеровані штучним інтелектом виправлення) і доступний виключно користувачам керівного та адміністративного рівнів.

Критично важливою інновацією програмної реалізації модуля Role Guard є наявність вбудованого механізму локального кешування прав доступу. Оскільки архітектура сучасних REST API-сервісів не зберігає стан між запитами (stateless), постійне мережеве звернення до зовнішнього сервісу OSDU Entitlements API для перевірки ролей під час кожного клієнтського кліку призвело б до неприпустимого падіння загальної продуктивності.

Для усунення цього архітектурного вузького місця було розроблено оперативний словник, який ізольовано функціонує в оперативній пам'яті кожного активного процесу-воркера. Після першої успішної ідентифікації користувача сервер запитує його ролі та тимчасово фіксує їх у пам'яті із заданим часом життя (TTL), що в даній системі становить п'ять хвилин.

Такий підхід забезпечує компроміс між продуктивністю та безпекою: з одного боку, кеш гарантує мілісекундну перевірку привілеїв при активній роботі в інтерфейсі, з іншого — у разі відкликання корпоративних прав адміністратором, локальна сесія користувача буде гарантовано та автоматично інвалідована після завершення короткого п'ятихвилинного життєвого циклу запису.

4.3 Архітектура бази даних та NoSQL-моделювання у Redis

Класичні реляційні бази даних (наприклад, PostgreSQL або MySQL) традиційно використовуються для зберігання суворо структурованої табличної інформації. Однак, з огляду на специфіку геоданих платформи OSDU, які представляють собою складні, глибоко вкладені JSON-документи зі змінною схемою (з гнучкими блоками), використання реляційного підходу вимагало б надзвичайно складної та ресурсомісткої нормалізації. Саме тому для реалізації внутрішнього сховища розроблюваної системи було обрано NoSQL резидентну базу даних Redis. Такий архітектурний вибір дозволив забезпечити максимальну гнучкість структур даних та критично важливу для аналітичної системи швидкість доступу.

У межах реалізованої архітектури сховище Redis виконує дві принципово різні, але однаково важливі функції: виступає буферним кешем для пакетів інформації з хмари та слугує постійною базою даних для бізнес-об'єктів самої системи контролю якості (рис. 4.2).

Перший сценарій використання реалізований за допомогою паттерну ключ-значення (Key-Value), де результати масового витягування записів з OSDU Storage API серіалізуються і зберігаються під унікальними ідентифікаторами (наприклад, `osdu_record:{record_id}`). Завдяки цьому підходу, модуль валідації та алгоритми штучного інтелекту отримують блискавичний доступ до «гарячих» даних без необхідності генерувати тисячі повільних HTTP-запитів до зовнішніх сервісів. Така структура кешу ефективно вирішує проблему вузьких місць при обробці багатотисячних масивів стовбурів свердловин чи сейсмічних маркерів.

Другий сценарій використання Redis полягає у повноцінному зберіганні внутрішніх системних об'єктів. Для забезпечення надійної взаємодії об'єктно-орієнтованого коду на Python із NoSQL сховищем було використано спеціалізовану бібліотеку `aredis-om` (Object Mapping). Це дозволило спроектувати строгі моделі даних (клас `JsonModel`), зокрема основну сутність — `DataQualityRule`. Ця модель інкапсулює в собі всі необхідні метадані правила контролю якості: унікальний ідентифікатор, назву, опис, прив'язку до конкретного типу даних OSDU

(наприклад, master-data--Wellbore) та безпосередньо саму бізнес-логіку або поріг семантичної довіри для штучного інтелекту.

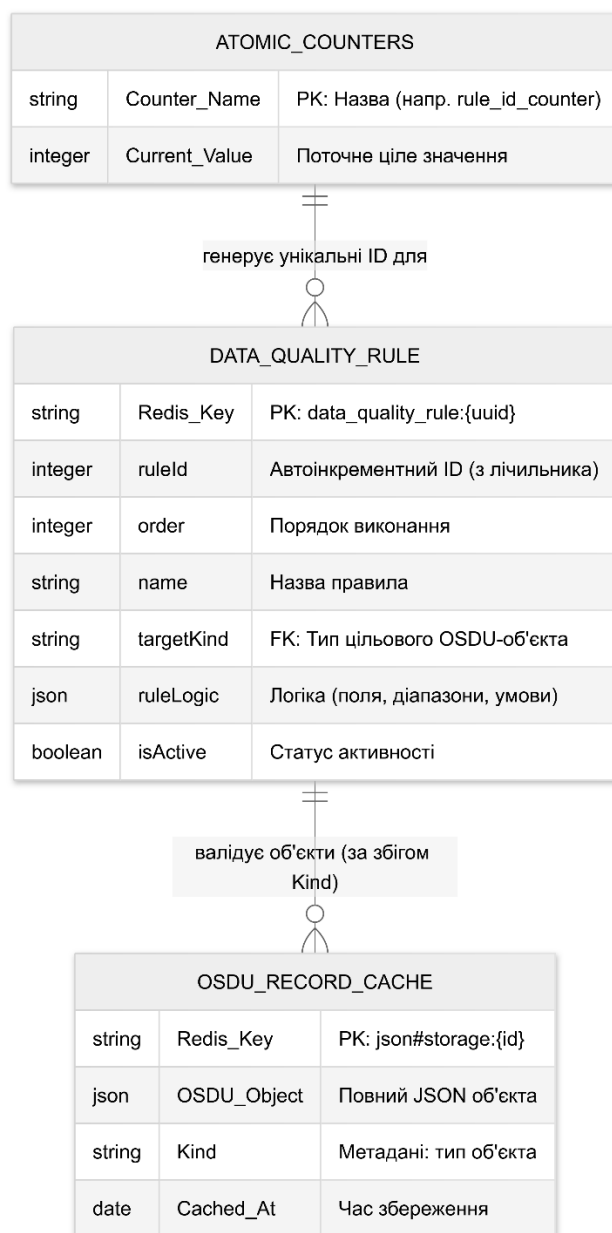


Рисунок 4.2 — Моделі даних в NoSQL базі даних

Цікавою особливістю розробленої моделі даних є реалізація механізмів гарантування унікальності та послідовності. Оскільки Redis за своєю природою не має вбудованих автоінкрементних полів у стилі традиційних SQL-баз, для сутності DataQualityRule було програмно реалізовано використання спеціалізованих

атомарних лічильників (counters). При створенні нового правила система звертається до лічильників, отримуючи унікальне значення, що гарантує цілісність даних та коректність їх подальшого сортування у веб-інтерфейсі. Такий гібридний підхід, поєднання надшвидкого JSON-кешування «сирих» даних OSDU з об'єктно-модельованим зберіганням внутрішніх правил та звітів, формує гнучку, високопродуктивну інформаційну основу для всієї архітектури.

4.4 Розробка API-шлюзу та маршрутизації запитів

Серцем серверної частини розробленого програмного комплексу виступає API-шлюз, побудований на базі високопродуктивного мікрофреймворку FastAPI. Цей архітектурний компонент виконує функції єдиної централізованої точки входу для всіх клієнтських запитів, забезпечуючи їх прийом, попередню валідацію, перевірку прав доступу та маршрутизацію до відповідних внутрішніх підсистем. Головною технічною перевагою застосування FastAPI є його повна нативна орієнтація на сучасні стандарти асинхронного програмування в Python (async/await). Оскільки система інтенсивно взаємодіє із зовнішніми хмарними сервісами (API платформи OSDU) та мережевими базами даних (Redis), абсолютна більшість її серверних операцій є I/O-залежними (bound to Input/Output). Завдяки асинхронному дизайну сервер не блокує головний операційний потік під час очікування відповіді від хмарного сховища, а натомість миттєво перемикається на паралельну обробку інших запитів. Цей механізм гарантує надзвичайно високу пропускну здатність системи та мінімізує затримки навіть під час пікових навантажень з боку багатьох користувачів й мінімізує ресурси.

З метою забезпечення високої модульності, полегшення тестування та зручності подальшої підтримки програмного коду, уся складна бізнес-логіка API-шлюзу була декомпозирована на серію незалежних маршрутизаторів (routers), кожен з яких відповідає за свою специфічну предметну область. Архітектура маршрутизації серверної частини включає наступні ключові модулі:

— `/utils` єдиний публічний маршрутизатор, який не вимагає перевірки токенів авторизації. Використовується інфраструктурою для перевірки стану здоров'я сервера (`health-checks`) та виконання базових системних функцій;

— `/browse` модуль для безпосередньої взаємодії з екосистемою OSDU. Обробляє пошукові запити до платформи, забезпечує пагінацію результатів та завантажує оригінальні JSON-схеми типів даних;

— `/bu` відповідає за роботу з конфігураціями бізнес-одиниць, дозволяючи динамічно адаптувати інтерфейс та набори даних під різні географічні регіони;

— `/rules` інкапсулює життєвий цикл користувацьких правил перевірки якості (CRUD-операції). Забезпечує створення нових конфігурацій валідації, їх модифікацію та вилучення з локальної бази Redis;

— `/report` модуль управління результатами перевірок. Надає клієнтському застосунку агреговані статистичні звіти з деталізацією кожної виявленої синтаксичної чи семантичної аномалії;

— `/explorer` спеціалізований маршрутизатор для екрану Data Explorer, який забезпечує швидке витягування масивів даних для їх зручного табличного відображення та фільтрації користувачем;

— `/map` відповідає за агрегацію та передачу просторових метаданих, що дозволяє клієнтській частині візуалізувати точні координати стовбурів свердловин на інтерактивній веб-карті;

— `/user` модуль управління профілями. Використовується для налаштування персональних даних та розподілу рівнів привілеїв між користувачами адміністраторами системи (через підсистему Role Guard);

— `/ai` маршрутизатор інтелектуальної обробки, який ізолює логіку взаємодії зі штучним інтелектом (Gemini API), обробляючи запити на генерацію семантичних підказок та розумних виправлень даних;

— `/graph` складний алгоритмічний модуль, у межах якого реалізовано рекурсивний обхід вкладених масивів зв'язків OSDU-об'єктів для побудови візуального дерева залежностей (Record Relationship Graph).

Усі системні маршрути стандартизовано з використанням префіксу версіонування (наприклад, /api/v1/), що гарантує збереження зворотної сумісності (рис. 4.3).

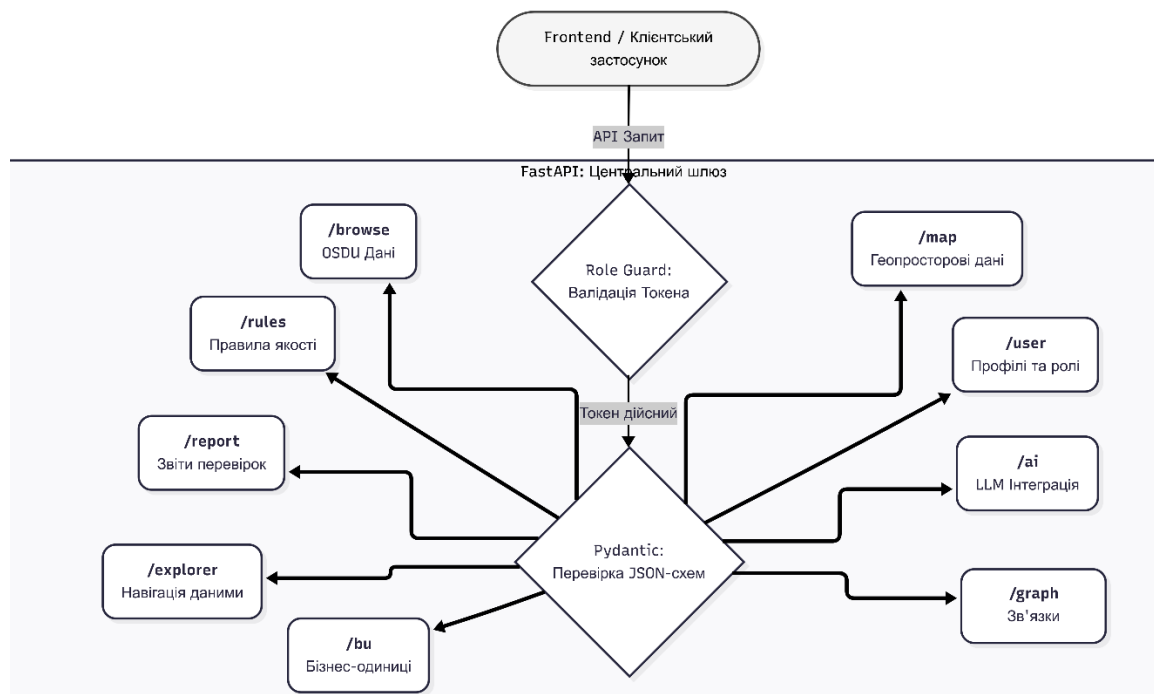


Рисунок 4.3 — Дерево модулів застосунку

Усі вищезазначені модулі (окрім утиліт) захищені залежністю авторизації. Тобто неможливо виконати запит на сервер не маючи дійсного токена, оскільки він перевіряється на сервері. Також, користувачу може буде відмовлено у запиті при умові що він не має достатніх прав, права користувача записуються на рівні OSDU й кешуються на стороні клієнта. Крім того, завдяки глибокій нативній інтеграції FastAPI з бібліотекою об'єктної валідації Pydantic [26], API-шлюз автоматично перевіряє вхідні тіла запитів до будь-якого з цих маршрутів. У випадку передачі некоректних параметрів, система автоматично відхиляє транзакцію (повертаючи код HTTP 422 Unprocessable Entity) ще до того етапу, коли хибні дані могли б викликати збій у бізнес-логіці програми й заподіяти шкоди.

4.5 Інтеграція з мікросервісами платформи OSDU

Оскільки галузева платформа OSDU функціонує як масштабна хмарна екосистема, що складається з десятків ізольованих мікросервісів, надійна та оптимізована взаємодія з ними стала одним із найважливіших архітектурних завдань при створенні програмного комплексу. Для абстрагування бізнес-логіки системи від складності низькорівневих мережевих транзакцій було спроектовано спеціалізований фасадний клас-обгортку (OSDUApi). Цей програмний інтерфейс (Data Interface) повністю інкапсулює логіку формування HTTP-запитів, обробки специфічних хмарних помилок та управління мережевими таймаутами.

Ключовою архітектурною особливістю розробленого інтеграційного модуля є використання підходу «ледачої ініціалізації» асинхронних HTTPX-клієнтів для кожного окремого мікросервісу платформи. Замість підтримки постійних та ресурсомістких з'єднань з усіма вузлами OSDU, система динамічно встановлює канали зв'язку виключно з тими сервісами, які необхідні для виконання конкретного завдання. В архітектурі розробки реалізовано інтеграцію з трьома базовими сервісами екосистеми. Спочатку модуль взаємодіє з сервісом пошуку (Search API) для виконання складних Elasticsearch-запитів та отримання масивів унікальних ідентифікаторів об'єктів на основі заданих користувачем фільтрів. Далі, використовуючи ці ідентифікатори, система звертається до сервісу зберігання (Storage API) для витягування «важких» JSON-документів із самими геоданими. Паралельно імплементовано інтеграцію з сервісом метаданих (Schema API), що дозволяє застосунку динамічно завантажувати актуальні корпоративні JSON-схеми об'єктів для проведення попереднього синтаксичного контролю їхньої структури.

З метою радикальної оптимізації мережевого трафіку та мінімізації навантаження на інфраструктуру OSDU, у модулі інтеграції програмно реалізовано алгоритм пакетного (batch) витягування даних. Замість традиційного, але вкрай неефективного циклічного відправлення індивідуальних GET-запитів для кожного запису, розроблений клас формує агреговані POST-запити на спеціалізовані ендпоінти Storage API. Такий підхід дозволяє за одне мережеве звернення витягувати

масиви з сотень записів, що критично знижує інфраструктурну затримку під час масової синхронізації бази даних.

