Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення на основі динамічної багатосервісної архітектурної моделі для збору та аналізу даних сейсмічної активності

Виконав: студент IV курсу, групи СПс-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Фуштор В.М.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту _____

1. Тема роботи Розробка програмного забезпечення на основі динамічної багатосервісної архітектурної моделі для збору та аналізу даних сейсмічної активності
Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « 06 » квітня 2026 року № _____
2. Термін подання студентом роботи 22.06.2026
3. Вихідні дані до роботи технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ.
1. Аналіз вимог та огляд предметної області.
2. Проєктування та реалізація.
3. Тестування та верифікація програмного забезпечення.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання ___6 квітня_____ 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «Аналіз вимог та огляд предметної області»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «Проєктування та реалізація»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «Тестування та верифікація програмного забезпечення»*	*04.05 – 17.05*	
	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	*18.05 – 24.05*	
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	
6.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
7.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
8.	*Захист кваліфікаційної роботи бакалавра*		
9.			
10.			
11.			
12.			

Студент _____ Фуштор В.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Фуштор В. М. Розробка програмного забезпечення на основі динамічної багатосервісної архітектурної моделі для збору та аналізу даних сейсмічної активності: робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 - інженерія програмного забезпечення / наук. кер. М. Р. Петрик. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 53 с. // С. - 53, табл. - 3, рис. - 24, бібліогр. – 24, слайд. – 14, додат. - 5 ,

Ключові слова: багатосервісна архітектура, Medallion Architecture, FastAPI, Dagster, DuckDB, React, сейсмічний моніторинг, ГІС, інформаційна безпека.

Метою роботи є створення програмної платформи моніторингу сейсмічної активності на основі багатосервісної архітектурної моделі.

У першому розділі описано актуальність теми, проведено аналіз аналогів, обрано стек технологій та сформульовано вимоги до системи.

У другому розділі спроєктовано архітектуру, базу даних системи, змодельовано UML-діаграми прецедентів, класів, послідовностей та описано реалізацію.

У третьому розділі описано методологію тестування, проаналізовано сценарії 31 тест та результати верифікації веб-інтерфейсу.

У четвертому розділі розглянуто важливість моделювання та прогнозування небезпечних ситуацій та значення автоматизації виробничих процесів в питаннях охорони праці.

Об’єктом дослідження є процеси автоматизованого збору, обробки та візуалізації просторово-часових сейсмічних даних.

Предметом дослідження є архітектурні моделі, конвеєри даних, алгоритми збагачення сейсмоданих та реалізація API й інтерфейсу.

Методи дослідження включають: аналіз аналогів, UML-моделювання, проєктування баз даних та автоматизоване тестування Pytest.

ABSTRACT

Fushtor V. M. Software development based on a dynamic multi-service architectural model for collection and analysis of seismic activity data : bachelor thesis : specialty 121 "Software Engineering": Ternopil Ivan Puluj National Technical University, 2026, 53p. // Pages - 53, tables - 3, figures - 24, bibliographic references - 24, presentation slides – 14, appendices -5.

Keywords: multi-service architecture, Medallion Architecture, FastAPI, Dagster, DuckDB, React, seismic monitoring, GIS, information security.

The purpose of the work is to create a seismic monitoring software platform based on a multi-service architectural model.

In the first chapter, the relevance of the topic is described, analogues are analyzed, the technology stack is justified, and requirements are formulated.

In the second chapter, the system architecture and database are designed, UML diagrams are modeled, and the software implementation is described.

In the third chapter, the testing methodology is presented, and scenarios of 47 automated tests and UI verification are analyzed.

In the fourth chapter, examines the importance of modeling and forecasting hazardous situations and the significance of automating production processes in the context of occupational safety.

The object of the study is the processes of automated collection, processing, and visualization of spatial-temporal seismic data.

The subject of the study is architectural models, data pipelines, enrichment algorithms, and REST API and UI implementation.

Research methods include: analogue analysis, UML modeling, database design, and automated testing using Pytest.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API – прикладний програмний інтерфейс.

DWH – сховище даних.

ETL – процес витягування, перетворення та завантаження даних.

GIS – геоінформаційна система.

JSON – текстовий формат обміну даними.

JWT – стандарт токенів безпеки у веб-додатках.

OLAP – технологія швидкої аналітичної обробки даних.

REST – архітектурний стиль взаємодії клієнта та сервера.

S3 – хмарна служба зберігання об'єктів.

SPA – односторінковий веб-застосунок.

SQL – мова структурованих запитів.

UML – уніфікована мова моделювання.

USGS – Геологічна служба США.

ЗМІСТ

ВСТУП..... 9
1 АНАЛІЗ ВИМОГ ТА ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ..... 10
1.1 Огляд аналогів 10
1.1.1 USGS Earthquake Hazards Program 10
1.1.2 EMSC Earthquake Monitor..... 11
1.1.3 GeoNet System..... 12
1.2 Моделювання функціональних сценаріїв використання..... 12
1.3 Огляд існуючих технологій реалізації 16
1.4 Висновки до першого розділу..... ‘17
2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ 18
2.1 Проєктування концептуальної архітектури системи 18
2.2 Організація сховища даних Medallion DWH (Bronze, Silver, Gold) . 18
2.3 Проєктування структури бази даних..... 19
2.4 Проєктування структури класів та модулів системи 20
2.5 Моніторинг конвеєрів та інтерфейс користувача платформи..... 28
2.6 Програмна реалізація та деталі коду..... 32
2.6.1 Структура каталогів та файлів проєкту 32
2.6.2 Опис логіки взаємодії підсистем та потоків виконання..... 34
2.7 Обґрунтування вибору програмних засобів реалізації..... 34
2.8 Висновки до другого розділу 36
3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.. 37
3.1 Стратегія та об’єкт тестування 37
3.2 Функціональне тестування..... 37
3.3 Модульне та інтеграційне тестування..... 39