Оскільки всі запити до OSDU вимагають криптографічної перевірки [1], безпека взаємодії з хмарою забезпечується окремим класом-одинаком (Singleton) — `OAuthTokenRefresherSingleton`. Цей компонент бере на себе завдання безперервного автоматичного оновлення службових токенів доступу (через механізм `Azure AD Client Credentials flow`). Його робота гарантує, що багатохвилинні сесії зв'язку з хмарними сервісами під час виконання глибокої валідації масивів даних не будуть раптово розірвані через закінчення стандартного терміну дії авторизаційних сертифікатів.

4.6 Організація асинхронної фонові обробки даних

Операції з масового завантаження інформації з хмарної інфраструктури OSDU, а також процеси багаторівневої валідації тисяч записів із залученням мовних моделей штучного інтелекту є обчислювально важкими та надзвичайно ресурсомісткими задачами. Виконання таких завдань у синхронному режимі, безпосередньо в асинхронному циклі подій API-сервера, неминуче призвело б до критичного зниження пропускної здатності та тривалого блокування (timeout) веб-інтерфейсу для користувачів. Для фундаментального вирішення цієї проблеми архітектура програмного комплексу була доповнена спеціалізованою підсистемою розподіленої фонові обробки на базі фреймворку Celery.

Впровадження Celery дозволило архітектурно відокремити важку бізнес-логіку обчислень від швидкісних маршрутизаторів FastAPI. Коли експерт ініціює перевірку якості або синхронізацію масиву даних, API-сервер лише формує стандартизоване повідомлення-завдання та миттєво відправляє його у брокер повідомлень (Message Broker, роль якого виконує резидентна база Redis). Після цього сервер негайно повертає клієнту підтвердження про постановку задачі в чергу, залишаючись вільним для обробки нових HTTP-запитів. Автономні вузли-робітники (Celery Workers)

перехоплюють ці завдання з брокера і виконують їх в ізольованих системних процесах. З метою оптимізації управління обчислювальними ресурсами, у програмному комплексі було спроектовано дві черги завдань: default (стандартна) та high_priority (високопріоритетна). Стандартний вузол-робітник призначений для безперервної обробки масштабних рутинних завдань з фонові синхронізації сховищ. Натомість високопріоритетна черга оптимізована за допомогою механізмів динамічного автоматичного масштабування, що дозволяє їй автоматично розширюватися до чотирьох паралельних процесів. Цей механізм гарантує, що система миттєво реагуватиме на критично важливі дії користувача (наприклад, інтерактивний запит до штучного інтелекту щодо виправлення конкретного запису) навіть під час максимального завантаження інфраструктури рутинними перевітками (рис. 4.4).

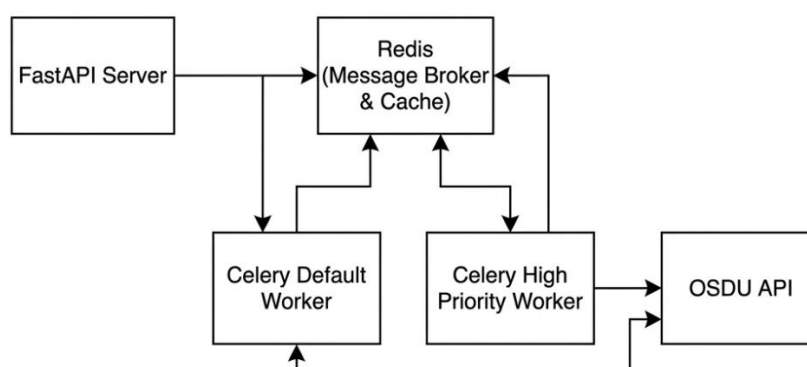


Рисунок 4.4 – Діаграма організації фонові обробки даних

Одним із найбільш технологічно складних алгоритмів фонові підсистеми є реалізований механізм «прогрівання кешу». Оскільки хмарні API OSDU можуть обробляти масові агреговані запити з певними затримками, пряме звернення до них при кожному завантаженні каталогу даних на клієнті створювало б дискомфорт під час навігації. Процес прогрівання превентивно усуває цю проблему. Під час запуску системи або за явним запитом адміністратора, Celery Manager генерує завдання верхнього рівня, яке звертається до сервісу пошуку OSDU та отримує масив усіх доступних ідентифікаторів об'єктів заданого типу (наприклад, усі стовбури

свердловин у системі). Далі застосовується алгоритмічний патерн «розкидання завдань»: головний процес рекурсивно створює десятки дочірніх підзадач. Кожна підзадача звертається до сервісу зберігання, витягує повні JSON-документи пакетами по 100 одиниць та паралельно серіалізує їх у базу Redis.

У результаті роботи цього конвеєра формується постійно актуалізована, «гаряча» локальна копія необхідних корпоративних даних. Це інженерне рішення забезпечує можливість миттєвого завантаження багатотисячних списків складних об'єктів в інтерфейсі клієнтського застосунку Data Explorer.

4.7 Програмна реалізація гібридного конвеєра валідації

Найважливішою функціональною підсистемою розробленого програмного комплексу є модуль валідації. Головною метою цього модуля є автоматизоване виявлення хибних даних, локалізація аномалій та формування рекомендацій щодо їх усунення. Ключова архітектурна інновація даного рішення полягає у відмові від виключно детермінованих алгоритмів на користь гібридного підходу. Цей підхід гармонійно поєднує надшвидке виконання традиційних бізнес-правил із глибиною семантичного аналізу, притаманною генеративному штучному інтелекту. Технічно процес перевірки реалізовано у вигляді послідовного трирівневого інформаційного конвеєра, через який програмно проходить кожен геопросторовий запис (рис. 4.5).

На першому рівні (нульовий етап) відбувається базовий синтаксичний контроль. Модуль валідації звертається до офіційного сервісу Schema API платформи OSDU, динамічно завантажує корпоративну JSON-схему для поточного типу об'єкта та перевіряє його загальну структурну цілісність. На цьому етапі алгоритм гарантує наявність обов'язкових системних ідентифікаторів, блоків контролю доступу (ACL) та сувору відповідність базових типів даних заданим стандартам. Другий рівень конвеєра забезпечує детерміновану перевірку за користувацькими бізнес-правилами. На цьому кроці система завантажує з кешу Redis усі активні екземпляри моделі DataQualityRule, призначені для аналізованого типу об'єкта. Використовуючи

рефлексію та методи рекурсивного обходу дерева JSON-документів, модуль динамічно зіставляє значення конкретних полів із заданими експертом діапазонами.

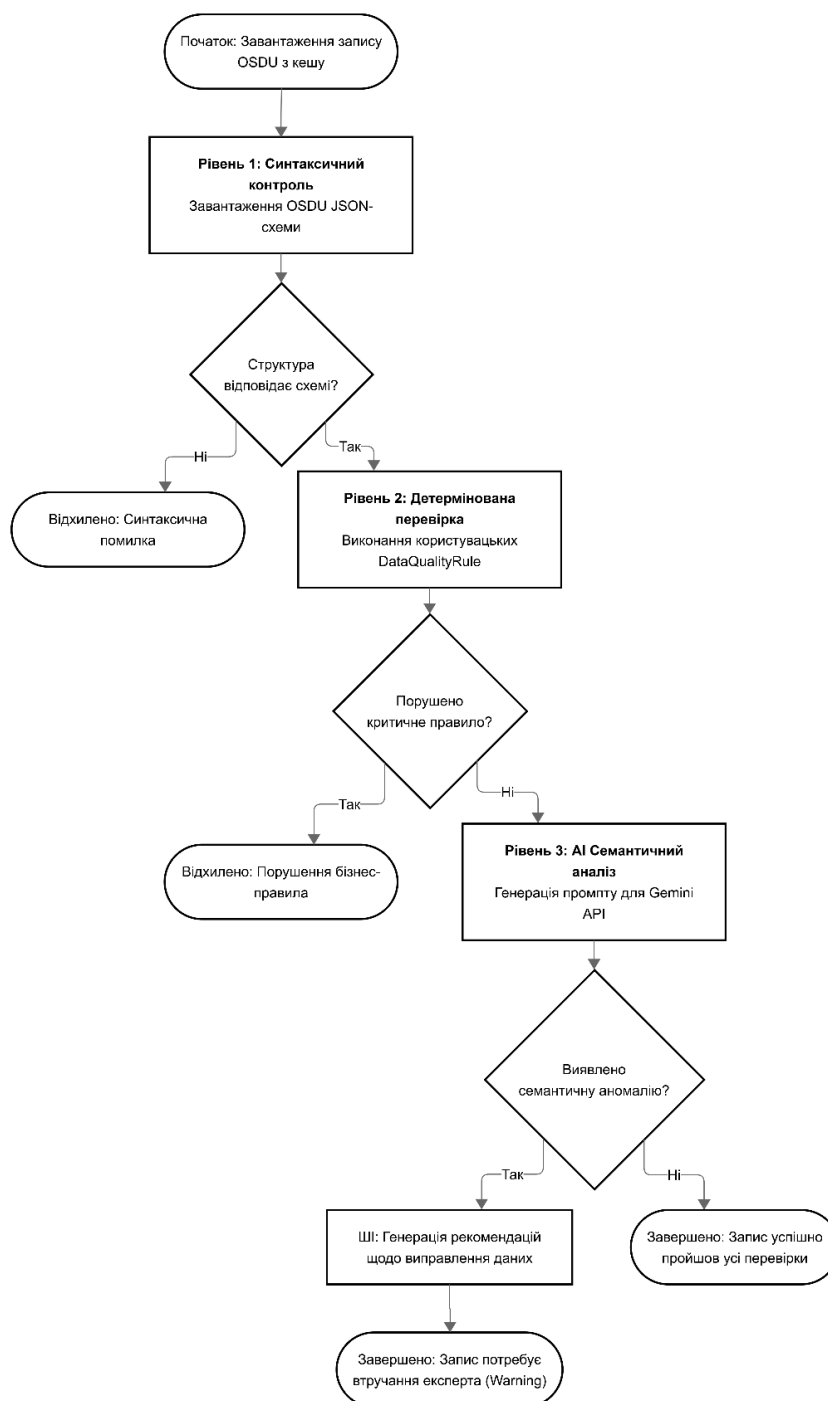


Рисунок 4.5 — Блок-схема алгоритму валідації даних

Наприклад, алгоритм перевіряє, чи знаходиться значення пористості породи строго в межах від 0 до 1, та чи не є виміряна глибина від’ємним числом. Якщо запис

порушує хоча б одне критичне бізнес-правило, йому миттєво присвоюється статус системної помилки, і він відхиляється від подальших перевірок, що дозволяє суттєво зекономити обчислювальні ресурси сервера.

Третій, найвищий рівень конвеєра — інтелектуальний семантичний аналіз, виступає ключовим ноу-хау розробленого комплексу. Геопросторові записи, які успішно пройшли формальну перевірку (або ті, які мають статус некритичного попередження, але потребують експертної оцінки), автоматично делегуються до підсистеми інтеграції з хмарною мовною моделлю (Gemini API). Для кожного такого запису алгоритм програмно конструює структурований текстовий запит (промпт). Цей системний запит включає не лише «сирий» масив даних об'єкта, але й додаткові метадані про предметну область та строгі інструкції щодо пошуку неявних фізичних чи логічних суперечностей. Наприклад, модель повинна виявити невідповідність між текстовим літологічним описом породи та її числовими геофізичними показниками. Отримана від нейромережі текстова відповідь обробляється парсерами серверної частини і перетворюється у внутрішній формат звіту. Принциповою технічною особливістю цієї реалізації є те, що штучний інтелект не обмежується простою фіксацією наявності аномалії. Модель запрограмована на генерацію точного виправленого значення (Data Remediation Suggestion). Це згенероване виправлення автоматично передається на клієнтський веб-застосунок, де експерт із відповідними правами може оцінити його достовірність і одним кліком застосувати до оригінального запису, після чого система перезapiше оновлений об'єкт у хмарному сховищі OSDU.

4.8 Архітектура клієнтського застосунку та управління безпекою

Клієнтська частина програмного комплексу реалізована як динамічний односторінковий застосунок на базі бібліотеки React та суворо типізованої мови TypeScript.

Фундаментом безпеки клієнтського застосунку є інтеграція з бібліотекою `@azure/msal-browser` для реалізації протоколу OAuth2. Точкою входу в систему є компонент маршрутизатора, який обгортає всі робочі сторінки спеціалізованим компонентом вищого порядку (HOC) — `ProtectedRoute`. Цей компонент перевіряє наявність активної сесії Microsoft Entra ID (Azure AD). У разі відсутності авторизації користувач примусово перенаправляється на сторінку логіну. Для зберігання глобального стану користувача після авторизації використовується `React Context API`. Програмна абстракція `UserContext` витягує з серверу профіль користувача, кешує його ролі доступу (наприклад, `DataCustodian` або `DataSteward`) і робить їх доступними для будь-якого дочірнього компонента. Це дозволяє динамічно перебудовувати графічний інтерфейс: наприклад, приховувати кнопку створення правил або редагування записів від користувачів з рівнем прав `EndUser`. Аналогічним чином реалізовано `LanguageContext`, який ізолює логіку інтернаціоналізації і реактивно підставляє текстові значення зі словників у всі компоненти без перезавантаження дерева DOM.

Критично важливим архітектурним рішенням стала розробка єдиного шару мережевої абстракції. Замість того, щоб розпорошувати виклики `fetch` по десятках `React`-компонентів, було створено централізований API-фасад. Цей модуль містить суворо типізовані методи для кожної кінцевої точки серверу (наприклад, `exploreRecords`, `evaluateRecord`, `getRelationshipGraph`). Найголовніша функція цього фасаду — автоматичне перехоплення мережевих запитів (`Interceptor pattern`). Перед кожним викликом до сервера система асинхронно звертається до методу `getToken()`, який комунікує з екземпляром MSAL. Алгоритм намагається отримати токен у фоновому режимі, а у разі помилки `InteractionRequiredAuthError` — ініціює спливаюче вікно для повторної авторизації. Отриманий токен динамічно ін'єктується в HTTP-заголовки. Також на цьому рівні реалізовано глобальний перехоплювач помилок `extractError`, який стандартизує обробку кодів доступу (зокрема, генерує об'єкт `PermissionError` при отриманні HTTP 403), відв'язуючи компоненти інтерфейсу від рутинної обробки мережевих винятків.

Архітектура фронтенду спроектована за принципом розділення відповідальності, де логіка маршрутизації, управління станом сесії, взаємодія з мережею та безпосередній рендеринг компонентів повністю ізольовані одне від одного (рис. 4.6).

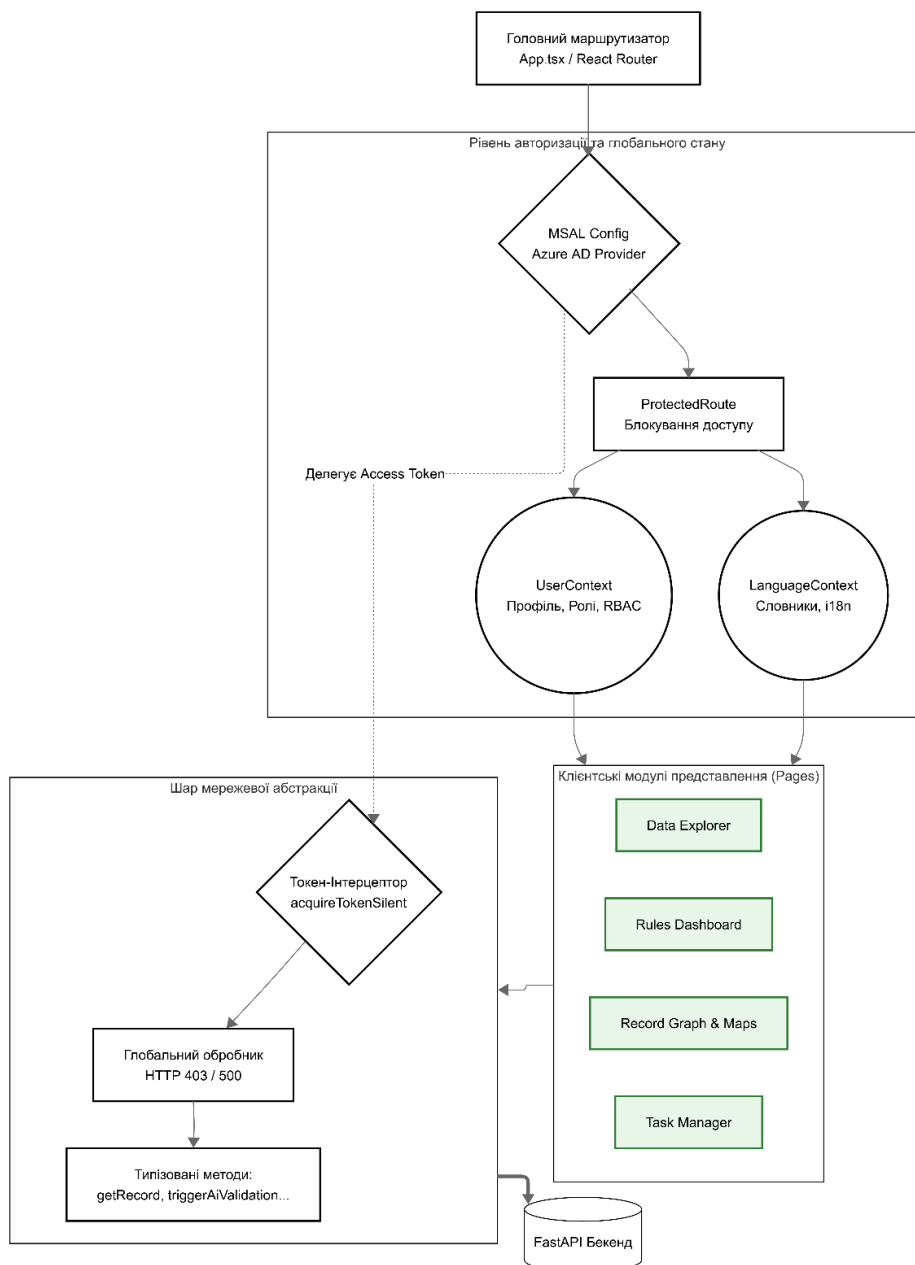


Рисунок 4.6 — Діаграма архітектури клієнтської частини застосунку

Самі графічні модулі інтерфейсу інкапсулюють специфічну бізнес-логіку. Наприклад, компонент `ExplorerRecordDetail` реалізує складний рекурсивний парсинг

ієрархічних JSON-документів OSDU для їх візуального відображення, а сторінки Map та WellsMap містять інтеграцію з картографічними бібліотеками, конвертуючимасиви координат OSDU в геопросторові полігони.

4.9 Реалізація функціональних модулів користувацького інтерфейсу

Клієнтський застосунок розділено на низку спеціалізованих функціональних модулів, кожен з яких вирішує конкретні задачі навігації, аналітики або управління даними (рис 4.7) [27].

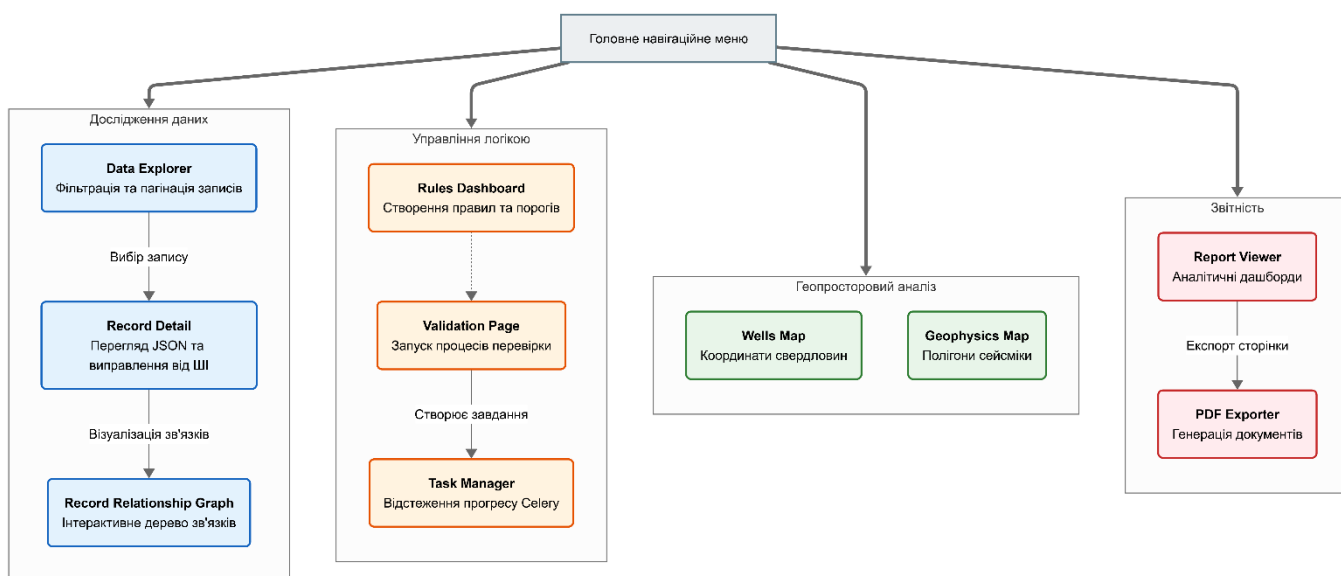


Рисунок 4.7 — Діаграма функціональних модулів

Ключовим інструментом для взаємодії з масивами інформації OSDU є модуль Data Explorer (Провідник даних). Він реалізований у вигляді динамічної таблиці з підтримкою серверної пагінації та багатокритеріальної фільтрації. Користувач може фільтрувати записи за їхнім типом (наприклад, виокремлювати лише master-data--Wellbore або work-product-component--SeismicHorizon), статусом

валідації чи приналежністю до певних бізнес-одиниць. При виборі конкретного запису відкривається сторінка деталей, де складний ієрархічний JSON-документ парситься і візуалізується у вигляді зручних деревовидних структур. На цій же сторінці реалізовано інтерфейс інтелектуального виправлення, користувачу відображаються згенеровані ІІІ підказки, які можна застосувати до запису.

Управління бізнес-логікою здійснюється через модуль Rules Dashboard. Цей інтерфейс надає адміністраторам даних інструментарій для повного циклу управління правилами перевірки якості (CRUD-операції). Форми створення правил містять вбудовану валідацію на стороні клієнта: користувач може задавати регулярні вирази, вказувати числові пороги (мінімум/максимум) для геофізичних показників та обирати цільові атрибути зі списку, що динамічно підвантажується із Schema API платформи OSDU. Для ручного запуску перевірок розроблено сторінку Validation, де експерт може обрати конкретні об'єкти та примусово пропустити їх через активні правила або направити на семантичний аналіз до мовної моделі. Для відстеження таких тривалих процесів використовується модуль Task Manager, який періодично опитує сервер і відображає прогрес-бари виконання фонових завдань Celery.

Враховуючи геопросторову специфіку даних, розроблено модулі візуалізації на карті (WellsMap та Map). Ці компоненти інтегровані з картографічними бібліотеками і здатні динамічно рендерити візуальні шари на основі просторових метаданих OSDU. Наприклад, система автоматично витягує координати (широту та довготу) зі структури SpatialLocation запису свердловини і наносить їх на карту у вигляді маркерів, а для масштабних сейсмічних зйомок — малює географічні контури.

Особливою технічною інновацією клієнтського інтерфейсу є програмна реалізація модуля Record Relationship Graph. Оскільки об'єкти OSDU пов'язані між собою через складні масиви посилань, використовуючи графічні бібліотеки, система генерує інтерактивну діаграму, де користувач може візуально простежити, як помилка в базовому дослідженні впливає на всі похідні документи [9].

Наостанок, для забезпечення потреб корпоративного аудиту, створено модуль генерації звітності. Цей скрипт конвертує HTML-представлення сторінки «Звіти» у відформатований PDF-документ, дозволяючи завантажити результати валідації [9].

5 ІНСТРУКЦІЯ КОРИСТУВАЧА ТА ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМНОЇ СИСТЕМИ

Розробка сучасного програмного комплексу вимагає не лише надійної серверної архітектури, але й забезпечення зрозумілої та безперебійної взаємодії з кінцевим користувачем. Відповідно досить важливо присвятити час практичним аспектам експлуатації створеної інтелектуальної системи контролю якості даних платформи OSDU. Навести вичерпний опис життєвого циклу розгортання програмного продукту, починаючи від визначення мінімальних системних вимог та інструкцій з інсталяції за допомогою технологій контейнеризації Docker, до безпосередньої повсякденної роботи у веб-інтерфейсі.

5.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної та високопродуктивної роботи розробленого інтелектуального програмного комплексу необхідно дотримуватись визначених системних вимог.

— процесор: Сучасний багатоядерний процесор (рекомендується щонайменше 4 фізичних ядра). Наявність кількох ядер є критичною для паралельного виконання фонових завдань алгоритмом Celery, який здатний динамічно масштабуватися до 4-х незалежних процесів-робітників;

— оперативна пам'ять: Мінімально допустимий обсяг становить 4 ГБ, однак для комфортної роботи та обробки масштабних запитів до екосистеми OSDU настійно рекомендується 8 ГБ. Значний обсяг пам'яті необхідний для підтримання резидентного сховища Redis, яке виконує роль «гарячого» кешу для тисяч серіалізованих геопросторових документів;

— дисковий простір: Не менше 10 ГБ вільного місця на твердотільному накопичувачі (SSD) для зберігання базових образів Docker, ізольованих контейнерів та кешу операційної системи.

Оскільки архітектура системи базується на мікросервісному підході та активно використовує технології контейнеризації, вимоги висуваються як до апаратної, так і до програмної частини серверного (або локального) середовища (рис. 5.1).

ID %	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET IO	BLOCK IO	PIDS	CPU TIME	AVG CPU
e65b731adc8f	redis	0.99%	508.2MB / 16.47GB	3.09%	3.247MB / 48.21MB	0B / 0B	28	36.5619807s	0.99%
9bc420035a01	diploma-diduk	1.53%	873.6MB / 16.47GB	5.30%	48.46MB / 8.933MB	0B / 0B	40	56.0883608s	1.53%
2907da7e0bfe	diploma-frontend	0.00%	14.75MB / 16.47GB	0.09%	6.721MB / 8.526MB	0B / 0B	23	122.0917ms	0.00%

Рисунок 5.1 — Статистика використання ресурсів застосунку в бездіяльному стані

Програмні вимоги:

— операційна система: Кросплатформність технологій дозволяє розгорнути систему на базі Windows 10/11 (з обов’язковою підтримкою підсистеми WSL2), сучасних дистрибутивів Linux (Ubuntu 20.04+, CentOS) або macOS;

— інфраструктура: Наявність встановленого середовища Docker Engine (версії 20.10 або вище) та утиліти оркестрації Docker Compose (версії 2.0+);

— клієнтське ПЗ: Для роботи з вебінтерфейсом потрібен будь-який сучасний браузер із підтримкою стандарту ECMAScript 6 (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).

5.2 Процес інсталяції та розгортання системи

Завдяки впровадженню методології контейнеризації для всіх компонентів архітектури (як бекенду, так і фронтенду), процес розгортання системи максимально оптимізовано та позбавлено проблеми «конфлікту залежностей» на локальній машині. Процес інсталяції складається з кількох послідовних кроків.

На першому етапі адміністратор клонує вихідний код проєкту у цільову директорію сервера. Далі здійснюється конфігурація середовища: у кореневій теці створюється файл параметрів `.env`, куди вносяться конфіденційні облікові дані. Зокрема, файл повинен містити ключі доступу до корпоративного провайдера Microsoft Azure AD (`CLIENT_ID`, `TENANT_ID`, `CLIENT_SECRET`), адреси мікросервісів платформи OSDU (`OSDU_BASE_URL`, `DATA_PARTITION_ID`), конфігурації для API штучного інтелекту Gemini та параметри підключення до брокера Redis.

Після збереження конфігураційного файлу адміністратор відкриває термінал у кореневій папці проєкту та виконує єдину команду запуску оркестратора: `./deploy.sh`. Після ініціалізації цієї команди система автоматично виконує наступні дії:

- завантажує оптимізований офіційний образ бази даних Redis (`redis-stack`) та монтує для нього локальний диск для забезпечення стійкості даних (`persistence`);
- здійснює складання образу серверної частини: встановлює Python-залежності та ініціалізує API-сервер разом із фоновими воркерами Celery;
- запускає збірку клієнтського застосунку, спочатку компілює React-код через Node.js, а потім переносить оптимізовані статичні файли в контейнер Nginx.

По завершенню процесу збірки, який триває кілька хвилин, програмний комплекс вважається повністю розгорнутим. Клієнтський застосунок стає автоматично доступним за адресою `http://localhost:4200`. Вбудований у контейнер проксі-сервер Nginx самостійно перенаправляє всі системні API-запити до ізольованого контейнера бекенду, забезпечуючи цілісну роботу платформи [28].

5.3 Авторизація та навігація системою

Після успішного запуску всіх контейнерів та переходу у браузер, першим екраном, з яким стикається користувач, є сторінка авторизації [29]. Для входу в систему користувачеві необхідно натиснути центральну кнопку автентифікації, після

чого інтегрований модуль MSAL ініціює безпечний процес передачі даних за протоколом OAuth2 (рис. 5.2).

Оскільки розроблений програмний комплекс оперує конфіденційною корпоративною геопросторовою інформацією, анонімний доступ до будь-якого функціонала системи повністю заблоковано на рівні маршрутизатора.

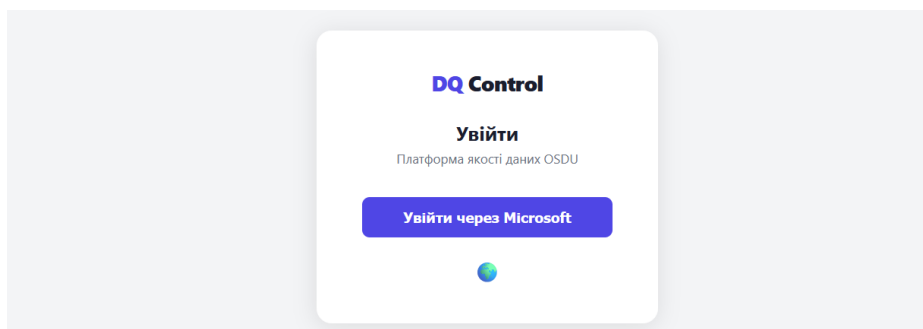


Рисунок 5.2 — Сторінка авторизації до застосунку

Користувачу відображається вікно від провайдера Azure AD, де він має ввести свої існуючі корпоративні облікові дані і підтвердити надання дозволів (рис. 5.3).