3.4 Навантажувальне тестування веб-сервісу 43
3. 5 Тестування безпеки та стійкості до відмов 44
3.6 Аналіз покриття коду та результати верифікації 45
3.7 Висновки до третього розділу..... 45
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 47
4.1 Моделювання та прогнозування небезпечних ситуацій..... 47
4.2 Значення автоматизації виробничих процесів в питаннях охорони праці 49
4.3 Висновки до четвертого розділу..... 50
ВИСНОВКИ..... 52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 54
ДОДАТКИ..... 57
ДОДАТОК А 58
ДОДАТОК Б..... 59
ДОДАТОК В..... 60
ДОДАТОК Д 61
ДОДАТОК Е..... 62

ВСТУП

Актуальність теми дослідження. Землетруси належать до найбільш руйнівних природних явищ, прогнозування та аналіз яких потребує постійного збору геодинамічних параметрів. Зі зростанням обсягів сейсмічних даних монолітні системи стають неефективними, що зумовлює потребу у використанні багатосервісних систем та концепції Lakehouse (Medallion Architecture) для побудови конвеєрів збору та аналізу просторово-часової інформації.

Мета дослідження – підвищення ефективності збору, збереження, аналітичної обробки та візуалізації даних сейсмічної активності шляхом розробки багатосервісного програмного забезпечення з інтегрованими конвеєрами очищення та збагачення даних.

Для досягнення мети вирішено такі завдання:

1. Проаналізувати існуючі програмні рішення та джерела сейсмічних даних.
2. Проєктувати та реалізувати архітектуру системи та структуру бази даних.
3. Розробити конвеєр збору та обробки даних за методологією Medallion.
4. Створити веб-інтерфейс для візуалізації подій та сповіщень.
5. Реалізувати механізми безпеки (захист від SQL Injection, JWT-автентифікація).
6. Провести автоматизоване тестування та оцінку працездатності сервісів.

Об'єкт дослідження – процеси збору, обробки, збереження та візуалізації географічно розподілених даних сейсмічної активності.

Предмет дослідження – архітектурні моделі, конвеєри даних (Data Pipelines), алгоритми збагачення сейсмоданих та засоби реалізації асинхронного REST API.

Методи дослідження – теорія баз даних, концепція Medallion Architecture, об'єктно-орієнтоване UML-моделювання та тестування ПЗ.

Практичне значення результатів – створено відмостійку платформу автоматизованого агрегування сейсмічних даних, розрахунку ризиків та інформування користувачів, придатну для інтеграції в сучасні геоінформаційні системи.

1 АНАЛІЗ ВИМОГ ТА ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Ця робота заснована на сучасних архітектурних концепціях збору та обробки великих просторово-часових даних сейсмічної активності, що є ключовим інструментом для глобального моніторингу природних явищ. У дослідженнях геодинамічних процесів важливим є використання розподілених багатосервісних рішень, які забезпечують швидкий доступ до інформації, надійність зберігання даних та гнучкість обробки неструктурованих та напівструктурованих логів геофізичних датчиків [1].

Ця тема охоплює кілька важливих аспектів: збір первинних даних з відкритих геофізичних веб-служб, дедуплікацію та географічне збагачення записів після кожного пакетного завантаження, розрахунок інтегральних класів сейсмічного ризику та оперативне інформування користувачів.

Аналіз предметної області показав потребу розробки ефективного конвеєра обробки на базі концепції Medallion Architecture для фільтрації шумів, перевірки цілісності географічних координат та колоночного представлення результатів з метою оптимізації аналітичних OLAP-запитів.

1.1 Огляд аналогів

У ході проведеного огляду аналогів від конкурентів було виявлено системи зі схожими функціональними можливостями.

1.1.1 USGS Earthquake Hazards Program

Система, створена Геологічною службою США [2], є світовим стандартом у зборі сейсмологічної інформації. Вона надає відкритий доступ до оновлюваних у реальному часі GeoJSON потоків даних. Проте її інтерфейс орієнтований на наукових фахівців, що робить його занадто складним для пересічного користувача, а можливості тонкого налаштування індивідуальних сповіщень є обмеженими.

Ілюстрація принципу роботи системи представлена на рисунку 1.1.