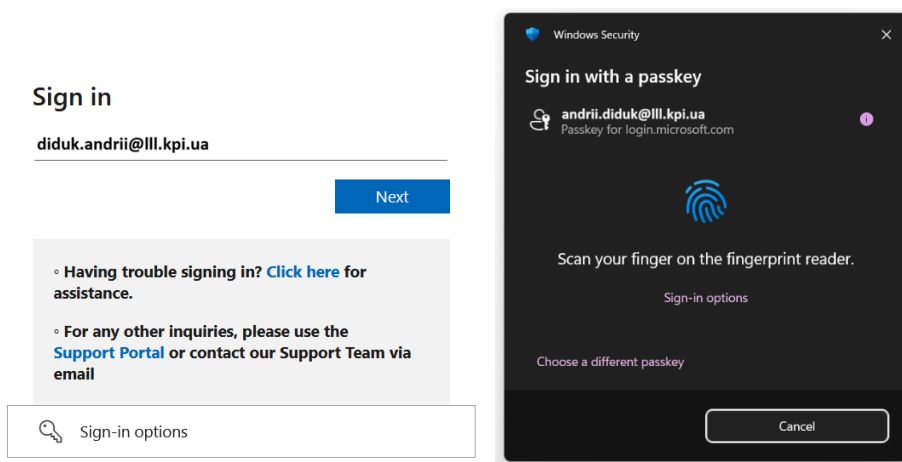


Рисунок 5.3 — Процес авторизації через обліковий запис Microsoft

Після успішної перевірки даних клієнтська частина системи перехоплює згенерований токен доступу (Bearer Token), безпечно фіксує його у стані браузера і

автоматично перенаправляє користувача до робочої зони. Навігація між усіма функціональними модулями (картами, провідником даних, редактором правил) здійснюється за допомогою статичної панелі меню. Важливою особливістю навігаційної системи є безшовне використання механізму Role Guard. Залежно від рівня привілеїв авторизованого працівника інтерфейс динамічно адаптується: адміністратори бачать повний набір інструментів, тоді як від кінцевих споживачів приховані вкладки управління.

5.4 Аналітичний дашборд та загальний огляд стану системи

Головна сторінка застосунку — це аналітичний дашборд який виконує функцію єдиного інформаційного центру, що дозволяє експерту миттєво оцінити поточний стан якості геоданих (рис. 5.4).

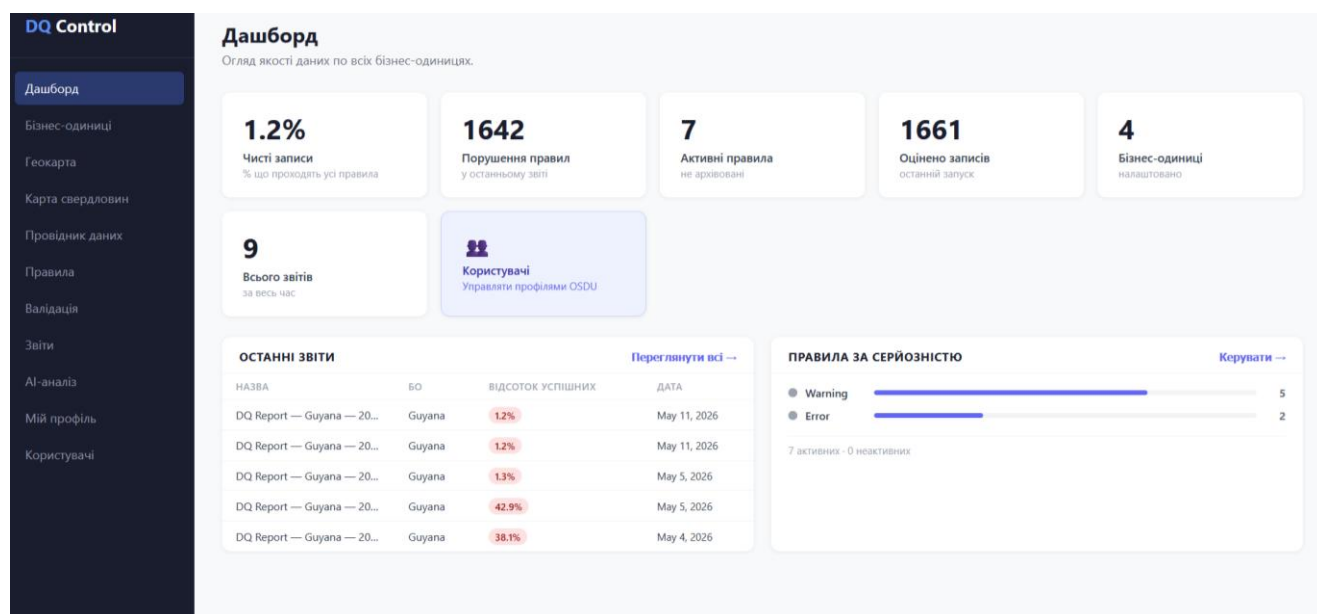


Рисунок 5.4 — Головна сторінка застосунку

У верхній частині дашборда розташована інтерактивна панель ключових показників. На ній агреговано відображаються найважливіші метрики, отримані на

основі останніх процесів валідації: відсоток повністю коректних записів, загальна кількість виявлених порушень, кількість активних бізнес-правил, кількість перевірених записів та налаштованих бізнес-одиниць. Кожна картка є клікабельною і дозволяє в один клік перейти до відповідного детального звіту. Важливою архітектурною деталлю є те, що дашборд є контекстно-залежним: якщо авторизований користувач має адміністративну роль, у цій панелі автоматично з'являється додаткова картка для переходу до модуля управління користувачами.

Під панеллю ключових показників інтерфейс ергономічно розділено на два інформаційні блоки. Лівий блок містить таблицю «Останні звіти», яка дозволяє швидко переглядати історію масових перевірок. У цій таблиці наочно підсвічується відсоток успішності — зеленим кольором, якщо він перевищує норму (80%), і червоним у разі критичної кількості знайдених аномалій. Правий блок («Правила за серйозністю») відображає детальну статистику налаштованих правил якості, згруповану за рівнем критичності. Візуалізація здійснюється за допомогою графічних індикаторів та смуг завантаження відповідних кольорів, що дозволяє адміністраторам даних з першого погляду оцінити, яка саме бізнес-логіка наразі домінує в системі [25].

5.5 Навігація бізнес-одиницями та геопросторова візуалізація

Однією з ключових функціональних вимог до розробленого програмного комплексу є здатність структурувати та обробляти дані в контексті глобальної географії корпорації. Для вирішення цієї задачі в системі реалізовано модуль управління регіонами «Бізнес-одиниці». При переході до цього розділу через головне меню користувачеві відображається інтерактивна панель із переліком доступних регіональних підрозділів компанії (наприклад, проєкти у Гаяні, Північному морі тощо). Кожна картка бізнес-одиниці містить не лише базову ідентифікаційну інформацію, але й динамічні індикатори якості даних. Система автоматично агрегує результати останніх фонових перевірок і виводить для кожного підрозділу відсоток повністю коректних записів, кількість активних порушень та загальний обсяг

проаналізованих об’єктів. Це дозволяє керівництву швидко та наочно оцінювати стан геопросторової інформації в розрізі регіонів (рис. 5.5).

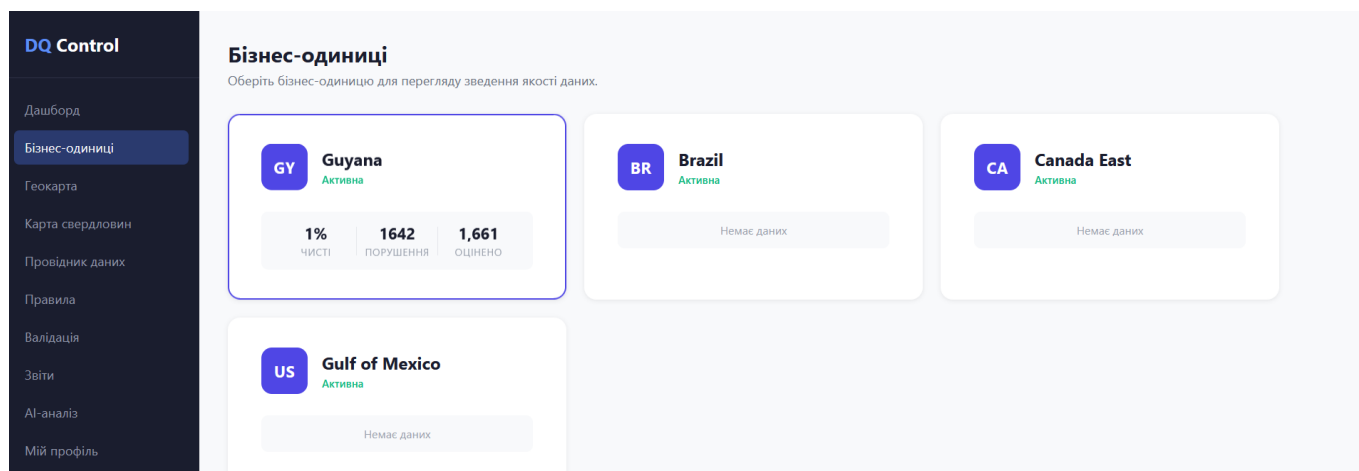


Рисунок 5.5 — Сторінка бізнес одиниць

При кліку на конкретну бізнес-одиницю (наприклад, Guyana) відкривається екран її деталізації (рис. 5.6). На цьому екрані користувач бачить розбивку масивів інформації за так званими предметними доменами. Домени логічно групують геопросторові записи OSDU за їх фізичною природою: наприклад, «Seismic/Geophysics» (Сейсморозвідка) або «Wells» (Дані свердловин).

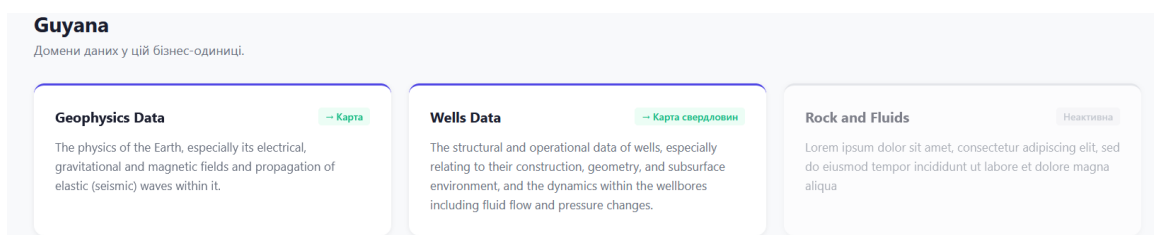


Рисунок 5.6 — Предметні області конкретної бізнес одиниці

Кожна картка домену містить статус його активності в системі та спеціальні інформаційні бейджі [30]. Якщо система ідентифікує, що домен містить просторові координати, вона автоматично робить його активним і пропонує користувачеві перейти до картографічної візуалізації.

5.6 Візуалізація та фільтрація геофізичних даних

Для роботи з масштабними об'єктами сейсмозвідки в системі реалізовано окремий геопросторовий модуль. На відміну від звичайних точкових координат, геофізичні дані в OSDU часто представлені складними просторовими полігонами (Bounding Boxes), які вказують на площу проведення сейсмічної зйомки або іншої операції.

Модуль інтегрований із картографічною бібліотекою Leaflet і динамічно рендерить GeoJSON-об'єкти поверх географічної основи. Кожен тип геофізичних даних має власне кольорове кодування на карті. Наприклад, базові дослідження (SeismicAcquisitionSurvey) відображаються синім кольором, проєкти обробки (SeismicProcessingProject) — зеленим, а набори 3D-інтерпретацій (Seismic3DInterpretationSet) — помаранчевим. Клік на будь-який полігон ініціює відкриття детальної картки відповідного запису в модулі Data Explorer для подальшого аналізу.

Особливою перевагою модуля є розширена система фільтрації, яка дозволяє експертам швидко знаходити потрібні дані. Окрім стандартного текстового пошуку за назвою проєкту, система пропонує глибоку вибірку на основі графа зв'язків OSDU. Користувач може відфільтрувати об'єкти, які мають специфічні робочі компоненти, містять певні формати наборів даних або відповідають заданим часовим діапазнам (за датами завантаження, модифікації або безпосереднього проведення сейсмозйомки).

Окрім цього, інтерфейс карти автоматично підраховує статистику якості геоданих: він показує, скільки об'єктів успішно нанесені на карту, а скільки записів позбавлені блоку SpatialLocation (без координат).

Спеціальна кнопка «Переглянути записи без геоданих» дозволяє миттєво згенерувати табличний звіт щодо таких проблемних об'єктів для їх подальшого огляду (рис. 5.7).

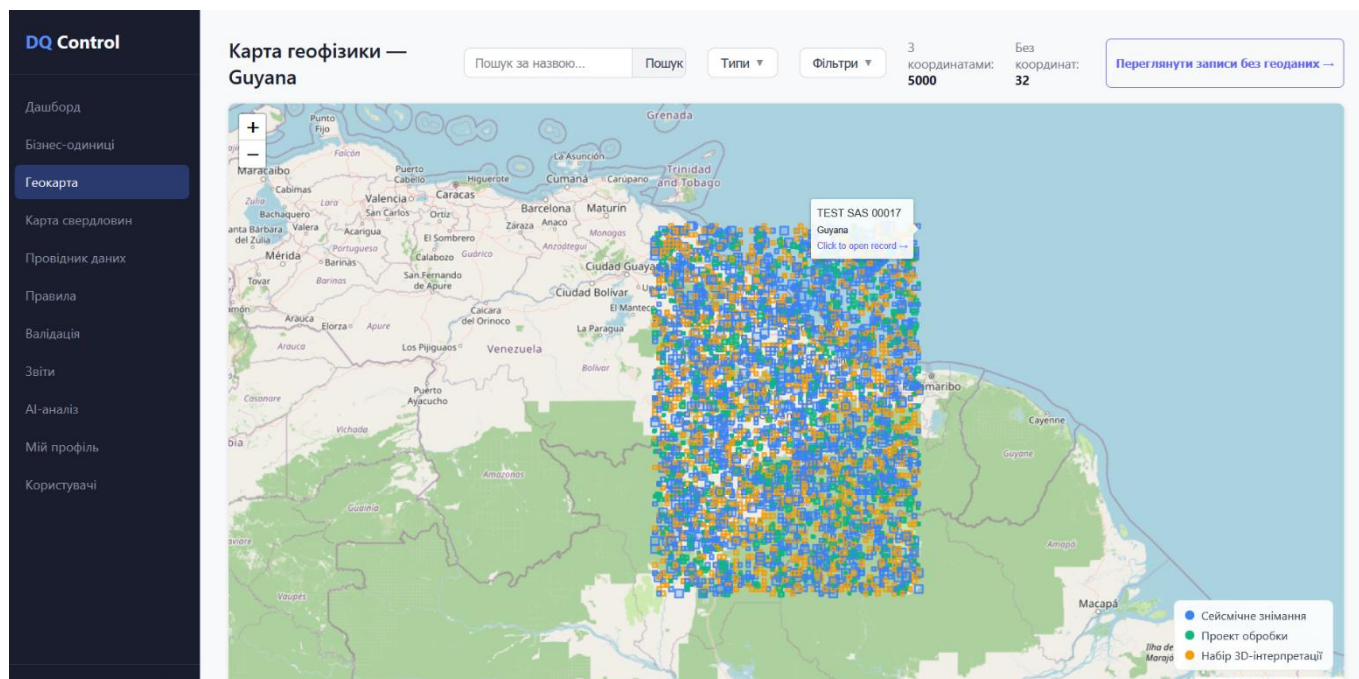


Рисунок 5.7 — Мапа геофізичних даних

Мапа також автоматично відкривається саме в тому місці де знаходяться записи, тобто у відповідність з країною де вони оперують.

5.7 Візуалізація об'єктів видобутку: Карта свердловин

Для роботи зі структурою фізичних активів нафтогазової компанії розроблено відокремлений модуль «Карта свердловин». На відміну від геофізичної карти, яка працює з площинними полігонами, цей інтерфейс оптимізовано для рендерингу точкових об'єктів, безпосередньо свердловин та їхніх стовбурів.

Логіка візуалізації на карті свердловин враховує ієрархічні зв'язки об'єктів. Базові свердловини позначаються більшими маркерами синього кольору (радіус 7 пікселів), тоді як залежні стовбури меншими маркерами помаранчевого кольору. Якщо система виявляє, що стовбур свердловини пов'язаний із базовою свердловиною, алгоритм фронтенду автоматично генерує з'єднувальну лінію (GeoJSON LineString) між їхніми координатами. Це дозволяє геологам візуально

перевіряти просторову коректність прив'язки стовбурів до гирла свердловини: аномально довгі лінії миттєво сигналізують про помилку в координатах одного з об'єктів (рис. 5.8).

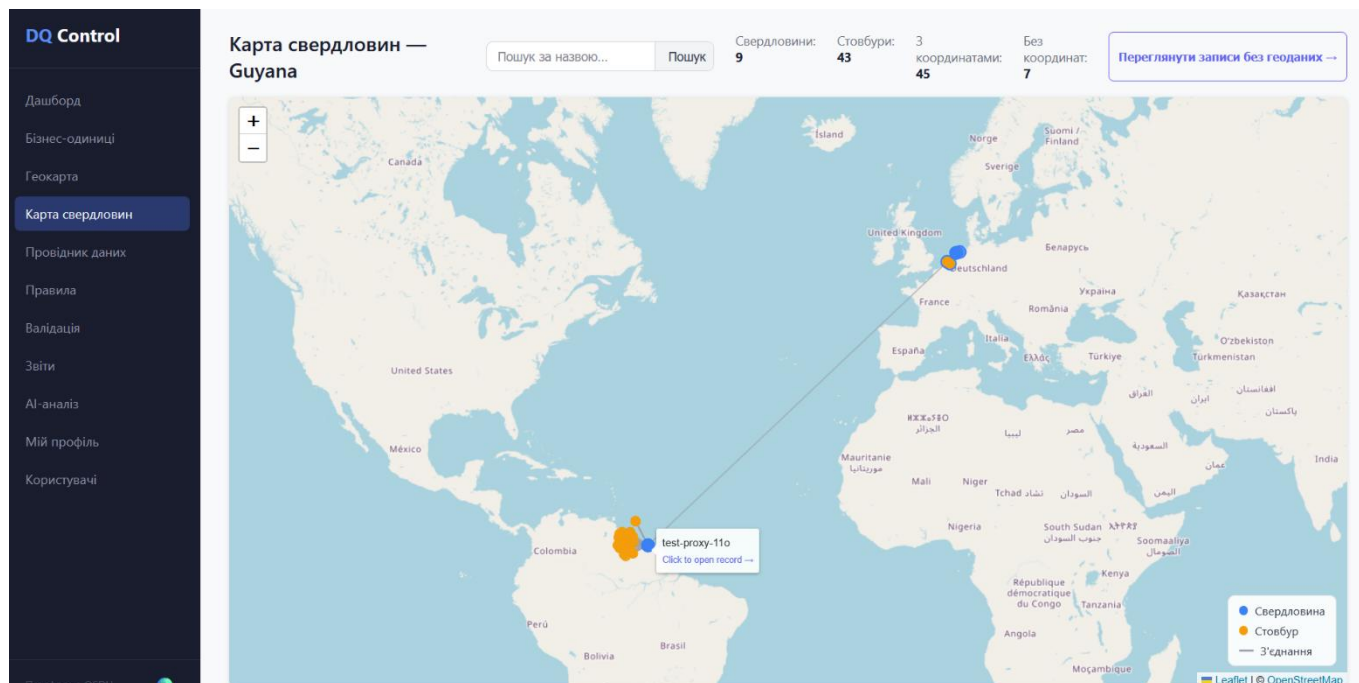


Рисунок 5.8 — Мапа свердловин

Так само як і на геофізичній карті, користувачу доступний текстовий пошук за назвою об'єкта (FacilityName) та загальна панель статистики. Панель динамічно відображає підраховану кількість свердловин та стовбурів у межах обраної бізнес-одиниці, а також виділяє ті об'єкти, координати яких відсутні або заповнені некоректно, з можливістю переходу до модуля ручного виправлення.

5.8 Робота з каталогом даних

Модуль «Провідник даних» є центральним інструментом системи, призначеним для зручної навігації, пошуку та фільтрації багатотисячних масивів геопросторової

інформації OSDU (рис. 5.9). Інтерфейс модуля спроектовано за принципом пошукової системи з розширеними фасетними фільтрами.

У верхній частині екрана розташована панель управління типами даних (Kind Selector). Користувач може обирати одну або декілька категорій об’єктів (наприклад, тільки свердловини, сейсмічні проєкти або специфічні компоненти). Нижче знаходиться рядок текстового пошуку, який дозволяє знаходити записи за їхньою унікальною назвою чи ідентифікатором.

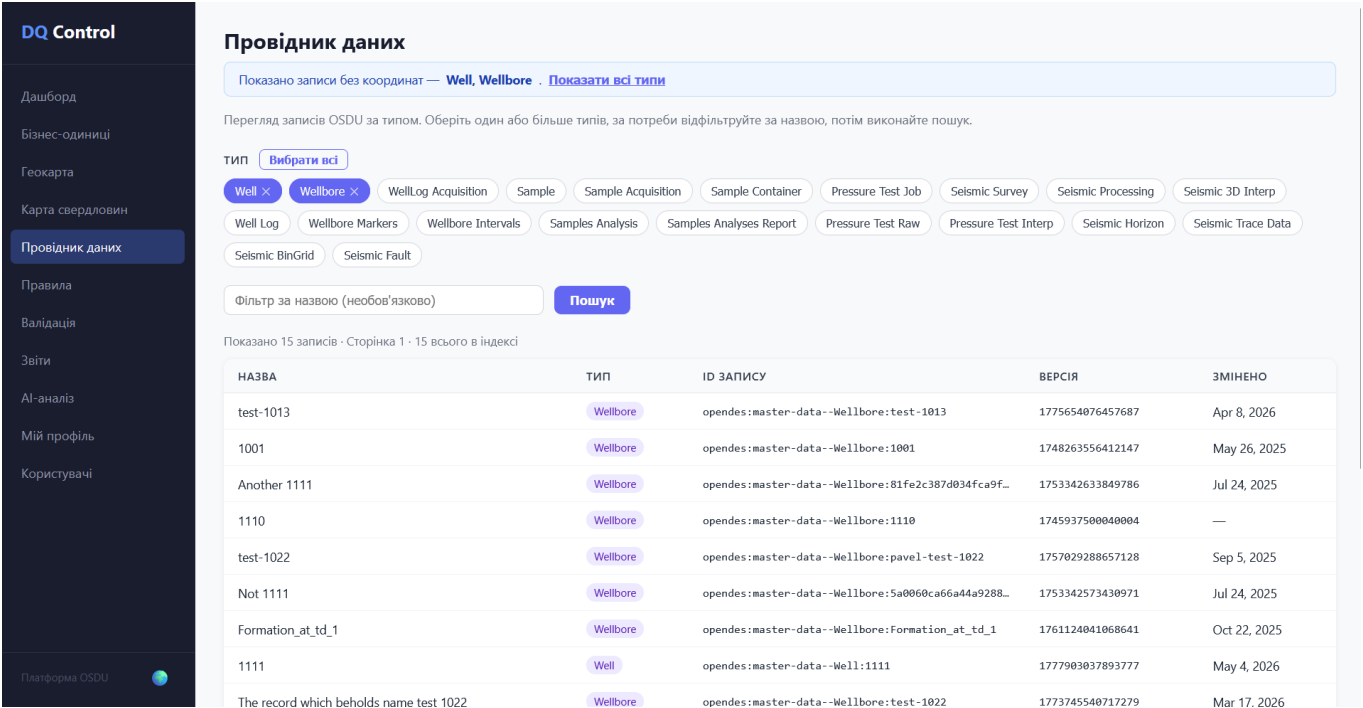


Рисунок 5.9 — Провідник даних

Важливою функцією модуля є підтримка контекстної фільтрації. Якщо користувач перейшов до Провідника зі сторінки картографії через кнопку «Переглянути записи без геоданих», таблиця автоматично відфільтрує і покаже виключно ті записи, в яких відсутній або пошкоджений блок просторових метаданих (SpatialLocation). Результати пошуку виводяться у вигляді динамічної таблиці з підтримкою серверної пагінації (сторінковості). Таблиця відображає назву об’єкта, його тип, унікальний ідентифікатор, версію та дату останньої модифікації.

5.9 Управління бізнес-правилами

Оскільки якість даних є динамічним показником, системі потрібен гнучкий інструмент для налаштування критеріїв цієї якості. Для цього розроблено модуль управління правилами (рис. 5.10). Цей інтерфейс взаємодіє з базою даних Redis, де зберігаються всі Data Quality Rules (DQ Rules).

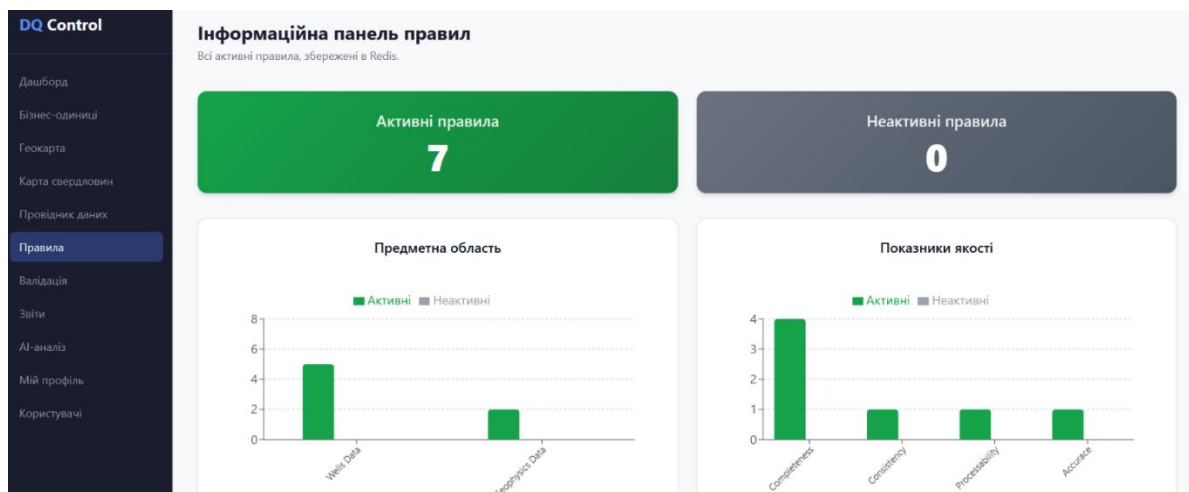


Рисунок 5.10 — Сторінка правил з аналітикою

Інтерфейс модуля поділено на два режими перегляду. Режим «Аналітика» надає загальну статистику за допомогою інтерактивних графіків, побудованих на базі бібліотеки Recharts. Користувач бачить загальну кількість активних, неактивних правил, а також гістограми, що розподіляють правила за предметними доменами та вимірами якості (Completeness, Accuracy, тощо).

Режим «Таблиця» надає детальний перелік усіх правил із можливістю текстового пошуку та сортування (рис. 5.11).

Користувачі з відповідними правами доступу можуть створювати нові або редагувати існуючі правила через спеціальне модальне вікно. Процес створення правила максимально автоматизовано:

— користувач обирає тип об'єкта OSDU (наприклад, Wellbore);

- фронтенд динамічно завантажує JSON-схему цього об'єкта та пропонує розумне автозаповнення для вибору цільового поля (наприклад, `data.FacilityName`);
- задається логічний оператор (наприклад, `isNull`, `isBetween`, `isEqual`) та очікувані еталонні значення;
- встановлюється рівень критичності (`Severity: Error, Warning, Info`) та метадані безпеки (`ACL` та `Legal Tags`) для подальшої синхронізації з хмарою.

Правила якості даних
Всі активні правила, збережені в Redis.

← Назад до панелі 7 правил [+ Додати правило](#)

ID	НАЗВА	ТИП	АТРИБУТ	ОПЕРАТОР	ЗНАЧЕННЯ
id14	data.SampleName is null	Sample	data.SampleName	isNull	—
id16	data.ProjectSpecifications[*].ProjectSpecificationQuantity should equal to '12345.6.'	Seismic3DInterpretationSet	data.ProjectSpecifications[*].ProjectSpecificationQuantity	isEqual	12345.6
id10	data.ResourceHostRegionIDs[*] should equal to 'opendes:reference-data--OSDURRegion:AWSEastUSA:'.	WellLogAcquisition	data.ResourceHostRegionIDs[*]	isEqual	opendes:refe OSDURRegion:A
id13	TechnicalAssurances must be null	Well	data.TechnicalAssurances	isNull	—
id12	SlotID must equal to 'WellSiteStructure:1100009117-Slot'	Well	data.SlotID	isEqual	WellSiteStru Slot
id11	TechnicalAssurances must be not null	Well	data.TechnicalAssurances	isNotNull	—
id15	data.TechnicalAssuranceID should be not null.	Seismic3DInterpretationSet	data.TechnicalAssuranceID	isNotNull	—

Рисунок 5.11 — Сторінка правил в табличному форматі

Після створення/редагування правила, зміни зберігаються одночасно в OSDU та Redis для синхронізації.

5.10 Запуск та контроль конвеєра валідації

Після того, як бізнес-правила налаштовані, експерти використовують модуль «Валідація» для масової перевірки записів. Цей інтерфейс дозволяє ініціювати процес оцінки даних для цілої бізнес-одиниці (наприклад, перевірити всі об'єкти в регіоні Guyana).

На панелі інструментів користувач обирає цільову бізнес-одиноцю та джерело даних для перевірки (безпосередньо з OSDU Storage або з кешу Redis для прискорення процесу). Натискання кнопки «Запустити оцінку» відправляє запит на бекенд, після чого на екрані з'являється панель агрегованої статистики. Панель складається з інтерактивних карток: загальна кількість проаналізованих записів, успішні, з помилками, пропущені та загальний відсоток чистоти даних (Чисті %) (рис. 5.12).