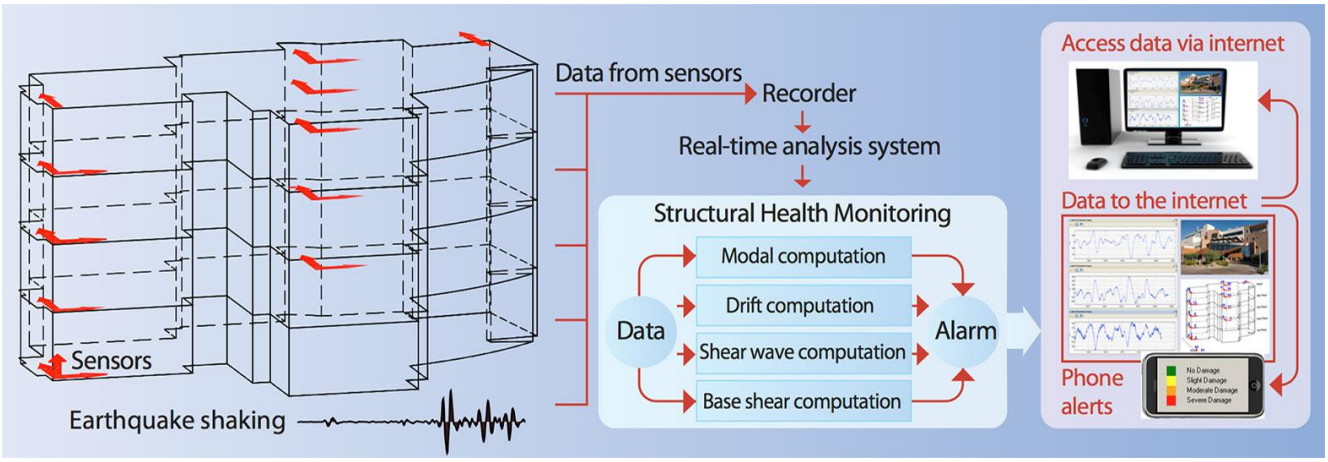


Рисунок 1.1 – Ілюстрація принципу роботи системи "USGS Hazards Program" [2]

1.1.2 EMSC Earthquake Monitor

Європейсько-Середземноморський сейсмологічний центр [3] забезпечує моніторинг сейсмічної активності в Євроазійському регіоні. Особливістю системи є швидкий збір суб'єктивних відгуків очевидців подій, але аналітична обробка історичних даних та візуалізація часових рядів мають обмежений характер.

Ілюстрація принципу роботи системи представлена на рисунку 1.2.

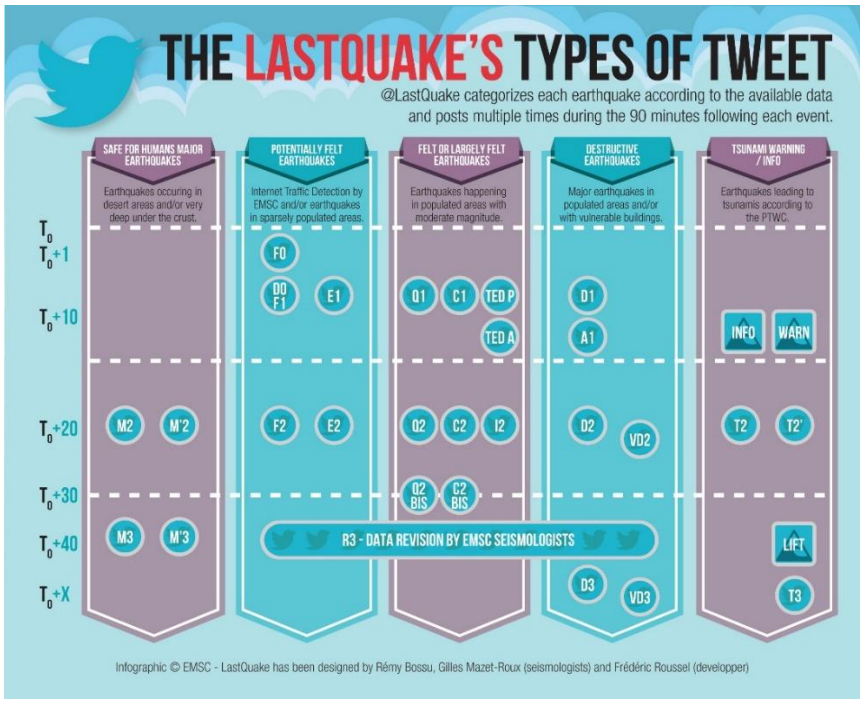


Рисунок 1.2 – Ілюстрація принципу роботи системи "EMSC Earthquake Monitor" [3]

1.1.3 GeoNet System

Новозеландська система моніторингу геодинамічних ризиків [4] поєднує високу якість локальної візуалізації та вбудовані алгоритми оцінки небезпеки. Проте її географічне охоплення є суворо регіональним, що обмежує її глобальне застосування.

1.2 Моделювання функціональних сценаріїв використання

Проектування архітектури сучасних веб-додатків вимагає чіткого розмежування прав доступу (Role-Based Access Control) та визначення доступного функціоналу для кожної групи осіб. З метою візуалізації поведінкових аспектів розроблюваного програмного продукту та відображення його основних сервісів було побудовано відповідну UML-модель. Взаємодія користувачів та адміністраторів із системою описується за допомогою діаграми прецедентів.

Взаємодія користувачів та адміністраторів із системою описується за допомогою діаграми прецедентів. У системі виділено три основні ролі.

Роль гостя відповідає неавторизованому відвідувачу сайту, який має базовий доступ до перегляду інтерактивної карти землетрусів, фільтрації сейсмічних подій та подачі запиту на відновлення паролю.

Авторизований користувач володіє розширеними правами, що дозволяють зберігати землетруси до списку вибраного та створювати індивідуальні правила сповіщень.

Адміністратор є технічним актором, який здійснює контроль за виконанням Dagster-конвеєрів, аналізує якість даних та здійснює моніторинг працездатності інфраструктури.

Діаграма варіантів використання системи представлена на рисунку 1.3.