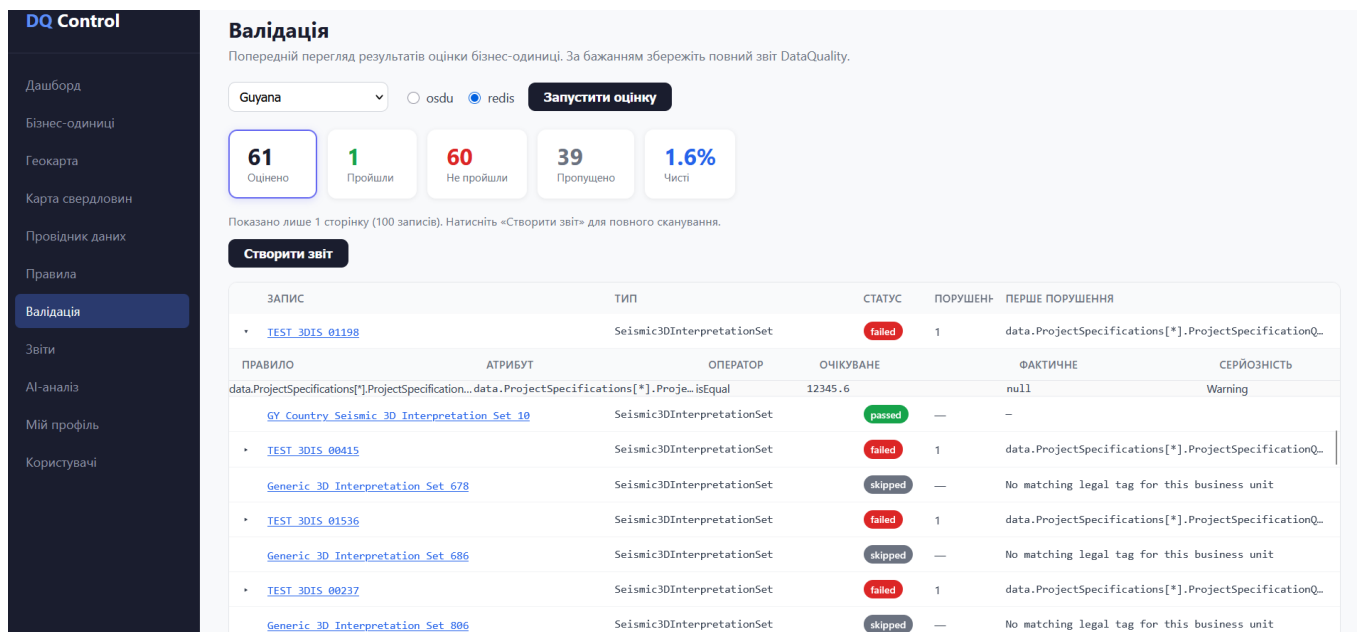


Рисунок 5.12 — Сторінка валідації

Клік по будь-якій з цих карток миттєво фільтрує детальну таблицю результатів, розташовану нижче. У таблиці виводяться ідентифікатори перевірених записів та їхні статуси. Якщо запис має статус «Failed», користувач може розгорнути рядок таблиці і побачити точний перелік порушених правил, яке саме поле перевірялося, який оператор використовувався, яке значення очікувалося і яке було знайдено в реальності. Звідси ж передбачено швидкий перехід до модуля Data Explorer для точкового виправлення помилки.

Окрім візуального аналізу, сторінка має критично важливу кнопку «Створити звіт». Вона викликає модальне вікно для генерації формального звіту (рис. 5.13).

Після підтвердження система ініціює фонову задачу, яка збирає результати валідації і формує постійний об'єкт звіту у каталозі OSDU для цілей аудиту.

Рисунок 5.13 — Створення звіту

Даний звіт також зберігається одночасно до двох джерел, OSDU й Redis для синхронізації даних.

5.11 Менеджер звітів та експорт результатів

Важливою вимогою до корпоративних систем управління якістю даних є наявність інструментів для аудиту та збереження історичної звітності. Для задоволення цієї вимоги в системі реалізовано модуль «Звіти», який акумулює всі результати перевірок, ініційованих користувачами (рис. 5.14).

Головна сторінка модуля відображає хронологічний реєстр усіх згенерованих звітів у вигляді таблиці. Кожен рядок містить ключову метаінформацію: назву звіту, цільову бізнес-одиницю, джерело даних (OSDU чи Redis), дату генерації та агреговану статистику (кількість перевірених записів, скільки з них пройшли

перевірку, скільки провалили, і загальну кількість знайдених порушень). Цей інтерфейс слугує єдиним вікном для керівників дата-доменів, дозволяючи їм відстежувати динаміку очищення даних з плином часу.

Звіти
Звіти оцінки якості даних. Натисніть на рядок для перегляду деталей.

НАЗВА	БО	ДЖЕРЕЛО	ДАТА	ОЦІНЕНО	ПРОЙШЛИ	НЕ ПРОЙШЛИ	ПОРУШЕННЯ	PDF
DQ Report — Guyana — 2026-05-11T13:39:44+0000	Guyana	redis	5/11/2026, 4:39:44 PM	1661	20	1641	1642	Створити PDF
DQ Report — Guyana — 2026-05-11T12:56:08+0000	Guyana	redis	5/11/2026, 3:56:08 PM	1661	20	1641	1642	Створити PDF
DQ Report — Guyana — 2026-05-05T16:21:35+0000	Guyana	redis	5/5/2026, 7:21:35 PM	1661	21	1640	1641	Створити PDF
DQ Report — Guyana — 2026-05-05T13:52:59+0000	Guyana	redis	5/5/2026, 4:52:59 PM	21	9	12	13	Створити PDF
DQ Report — Guyana — 2026-05-04T11:04:03+0000	Guyana	redis	5/4/2026, 2:04:03 PM	21	8	13	14	Створити PDF
DQ Report — Guyana — 2026-04-30T13:52:46+0000	Guyana	osdu	4/30/2026, 4:52:46 PM	21	8	13	14	Створити PDF
DQ Report — Guyana — 2026-04-29T19:06:15+0000	Guyana	redis	4/29/2026, 10:06:15 PM	21	8	13	14	Створити PDF
DQ Report — Guyana — 2026-04-29T17:22:19+0000	Guyana	redis	4/29/2026, 8:22:19 PM	21	8	13	14	Створити PDF
DQ Report — Guyana — 2026-04-28T17:11:06+0000	Guyana	redis	4/28/2026, 8:11:06 PM	21	7	14	15	Створити PDF

Рисунок 5.14 — Сторінка звітності

При кліку на конкретний звіт відкривається сторінка його деталізації. Як і на екрані валідації, користувачеві доступні клікабельні статистичні картки для миттєвої фільтрації таблиці результатів (наприклад, показати лише записи зі статусом «Failed»). У таблиці деталізації виводяться ідентифікатори перевірених записів, перелік порушених атрибутів у вигляді компактних бейджів, а також унікальний `osdu_dq_id`, який бекенд-система автоматично зберегла в хмарному сховищі OSDU для цього звіту (рис. 5.15).

Така архітектурна побудова сторінки деталізації забезпечує повну простежуваність результатів перевірки якості даних, дозволяючи фахівцям оперативно локалізувати проблемні записи в межах великих масивів інформації. Наявність прямого зв’язку з OSDU-ідентифікатором надає користувачам можливість миттєво переходити від результатів звіту до вихідного об’єкта в хмарній інфраструктурі, що значно оптимізує процеси корекції даних та знижує рівень технічного боргу системи.

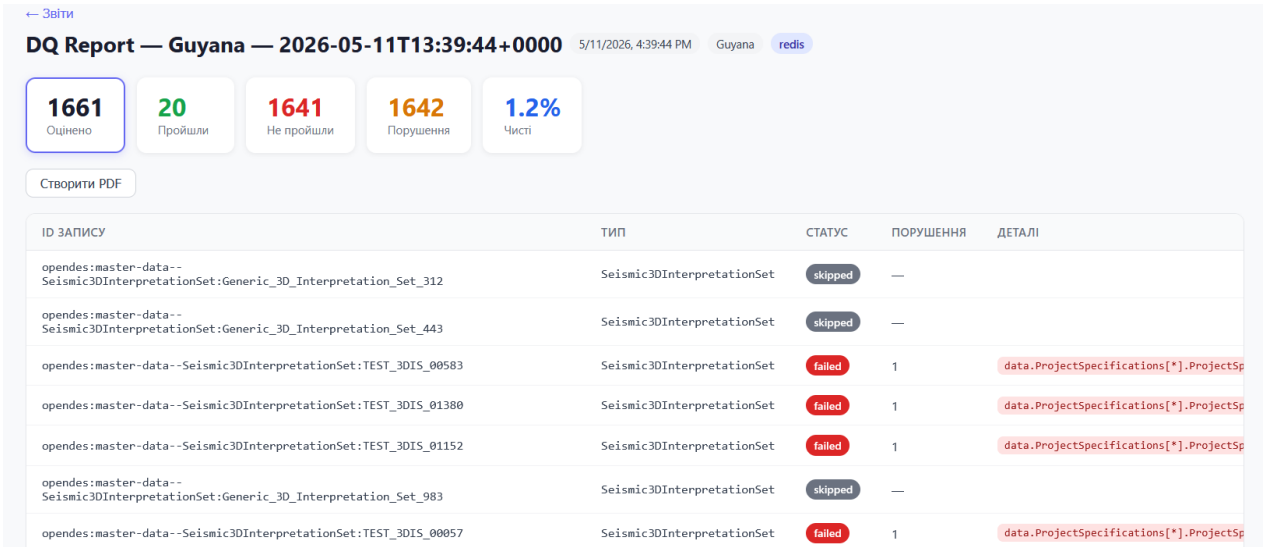


Рисунок 5.15 — Сторінка обраного звіту

Особливістю цього модуля також є функція клієнтської генерації PDF-документів (рис. 5.16).

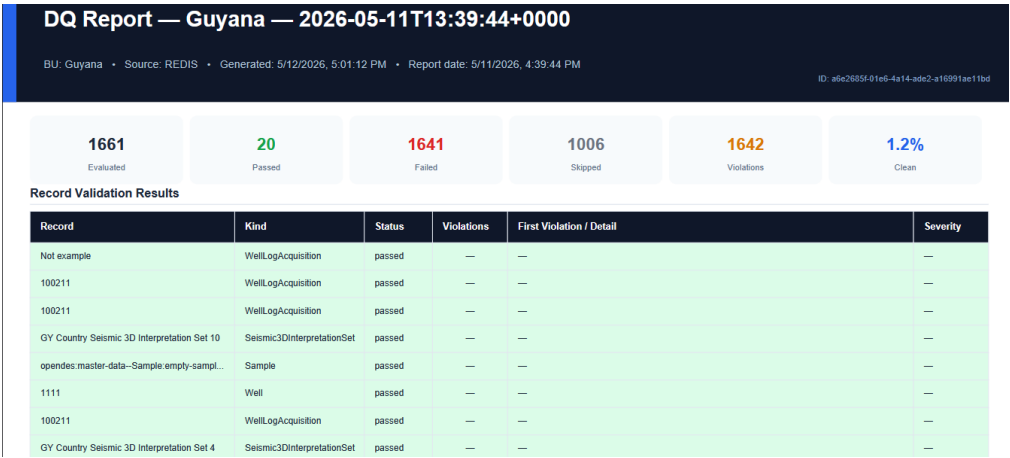


Рисунок 5.16 — Частина згенерованого PDF файлу на основі звіту

Замість того щоб навантажувати сервер рендерингом документів, фронтенд-застосунок використовує бібліотеки jspdf та jspdf-autotable для миттєвого створення PDF-файлів прямо в браузері. Користувач натискає кнопку «Створити PDF», після чого скрипт автоматично формує брендований документ. Згенерований файл містить титульну сторінку з підсумковою статистикою та деталізовані таблиці з переліком

усіх знайдених аномалій, що робить його ідеальним форматом для пересилання електронною поштою підрядникам чи операторам бази даних для подальшого виправлення.

5.12 Детальний аналіз записів та інтелектуальне виправлення

При натисканні на будь-який рядок у таблиці Провідника або всередині звіту відкривається сторінка детального аналізу конкретного запису (рис. 5.17).

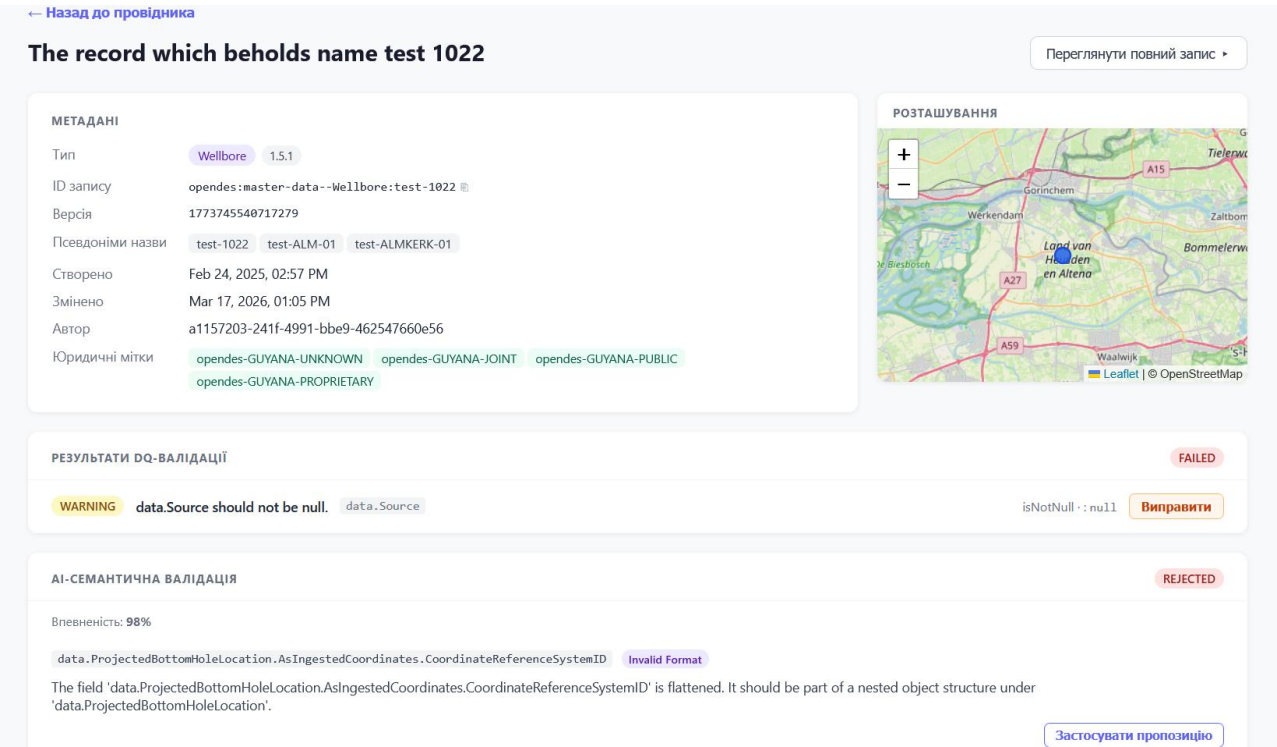
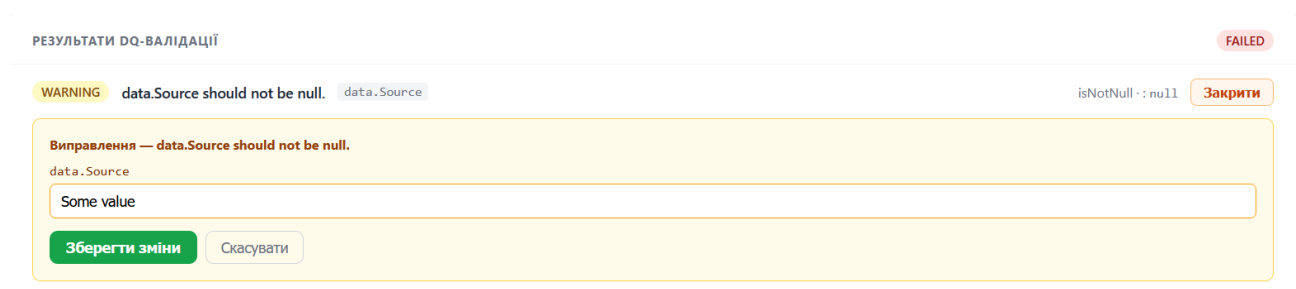


Рисунок 5.17 — Сторінка деталей запису

Цей екран акумулює всю наявну інформацію про об’єкт. У верхній частині відображається панель метаданих (дата створення, автор, легальні теги) та вбудована міні-карта, яка одразу показує географічне розташування саме цього запису [10]. Також передбачена можливість перегляду повного «сирого» JSON-документа у зручному деревовидному форматі за допомогою спеціального модального вікна.