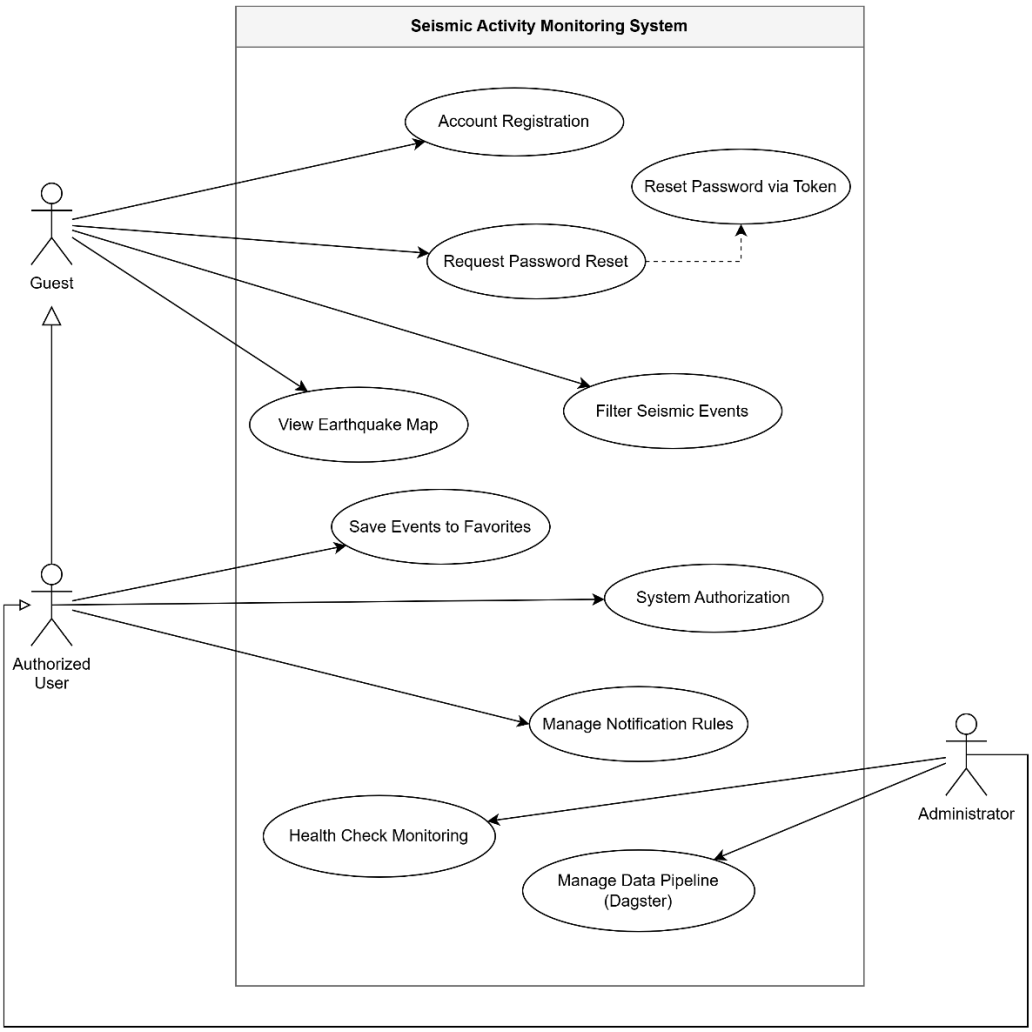


Рисунок 1.3 – Діаграма варіантів використання платформи

Забезпечення безпеки облікових записів є однією з ключових вимог до сучасних веб-додатків, що вимагає надійних механізмів управління доступом. Для детального опису роботи системи під час критичного процесу відновлення паролю розроблено поведінкову модель у вигляді діаграми послідовностей. Цей процес відображає покрокову взаємодію між клієнтським застосунком на React, асинхронним REST API на базі FastAPI та реляційною базою даних PostgreSQL, ілюструючи життєвий цикл запиту від ініціалізації до оновлення облікових даних.

Діаграма послідовностей процесу відновлення доступу представлена на рисунку 1.4.

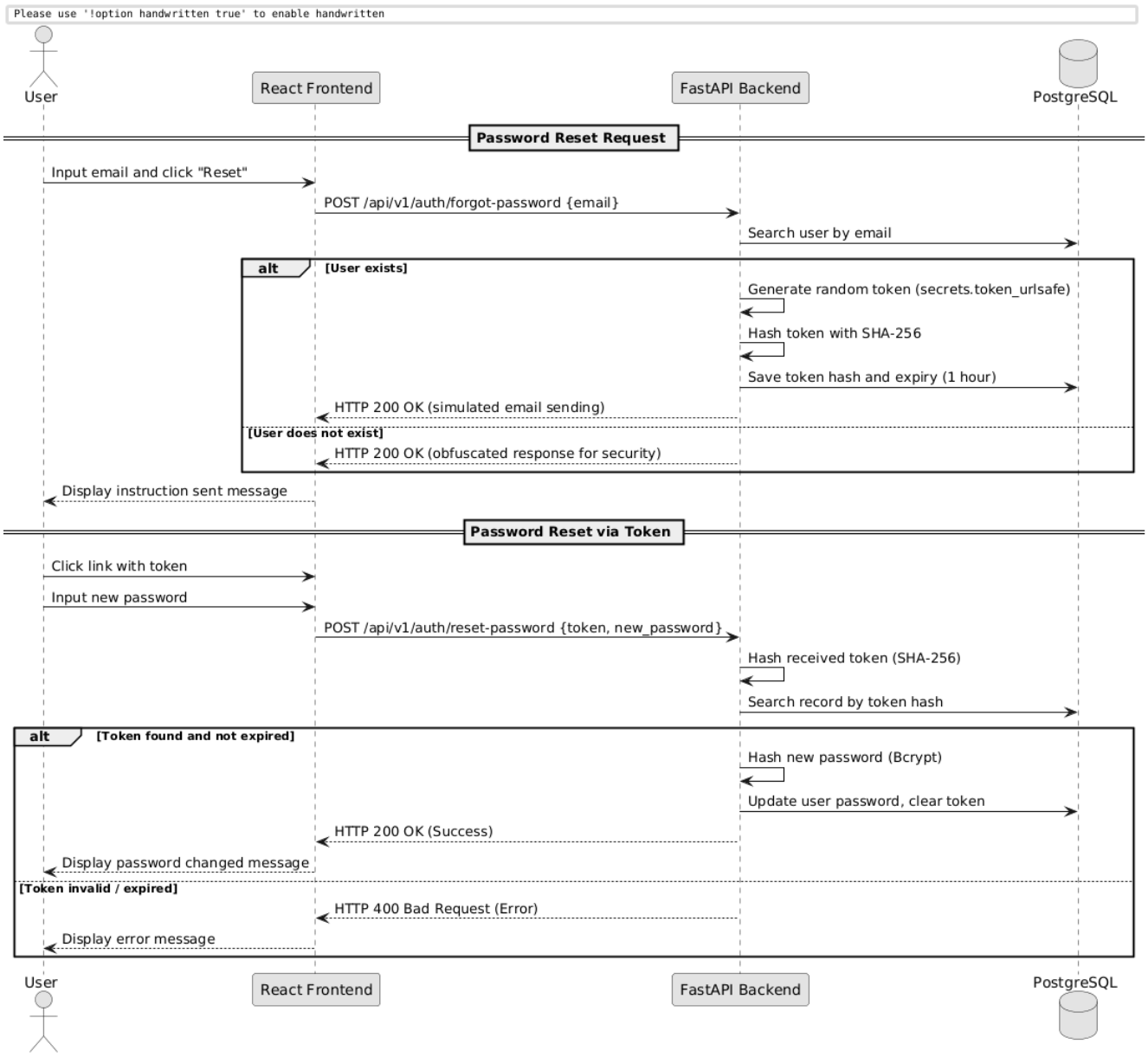


Рисунок 1.4 – Діаграма послідовностей процесу відновлення паролю

Для детального аналізу динамічної поведінки системи під час інтеграції інформаційних потоків необхідно розглянути хронологію взаємодії її ключових компонентів. З цією метою будується відповідна UML-модель, яка ілюструє життєвий цикл процесів вилучення, трансформації та завантаження інформації, а також логіку обміну повідомленнями між модулями в часі.

Діаграма послідовностей конвеєра обробки даних (ETL) представлена на рисунку 1.5.