Ключовим функціоналом цієї сторінки є секції перевірки якості даних. Перший блок — це автоматична системна валідація (рис. 5.18). Алгоритм перевіряє запис на відповідність усім активним бізнес-правилам, збереженим у Redis. Якщо знайдено порушення (наприклад, відсутній обов’язковий атрибут або значення виходить за межі норми), система виводить список помилок із зазначенням їх критичності. Натиснувши кнопку «Виправити», експерт відкриває вбудовану форму, де може ввести коректне значення, яке система збереже в хмарному сховищі OSDU.



The screenshot displays a web interface titled "РЕЗУЛЬТАТИ DQ-ВАЛІДАЦІЇ" (DQ-validation results). In the top right corner, there is a red "FAILED" status indicator. Below the title, a yellow warning box contains the message "WARNING data.Source should not be null." followed by a small "data.Source" label. To the right of this message, the text "isNotNull : null" and a red "Закрити" (Close) button are visible. Below the warning box, a larger yellow box titled "Виправлення — data.Source should not be null." (Correction — data.Source should not be null.) contains a text input field with the placeholder "data.Source" and the value "Some value". At the bottom of this box are two buttons: a green "Зберегти зміни" (Save changes) button and a grey "Скасувати" (Cancel) button.

Рисунок 5.18 — Вікно виправлення

Другий блок сторінки — це семантична валідація за допомогою штучного інтелекту. Користувач може ініціювати глибокий аналіз запису великою мовною моделлю Gemini. Якщо ШІ виявляє логічну аномалію, він генерує конкретну пропозицію щодо виправлення. Унікальність реалізації полягає в тому, що ШІ повертає точний шлях до помилкового атрибута у вкладеній JSON-структурі. Користувачеві відображається панель підтвердження, де він бачить поточне хибне значення та згенероване правильне. В один клік експерт може застосувати ці зміни, після чого застосунок виконає мутацію документа та оновить його базу даних. ШІ також може порадити видалити поле, або додати поле якого не існувало, оскільки інколи дані можуть надходити з одного джерела, але в різних форматах, й система валідації даної програми працює саме з даними які надходять з OSDU Storage API, оскільки ці дані є оригінальними (рис. 5.19).



Рисунок 5.19 — Валідація та виправлення за допомогою ІІІ

ІІІ також може порадити видалити поле, або додати поле якого не існувало. Всі результати аналізу штучним інтелектом зберігаються й можуть бути доступні зі сторінки «AI-аналіз» (рис. 5.20).

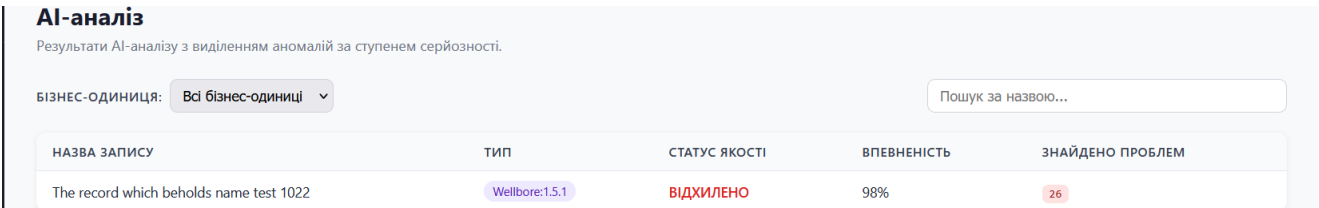


Рисунок 5.20 — Сторінка звітів ІІІ

На екрані можна побачити коротку інформацію про всі звіти аналізу штучним, існує можливість переходу до конкретної сторінки для подальшого аналізу вказівок створених ІІІ. На сторінці також доступні фільтри за бізнес-одиницею та назвою для кращої орієнтації.

5.13 Візуалізація зв’язків OSDU

Оскільки метадані OSDU мають складну реляційну природу (наприклад, стовбур свердловини завжди посилається на базову свердловину, а сейсмічний набір

посилається на проєкт обробки), користувачам вкрай важко орієнтуватися у цих зв'язках через звичайні текстові JSON-масиви. Для розв'язання цієї проблеми в модуль деталізації запису інтегровано інтерактивний граф зв'язків (рис. 5.21).

Граф реалізований на базі спеціалізованої бібліотеки reactflow. Він використовує радіальну стратегію компонування: поточний запис завжди відображається в центрі як «кореневий» вузол, а всі пов'язані з ним сутності рівномірно розподіляються по колу навколо нього. Кожен вузол має унікальне кольорове кодування залежно від типу об'єкта (наприклад, Well відображається зеленим). Інтерфейс підтримує масштабування, вільне переміщення елементів та містить міні-карту графа для зручної навігації при великій кількості зв'язків.

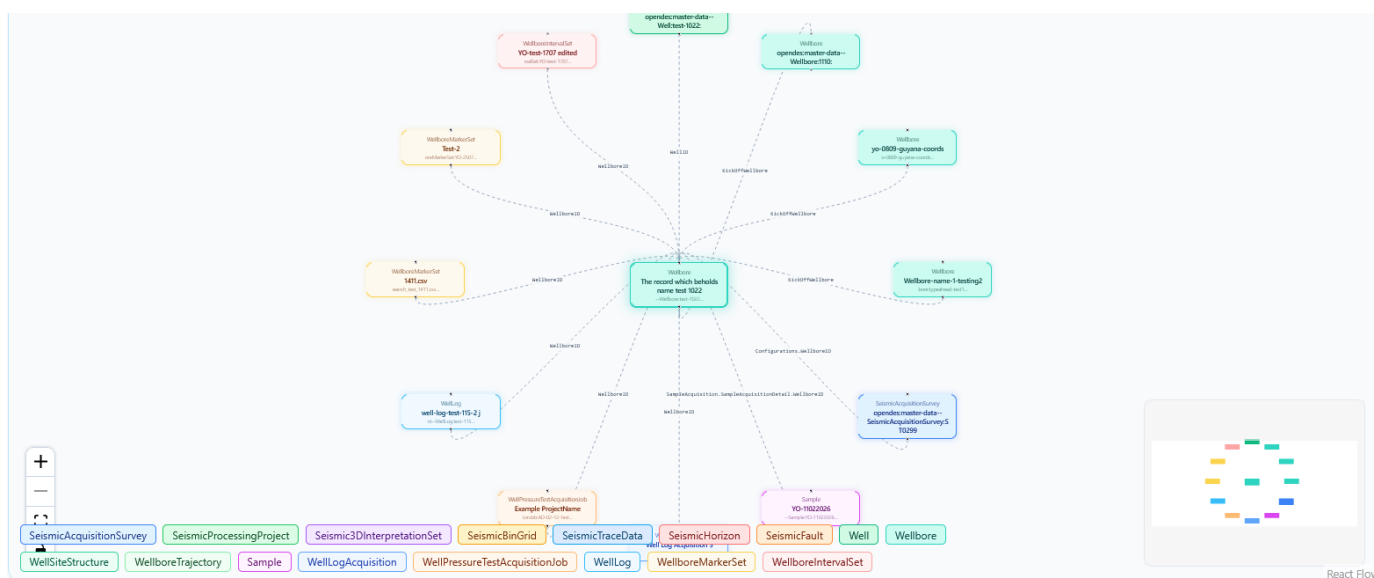


Рисунок 5.21 — Граф зв'язків геопросторових даних

Найважливішою архітектурною особливістю цього модуля є підтримка двонаправленого (bidirectional) розрізнення зв'язків. Коли користувач відкриває граф, фронтенд робить запит до бекенду. Прямі посилання знаходяться миттєво шляхом читання поточного документа і одразу відмальовуються на екрані. Однак пошук зворотних посилань, тобто пошук усіх сторонніх записів у всій базі OSDU, які посилаються на наш об'єкт є дуже ресурсоємним.

Щоб не блокувати інтерфейс, бекенд ініціює для зворотних посилань асинхронну фонову задачу. Фронтенд-компонент починає періодично опитувати статус цієї задачі (технологія Long Polling). Коли бекенд завершує пошук, React-компонент «на льоту» зливає нові вузли з уже існуючими на екрані, перераховує радіус графа і плавно додає нові елементи без перезавантаження сторінки.

Клік по будь-якому сторонньому вузлу графа автоматично перенаправляє користувача на сторінку деталізації цього об'єкта, що робить граф ідеальним інструментом для розслідування інцидентів із якістю даних.

5.14 Інтерфейс профілю користувача

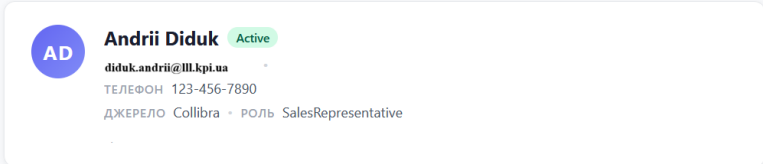
Для забезпечення високого рівня прозорості, безпеки та повноцінного контролю над власними персональними даними в системі передбачено спеціалізовану сторінку «Профіль користувача», яка слугує централізованим хабом управління обліковими записами.

Безпосередньо після успішного проходження процедури автентифікації користувача через корпоративний провайдер Azure AD, клієнтська частина (фронтенд) ініціює асинхронний запит до бекенд-сервісів. Метою цього запиту є отримання деталізованого та розширеного об'єкта профілю, що витягується в режимі реального часу безпосередньо зі сховища OSDU, забезпечуючи тим самим актуальність відображуваної інформації.

Сам користувацький інтерфейс сторінки профілю ретельно спроектований з урахуванням принципів ергономіки та логічно розділений на дві функціональні секції для зручності сприйняття. Верхня частина сторінки являє собою комплексну інформаційну картку користувача, яка візуально структурує ключові відомості: тут розташовано динамічно згенерований аватар, що полегшує ідентифікацію, а також повне ім'я, актуальну робочу електронну адресу, офіційну назву компанії-роботодавця та прямий контактний телефон. Для швидкої візуальної оцінки рівня повноважень, безпосередньо поруч із ім'ям користувача, передбачено розміщення

яскравого, легко помітного бейджа, який у режимі реального часу інформує про поточний статус доступу в системі (рис. 5.22).

Профіль користувача



Andrii Diduk Active
diduk.andrii@ll.kpi.ua
ТЕЛЕФОН 123-456-7890
ДЖЕРЕЛО Collibra · РОЛЬ SalesRepresentative

РОЛІ УПРАВЛІННЯ ДАНИМИ

ROLE	ORGANISATION	EFFECTIVE DATE	DEACTIVATION DATE
GeneralManager	UkrNafta	Dec 18, 2000	Nov 24, 2040
DataCustodian	Ukrnafta	Dec 18, 2000	Nov 24, 2040
EndUser	UkrNafta	Dec 18, 2000	Nov 24, 2040
LegalDataOwner	UkrNafta	Dec 18, 2000	Nov 24, 2040
DataCustodian	Schlumberger	Dec 18, 2000	Nov 24, 2040
DataProvider	Schlumberger	Dec 18, 2000	Nov 24, 2040
DataSteward	Schlumberger	Dec 18, 2000	Nov 24, 2040
SME	Schlumberger	Dec 18, 2000	Nov 24, 2040

Рисунок 5.22 — Профіль користувача

Нижня секція відведена під таблицю «Ролі управління даними». Оскільки OSDU оперує складною матрицею доступів, ця таблиця показує користувачеві його рівень повноважень. Кожен рядок таблиці містить ідентифікатор ролі, прив’язку до конкретної організації та життєвий цикл ролі, дати початку та автоматичної деактивації повноважень.

5.15 Управління користувачами та ролями

Оскільки система спроектована для корпоративного використання, управління доступом десятків співробітників є критичною задачею. Для цього розроблено адміністративний модуль «Користувачі», доступ до функціонала якого строго

регламентовано: змінювати дані можуть лише користувачі, які мають підтверджену роль DataSteward (рис. 5.23). Якщо рядовий користувач відкриє цю сторінку, фронтенд-компонент RoleGuard переведе інтерфейс у режим лише для читання і виведе відповідне попередження.

Головний екран модуля представляє собою таблицю-реєстр усіх зареєстрованих у системі профілів. Для зручної навігації адміністраторів реалізовано текстовий пошук у реальному часі за іменами та електронними адресами співробітників. Таблиця показує базові дані: ім'я, компанію, статус активності та загальну кількість призначених ролей.

Управління користувачами

Перегляд та управління записами OSDU UserProfile. Доступно лише для ролі DataSteward.

36 користувачів

NAME	EMAIL	COMPANY	STATUS	ROLES
T test IO Communication opendes:master-data--UserProfile:...	testIOCommunication@epam...	A.G. van den Bosch B.V.	Активна	1 роль
A Auto EPM-CCF TestUser4 opendes:master-data--UserProfile:...	Auto_EPM-CCF_TestUser4@e...	ExxonMobil	Активна	2 ролей

Рисунок 5.23 — Сторінка управління користувачами

Для користувачів із правами DataSteward навпроти кожного запису доступна кнопка редагування. Вона викликає модальне вікно (рис. 5.24), яке дозволяє керувати життєвим циклом співробітника в OSDU. Адміністратор може перевести користувача в неактивний стан, вказати точну дату деактивації, а також керувати його правами доступу.

Редагування масивів «Governance Roles» та «Persona Types» реалізовано через вбудований JSON-редактор, що забезпечує максимальну гнучкість при призначенні специфічних OSDU-політик безпеки.

Редаргувати — test IO Configuration

Можна змінити лише ці чотири поля.

Статус активності

Активна

Дата/час деактивації (ISO 8601, залиште порожнім для збереження поточного)

e.g. 2040-11-24T00:00:00.00+0000

Ролі управління даними (масив JSON)

[
 {
 "DataGovernanceRoleEffectiveDateTime": "2025-12-15T11:37:12.969Z",
 "DataGovernanceRoleIdentifier": "019b21cc-de09-767b-b4b1-93faf5cc765f",
 "DataGovernanceRoleTypeID": "opendes:reference-data--DataGovernanceRoleType:LegalDataOwner:",
 "OrganisationID": "opendes:master-data--Organisation:A%20acc%20ac%20usr%20den%20rech%20R_V..."
 }
]

ID типів персон робочих процесів (масив рядків JSON)

[]

Скасувати

Зберегти зміни

Рисунок 5.24 — Вікно редагування користувача

При збереженні система валідує введений JSON і відправляє API-запит для точкової мутації профілю у хмарі.

ВИСНОВКИ

У даній роботі було успішно створено актуальну науково-практичну систему, розроблено для автоматизації контролю та підвищення якості геопросторових даних на базі галузевого стандарту OSDU (Open Group).

Проведено детальний аналіз предметної області OSDU та виконано порівняльний аналіз наявних систем контролю якості (OSDU Data Quality Services, Great Expectations та Schlumberger Delfi). З'ясовано, що існуючі рішення орієнтовані виключно на жорстку синтаксичну перевірку схем документів, через що вони неспроможні виявляти логічні та контекстні суперечності у вкладених структурах даних. Це обґрунтувало необхідність створення нової інтелектуальної системи.

Обґрунтовано вибір технологічного стеку системи (FastAPI, Redis, Celery, React, Docker). З'ясовано, що застосування NoSQL СКБД Redis у поєднанні з об'єктно-документним проектуванням дозволяє забезпечити необхідну гнучкість збереження правил валідації без жорсткої нормалізації схем, а використання асинхронної черги Celery мінімізує навантаження на систему під час обробки великих масивів промислових метаданих.

Спроектовано та розроблено серверну частину системи, де реалізовано двохетапний валідаційний конвеєр: миттєву перевірку хронологічних і математичних аномалій засобами швидких правил Redis та семантичний аудит контексту за допомогою інтегрованого API великої мовної моделі Gemini. Реалізовано надійний модуль безпеки на основі протоколу OAuth2 та сервісів Azure Active Directory із використанням оптимізованого TTL-кешування прав, що дозволило розмежовувати ролі (RBAC) без створення зайвих запитів до платформи OSDU.