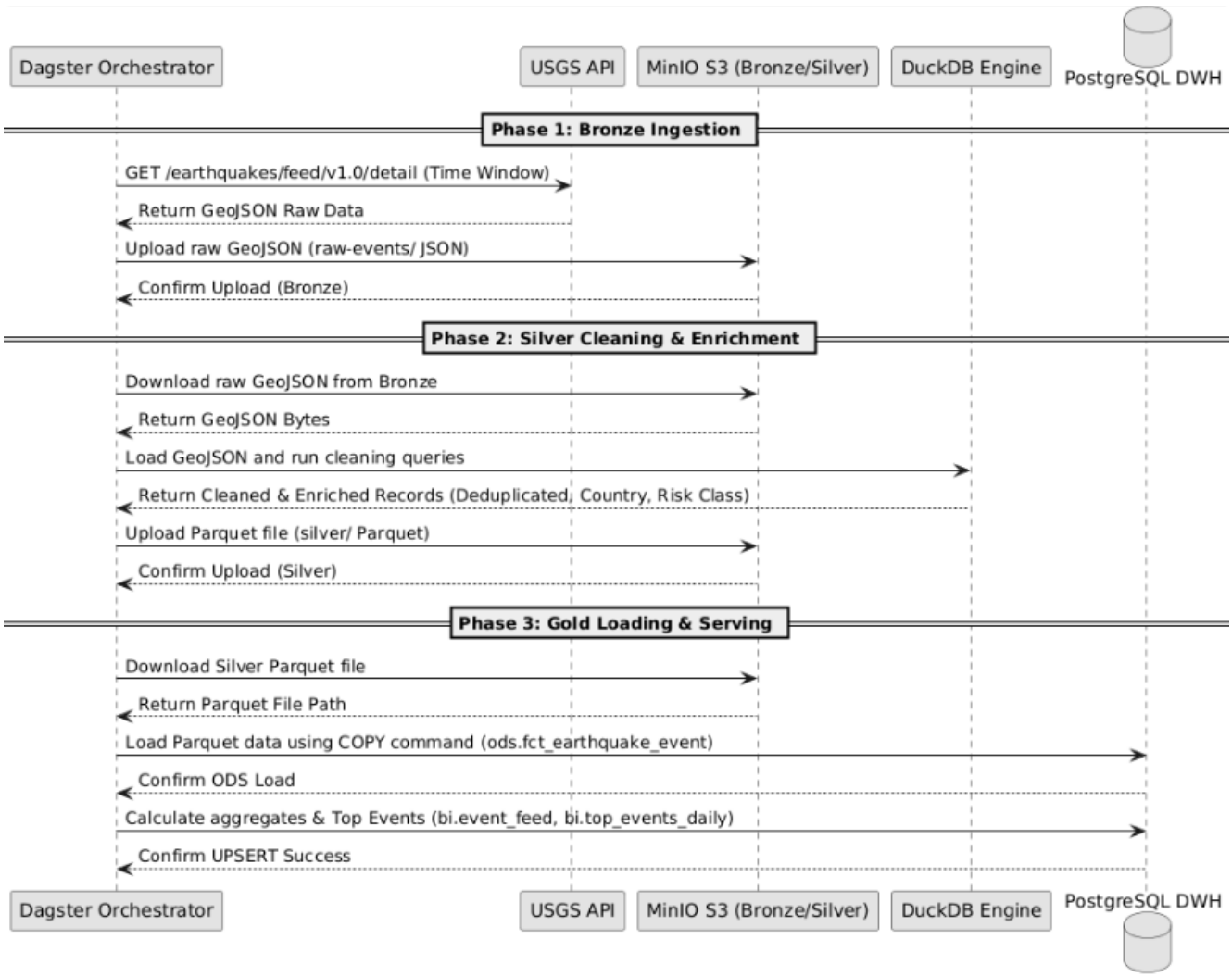


Рисунок 1.5 – Діаграма послідовностей конвеєра обробки даних (ETL)

Ефективний моніторинг стану системи та своєчасне інформування про критичні події вимагають надійного механізму налаштування тригерів. Для розуміння логіки роботи цього модуля необхідно проаналізувати хронологію викликів: від моменту конфігурації умов адміністратором до їх регулярної фонової перевірки та спрацьовування. З цією метою було розроблено відповідну поведінкову модель.

Діаграма послідовностей створення та оцінки правил сповіщень представлена на рисунку 1.6.

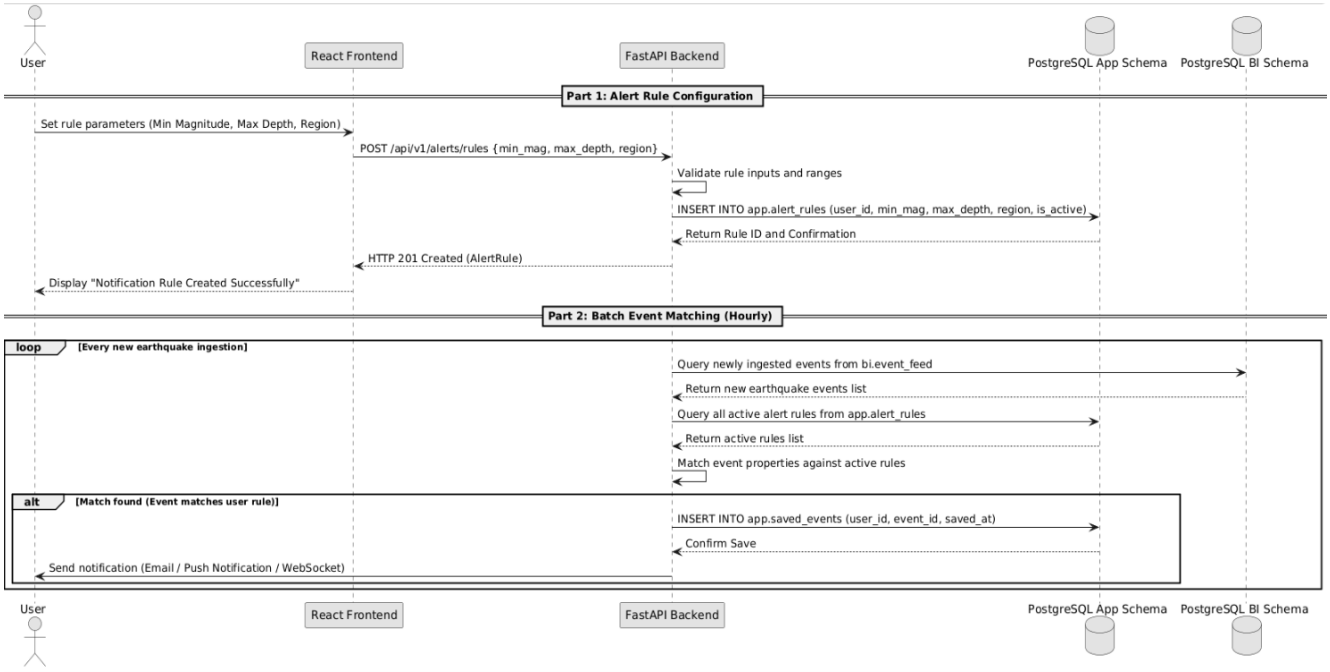


Рисунок 1.6 – Діаграма послідовностей створення та оцінки правил сповіщень

1.3 Огляд існуючих технологій реалізації

Для реалізації систем такого типу зазвичай розглядають наступні технологічні рішення:

– використання асинхронних бекенд-фреймворків для обробки великої кількості паралельних запитів клієнтів;

– застосування сучасних оркестраторів Data Pipelines для автоматизації періодичного збирання даних;

– збереження великих масивів інформації у колоночних форматах (Parquet) для прискорення OLAP-аналітики;

– використання реляційних СУБД для роботи зі структурованою інформацією користувачів;

– впровадження систем кешування та лімітування частоти запитів для захисту від перевантажень.

Розробка асинхронного REST API на FastAPI забезпечує високу швидкість обробки веб-запитів за рахунок підтримки протоколу ASGI та валідації Pydantic. На відміну від монолітних рішень на Django чи Flask, FastAPI генерує інтерактивну

OpenAPI документацію "з коробки" та демонструє близьку до Go продуктивність.

Для управління конвеєрами даних замість традиційного Apache Airflow використано Dagster, який пропонує концепцію Data Assets та дозволяє чітко розмежувати етапи Bronze, Silver та Gold обробки.

Швидке аналітичне перетворення Parquet файлів безпосередньо в оперативній пам'яті виконує вбудований OLAP-процесор DuckDB, мінімізуючи мережевий оверхед у порівнянні з PostgreSQL при складних агрегаціях.

1.4 Висновки до першого розділу

В цьому розділі було проведено детальний аналіз предметної області моніторингу та обробки сейсмологічних даних. Розглянуто існуючі глобальні та регіональні аналоги систем збору інформації, такі як USGS, EMSC та GeoNet, визначено їхні переваги та недоліки, зокрема складність інтерфейсів для кінцевого користувача та обмеженість налаштування персональних сповіщень. Обґрунтовано необхідність створення гнучкого розподіленого рішення на основі багатосервісної архітектури. Здійснено порівняльний аналіз сучасних технологій розробки, на основі якого для реалізації платформи обрано асинхронний фреймворк FastAPI, бібліотеку React для побудови SPA-клієнта, оркестратор конвеєрів даних Dagster та OLAP-процесор DuckDB. Сформульовані вимоги та обрані технологічні рішення закладають основу для проєктування архітектурних рішень та структури бази даних системи у наступному розділі.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

В даному розділі описано процес проєктування розроблюваних компонентів.

Також наведено програмні засоби та особливості реалізації компонент системи.

2.1 Проєктування концептуальної архітектури системи

При проєктуванні архітектури платформи було обрано сервісно-орієнтований підхід із чітким розділенням зон відповідальності (Separation of Concerns). Система складається з трьох автономних рівнів: конвеєра обробки великих даних (Data Pipeline), асинхронного REST API та односторінкового клієнтського веб-застосунку (SPA).

Взаємодія між компонентами відбувається через стандартизовані інтерфейси обміну даними (S3 API та REST HTTP), що мінімізує зв'язність сервісів та спрощує їх незалежне масштабування та розгортання в ізольованих контейнерах Docker.

Схема загальної архітектури розробленої багатосервісної системи представлена в додатку А.

2.2 Організація сховища даних Medallion DWH (Bronze, Silver, Gold)

Для впорядкування та підвищення швидкості аналітичної обробки вхідних сейсмічних даних використовується сучасний архітектурний підхід Lakehouse, організований за методологією Medallion. Сховище даних (DWH) розділене на три логічні шари, що дозволяє поступово трансформувати сирі потоки даних у чисту бізнес-аналітику:

– Bronze Layer (Raw Data): Зберігає незмінні сирі файли у форматі JSON, отримані безпосередньо з API USGS. Це забезпечує можливість повторного запуску аналітики у разі зміни правил обробки.

– Silver Layer (Cleaned & Enriched): Містить очищені від дублікатів дані,

переведені в ефективний колоночний формат Parquet. На цьому етапі відбувається географічне збагачення даних (координати відображаються на код країни) та автоматичний розрахунок інтегральних класів ризику землетрусу.

– Gold / Serving Layer (Aggregated Analytical tables): Шар реляційних таблиць бази даних PostgreSQL, який містить безпосередньо готові дані для швидкого відображення користувачам (наприклад, щоденні топи подій, аналітичні агрегати тощо).

Схему конвеєра обробки даних (Medallion Architecture) у сховищі DWH представлено в Додатку Б.

2.3 Проєктування структури бази даних

Для зберігання структурованої інформації застосунку та забезпечення роботи Serving-рівня аналітики використовується реляційна база даних PostgreSQL. Проєктування структури бази даних виконано у відповідності до вимог третьої нормальної форми. База даних складається з двох схем:

– "app": містить таблиці користувацької логіки, авторизаційних сесій та індивідуальних налаштувань ("users", "refresh_tokens", "alert_rules", "saved_events").

– "bi": містить таблиці представлення аналітичних зрізів для відображення на клієнті та аналізу працездатності каталогу ("event_feed", "top_events_daily", "catalog_health_daily").

– "dq": містить службові таблиці оцінки якості та валідації вхідних даних конвеєрів обробки ("dq_run", "dq_metric", "dq_issue").

– "ods": оперативна staging-таблиця для проміжного збереження оброблених записів перед їх публікацією ("fct_earthquake_event").

– "public": загальна схема для швидкого логування стану працездатності додатку ("app_heartbeat").

Схема зв'язків між таблицями бази даних представлена в Додатку В у вигляді ER-діаграми.

2.4 Проєктування структури класів та модулів системи

Для відображення об'єктно-орієнтованої структури розроблених модулів бекенду, конвеєрів збору даних та інтерфейсу побудовано загальну діаграму класів.

Діаграма класів системи представлена в Додатку Д.