Створено клієнтську частину вебзастосунку на базі React та TypeScript. Завдяки інтеграції картографічної бібліотеки Leaflet реалізовано наочну візуалізацію геопросторових об'єктів (свердловин та сейсмічних профілів), а впровадження інструменту React Flow дозволило розробити інтерактивний граф взаємозв'язків сутностей OSDU, який спрощує для експертів пошук і аналіз топологічних помилок у реляційних залежностях.

У процесі розробки було опрацьовано методологію проєктування складних корпоративних систем. А інтеграція AI-сервісів дозволила опанувати передові підходи до вирішення логічних задач, суттєво розширивши розуміння можливостей штучного інтелекту в сучасних промислових розробках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Home page. *The Open Group OSDU® Forum*. URL: <https://osduforum.org> (date of access: 23.11.2025).
2. Redis Documentation. *Redis*. URL: <https://redis.io/docs/> (date of access 20.11.2025).
3. The JavaScript Object Notation (JSON) Data Interchange Format. *Internet Engineering Task Force (IETF)*. URL: <https://datatracker.ietf.org/doc/html/rfc8259> (date of access 10.04.26).
4. Newman S. Building Microservices: Designing Fine-Grained Systems : навч. посіб. Вид. 2-ге, Sebastopol, USA : O'Reilly Media, 2021, 612 p.
5. Bass L., Clements P., Kazman R. Software Architecture in Practice : навч. посіб. Вид. 4-те. Boston, USA : Addison-Wesley Professional, 2021, 448 p.
6. Sommerville I. Software Engineering : навч. посіб. Вид. 10-те. London, UK : Pearson, 2015, 816 p.
7. FastAPI: Modern, fast (high-performance), web framework for building APIs with Python 3.8+. *FastAPI*. URL: <https://fastapi.tiangolo.com/> (date of access: 20.11.2025).
8. Celery — Distributed Task Queue Documentation. *Celery Project*. URL: <https://docs.celeryq.dev/> (date of access: 20.11.2025).
9. React: The library for web and native user interfaces. *Meta Platforms*. URL: <https://react.dev/> (date of access: 03.12.2025).
10. TypeScript Documentation: Typed JavaScript at Any Scale. *Microsoft*. URL: <https://www.typescriptlang.org/docs/> (date of access: 03.12.2025).
11. A composable charting library built on React components. *Recharts*. URL: <https://recharts.org/> (date of access: 22.04.2026).
12. Docker Compose Overview and Documentation. *Docker Inc.* URL: <https://docs.docker.com/compose/> (date of access: 20.11.2025).
13. Microsoft Authentication Library (MSAL) for JS Documentation. *Microsoft Azure*. URL: <https://learn.microsoft.com/en-us/entra/identity-platform/msal-overview> (date of access: 10.04.2026).

14. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems : навч. посіб. Sebastopol, USA : O'Reilly Media, 2017, 616 p.
15. OpenAPI Specification v3.1.1. *OpenAPI Initiative*. URL: <https://swagger.io/specification/> (date of access: 02.04.2026).
16. Professional Petroleum Data Management. *PPDM Association*. URL: <https://ppdm.org> (date of access: 25.11.2025).
17. Redman T. Data Quality: The Field Guide : навч. посіб. Boston, USA : Digital Press, 2001, 256 p.
18. Batini C., Scannapieco M. Data Quality: Concepts, Methodologies and Techniques : навч. посіб. Berlin, Germany : Springer, 2006, 262 p.
19. Gehrke J. et al. Data Quality Management in the Era of Big Data. *IEEE Data Engineering Bulletin* : наукова стаття, 2020, 20 p.
20. Vaswani A. et al. Attention Is All You Need. *Advances in Neural Information Processing Systems 30 (NIPS 2017)* : materials of the International Conference. Long Beach, CA, 2017. P. 5998–6008.
21. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint* : наукова стаття, 2018, 16 p. URL: <https://arxiv.org/abs/1810.04805> (date of access: 10.04.2026).
22. Neepa K. Innovations in Data Quality Management: Lessons from the Oil & Gas Industry. *International journal of research and analytical reviews*. 2018, 6 p.
23. Python 3.11 Documentation. *Python Software Foundation*. URL: <https://docs.python.org/3/> (date of access: 20.11.2025).
24. The OAuth 2.0 Authorization Framework. RFC 6749. *Internet Engineering Task Force (IETF)*. URL: <https://datatracker.ietf.org/doc/html/rfc6749> (date of access: 10.04.2026).
25. Fowler M. Patterns of Enterprise Application Architecture : навч. посіб. Boston, USA : Addison-Wesley Professional, 2002, 560 p.
26. Pydantic: Data validation and settings management using Python type hints. *Pydantic*. URL: <https://docs.pydantic.dev/> (date of access: 26.11.2025).

27. Declarative Routing for React.js. *React Router*. URL: <https://reactrouter.com/> (date of access: 03.12.2025).
28. NGINX Reverse Proxy and Load Balancer Documentation. *Nginx*. URL: <https://nginx.org/en/docs/> (date of access: 28.11.2025).
29. Vite: Next Generation Frontend Tooling. *Vite*. URL: <https://vitejs.dev/guide/> (date of access: 03.12.2025).
30. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software : навч. посіб. Boston, USA : Addison-Wesley Professional, 2003, 560 p.

ДОДАТОК А

Інтелектуальна система контролю якості даних з платформи OSDU

Програмна реалізація

НТУУ «КПІ ім. Ігоря Сікорського»

Аркушів 4

Київ — 2026

Програмні засоби:

- мова програмування серверної частини – Python
- фреймворк серверної частини - FastAPI
- мови стилів та розмітки – CSS, HTML
- мова програмування клієнт частини – TypeScript
- фреймворк клієнт частини - React

```

Файл app/main.py
// Ініціалізація API-шлюзу, підключення Redis та маршрутизація (FastAPI)
import logging
import time
from contextlib import asynccontextmanager
from typing import AsyncIterator
import redis.asyncio as aioredis
from redis_om import Migrator as AsyncMigrator
from fastapi import FastAPI
from redis.exceptions import BusyLoadingError
from redis_om import get_redis_connection
from starlette.middleware.cors import CORSMiddleware
from core.config import app_settings
from db_models.models import DataQualityRule, ValidationReport
_logger = logging.getLogger(__name__)
_REDIS_READY_RETRIES = 10
_REDIS_READY_INTERVAL = 2 # seconds
def _wait_for_redis(url: str) -> None:
    """Block until Redis finishes loading its dataset, or raise after retries."""
    client = get_redis_connection(url=url, decode_responses=True)
    for attempt in range(1, _REDIS_READY_RETRIES + 1):
        try:
            client.ping()
            return
        except BusyLoadingError:
            _logger.warning("Redis is loading dataset, waiting... (%d/%d)", attempt, _REDIS_READY_RETRIES)
            time.sleep(_REDIS_READY_INTERVAL)
    raise RuntimeError("Redis did not finish loading after retries")
@asynccontextmanager
async def lifespan(app: FastAPI) -> AsyncIterator[None]:
    # Wait for Redis to be fully ready
    _wait_for_redis(app_settings.REDIS_DATA_URL)
    _logger.info("Redis is reachable")
    sync_redis = get_redis_connection(url=app_settings.REDIS_DATA_URL, decode_responses=True)
    # Initialize atomic counters if they don't exist
    if not sync_redis.exists("data_quality_rule_id_counter"):
        sync_redis.set("data_quality_rule_id_counter", 0)
    # Wire async connection to the model so Migrator uses the right host
    async_redis = aioredis.from_url(app_settings.REDIS_DATA_URL, decode_responses=True)
    DataQualityRule.Meta.database = async_redis
    ValidationReport.Meta.database = async_redis
    await AsyncMigrator().run()
    _logger.info("Redis indexes migrated")
    yield
    await async_redis.aclose()
    _logger.info("Shutting down")
app = FastAPI(lifespan=lifespan, **app_settings.fastapi_kwargs)

app.add_middleware(
    CORSMiddleware,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    allow_origins=["*"],
)
# Router integration
from api.router import router as utils_router
app.include_router(utils_router, prefix="/api/v1")
if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="127.0.0.1", port=8020, reload=True)

```

```

Файл app/ai_integration/validator.py
// Модуль семантичної валідації геоданих за допомогою Gemini AI
import hashlib
import json
from datetime import date
from typing import Any
import redis.asyncio as aioredis
from google import genai
from google.genai.types import GenerateContentConfig
from pydantic import BaseModel

SYSTEM_PROMPT = """You are an automated, high-precision Data Quality Assessment engine for OSDU geoscience records. Your primary function is to
evaluate a provided JSON Data Record against its corresponding JSON Schema to identify semantic anomalies, logical inconsistencies, and business-rule
violations.
Evaluation Criteria:
1. Compare the Data Record against the Schema constraints.
2. Perform deep semantic validation specific to the oil & gas domain. Actively look for:
2.1 Physical impossibilities (e.g., measured depth exceeding total well depth).
2.2 Chronological errors (e.g., spud dates occurring after completion dates).
2.3 Placeholder, incomplete, or lazy data (e.g., "TBD", "N/A").
Respond strictly with a single, valid JSON object detailing 'is_valid', 'confidence_score', and an 'issues' array containing 'field', 'issue_type', 'advice', and
'suggested_value'."""

class AllIssue(BaseModel):
    field: str
    issue_type: str = ""
    advice: str
    has_suggested_value: bool = False
    suggested_value: Any = None

class AIValidationResult(BaseModel):
    is_valid: bool
    confidence_score: int
    issues: list[AllIssue]
    dq_status: str

class AIV validator:
    def __init__(self, redis_url: str, api_key: str, model_name: str | None = None) -> None:
        self.redis_url = redis_url
        self.api_key = api_key
        self.model_name = model_name
    def _redis(self) -> aioredis.Redis:
        return aioredis.from_url(self.redis_url, decode_responses=True)
    async def validate_record(self, record_id: str, kind: str) -> AIValidationResult:
        r = self._redis()
        try:
            record = await r.json().get(f'json#storage:{record_id}')
            schema = await r.json().get(f'json#schema:{kind}')
            # Formatting prompt
            user_prompt = f'JSON Schema:\n {json.dumps(schema)}\n\nRecord to validate:\n {json.dumps(record)}'
            client = genai.Client(api_key=self.api_key)
            response = await client.aio.models.generate_content(
                model=self.model_name,
                contents=user_prompt,
                config=GenerateContentConfig(
                    system_instruction=SYSTEM_PROMPT,
                    temperature=0.0,
                    response_mime_type="application/json",
                ),
            )
            parsed = json.loads(response.text)
            dq_status = "approved" if parsed["is_valid"] and parsed["confidence_score"] > 80 else "manual_review_required"
            # Map issues...
            issues = [AllIssue(**issue) for issue in parsed.get("issues", [])]
            result = AIValidationResult(is_valid=parsed["is_valid"], confidence_score=parsed["confidence_score"], issues=issues, dq_status=dq_status)
            # Update Record with AI metadata in Redis cache
            storage_key = f'json#storage:{record_id}'
            await r.json().set(storage_key, "$.ai_reviewed", True)
            await r.json().set(storage_key, "$.dq_status", dq_status)
            await r.json().set(storage_key, "$.ai_issues", [i.model_dump() for i in result.issues])
            return result
        finally:
            await r.aclose()

Файл app/api/auth/role_guard.py
// Підсистема безпеки Role Guard (RBAC)
import time
from fastapi import Depends, HTTPException
from api.auth.dependencies import get_current_user
from api.auth.user_management import user_manager
# Role type constants
ROLE_END_USER = "EndUser"
ROLE_DATA_STEWARD = "DataSteward"
ROLE_DATA_CUSTODIAN = "DataCustodian"
ROLE_SME = "SME"

```

```

# Permission sets
ROLES_CAN_WRITE_RULES: frozenset[str] = frozenset({ROLE_SME, ROLE_DATA_STEWARD, ROLE_DATA_CUSTODIAN})
ROLES_CAN_WRITE_RECORDS: frozenset[str] = frozenset({ROLE_DATA_STEWARD, ROLE_DATA_CUSTODIAN})
# In-memory role cache (per-worker, 5-minute TTL)
_role_cache: dict[str, tuple[frozenset[str], float]] = {}
_CACHE_TTL = 300.0 # seconds
async def _get_user_role_types(email: str) -> frozenset[str]:
    """Return the set of short role type names for the user (cached 5 min)."""
    now = time.monotonic()
    cached = _role_cache.get(email)
    if cached and cached[1] > now:
        return cached[0]
    role_types: frozenset[str] = frozenset()
    try:
        profile, _ = await user_manager.get_user_profile_by_email_address(email)
        if profile:
            raw = profile.get("data", {}).get("UserDataGovernanceRoles", [])
            role_types = frozenset(r["DataGovernanceRoleTypeID"].rstrip(".").split(":-")[-1] for r in raw)
    except Exception:
        pass # treat as no roles — read-only access
    _role_cache[email] = (role_types, now + _CACHE_TTL)
    return role_types
def require_roles(allowed: frozenset[str]):
    """Return a FastAPI Depends that enforces membership in allowed roles."""
    async def _dep(current_user: dict = Depends(get_current_user)) -> None:
        email = current_user.get("preferred_username", "")
        role_types = await _get_user_role_types(email)
        if not role_types.intersection(allowed):
            raise HTTPException(
                status_code=403,
                detail=f"Insufficient permissions. Required one of: {sorted(allowed)}"
            )
    return Depends(_dep)

```

```

Файл frontend/src/api/index.ts
// Клієнтський API-фасад та інтеграція MSAL
import { InteractionRequiredAuthError } from "@azure/msal-browser";
import { loginRequest, msalInstance } from "../auth/msalConfig";
const BASE = "/api/v1";
async function getToken(): Promise<string> {
    const accounts = msalInstance.getAllAccounts();
    if (!accounts.length) throw new Error("Not authenticated");
    try {
        const result = await msalInstance.acquireTokenSilent({
            ...loginRequest,
            account: accounts[0],
        });
        return result.accessToken;
    } catch (err) {
        if (err instanceof InteractionRequiredAuthError) {
            const result = await msalInstance.acquireTokenPopup(loginRequest);
            return result.accessToken;
        }
        throw err;
    }
}
export class PermissionError extends Error {
    constructor(detail?: string) {
        super(detail ?? "Insufficient permissions");
        this.name = "PermissionError";
    }
}
async function extractError(res: Response): Promise<never> {
    let detail: string | undefined;
    try {
        const body = await res.json();
        if (body?.detail) detail = String(body.detail);
    } catch {}
    if (res.status === 403) throw new PermissionError(detail);
    throw new Error(detail ?? `${res.status} ${res.statusText}`);
}
async function post<T>(path: string, body: unknown): Promise<T> {
    const token = await getToken();
    const res = await fetch(`${BASE}${path}`, {
        method: "POST",
        headers: {
            Authorization: `Bearer ${token}`,
            "Content-Type": "application/json",
        },
    });
}

```

```
    body: JSON.stringify(body),
  });
  if (!res.ok) await extractError(res);
  return res.json() as Promise<T>;
}
// Unified export point for UI components
export const api = {
  exploreRecords: (req: ExplorerRequest) => post<ExplorerResponse>("/explorer/records", req),
  evaluateRecord: (req: EvaluateRecordRequest) => post<RecordValidationResult>("/rules/evaluate-record", req),
  triggerAiValidation: (req: AIValidateRequest) => post<BaseResponse<AIValidateQueued>>("/ai/validate-record", req),
  // ... other endpoints
};
```