Варто зазначити, що у архітектурі розробленого програмного забезпечення підсистема API активно використовує три основні сервіси-фасади для спрощення взаємодії між HTTP-контролерами та складними внутрішніми компонентами системи. Ці фасади інкапсулюють бізнес-логіку авторизації, криптографії та збереження даних, надаючи зовнішнім маршрутизаторам простий інтерфейс доступу.

Сервіс авторизації AuthService реалізує патерн проектування Фасад. Він об'єднує низькорівневі інструменти обробки паролів PasswordService та генерації токенів TokenService з транзакційним доступом до бази даних PostgreSQL. Завдяки цьому складні операції реєстрації користувачів, ротації токенів оновлення та скидання паролів приховані за простими методами. Головним споживачем цього сервісу є маршрутизатор AuthRouter, який завдяки впровадженню залежностей FastAPI делегує всю бізнес-логіку фасаду, фокусуючись виключно на обробці HTTP-запитів та відповідей.

Сервіс подій EventsServiceFacade виступає процедурно-функціональним фасадом над об'єктно-орієнтованим репозиторієм подій землетрусів EventRepository. Він надає набір функцій, які приховують деталі ініціалізації репозиторію та прямої взаємодії з базою даних, що забезпечує зворотну сумісність із маршрутизаторами. Основним споживачем цього фасаду є контролер подій EventsRouter, який використовує функції отримання списку землетрусів та розрахунку агрегованої статистики для формування відповідей клієнтському інтерфейсу.

Сервіс профілів та правил сповіщень UsersServiceFacade інтегрує логіку роботи з двома окремими сутностями — збереженими подіями користувача та тригерами сповіщень про сейсмічну активність. Він приховує за собою складну

роботу з репозиторіями SavedEventRepository та AlertRuleRepository. Маршрутизатор профілів UsersRouter є безпосереднім споживачем цього фасаду, викликаючи його методи для створення, оновлення чи видалення збережених подій та правил без необхідності безпосереднього знання про структуру сховища чи транзакційну логіку бази даних. Нижче наведено детальний опис призначення, полів та методів кожного ключового класу системи.

Абстрактний клас "BaseJob" представлено на рисунку 2.1.

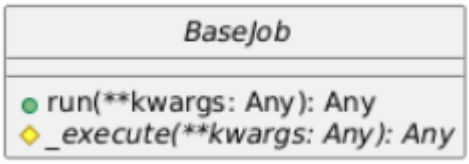


Рисунок 2.1 – Діаграма класу "BaseJob"

Абстрактний клас BaseJob виступає базовим шаблоном для всіх кроків обробки (джоб) у ETL-конвеєрі даних, реалізуючи патерн проектування Template Method (Шаблонний метод). Він визначає єдину публічну точку входу — метод run(), який приймає довільні параметри обробки у вигляді іменованих аргументів **kwargs. Цей метод делегує виконання специфічної логіки кроку внутрішньому абстрактному методу _execute(), який повинен бути перевизначений у кожному конкретному дочірньому класі. Таке розділення дозволяє інкапсулювати наскрізну логіку (наприклад, логування, моніторинг часу виконання, транзакційні обгортки) в базовому класі, не дублюючи її в реалізаціях.

Інтерфейс "ISeismicDataSource" представлено на рисунку 2.2.

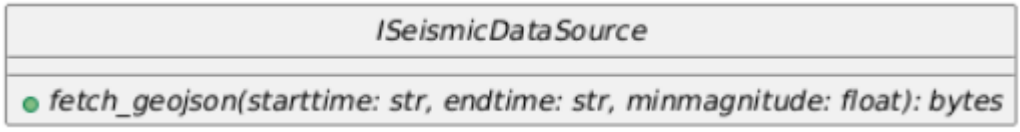


Рисунок 2.2 – Діаграма інтерфейсу "ISeismicDataSource"

Інтерфейс ISeismicDataSource є частиною реалізації патерну Strategy (Стратегія), призначеної для абстрагування джерел отримання сейсмічних даних. Головна мета інтерфейсу — ізолювати ETL-конвеєр від конкретних механізмів мережевої взаємодії та форматів сторонніх API. Він декларує один абстрактний метод fetch_geojson(), який приймає обов'язкові параметри часового вікна (starttime та endtime у форматі ISO-8601) та необов'язковий фільтр мінімальної сили землетрусу minmagnitude. Метод повертає масив сирих байтів (bytes), що містить отриману відповідь у форматі GeoJSON.

Клас "USGSClient" представлено на рисунку 2.3.

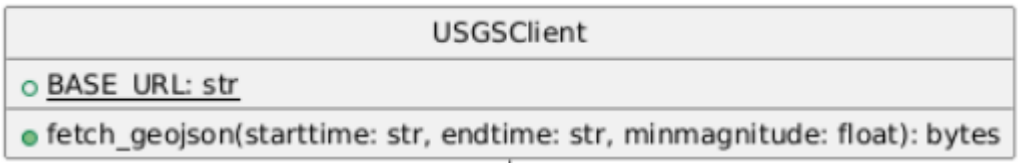


Рисунок 2.3 – Діаграма класу "USGSClient"

Клас USGSClient є конкретною реалізацією стратегії отримання даних ISeismicDataSource для взаємодії з REST API сейсмологічної служби США (USGS). У класі визначено статичну константу BASE_URL, яка зберігає адресу endpoint-запиту до USGS FDSN Web Service. Метод fetch_geojson() реалізує HTTP-запит методом GET з необхідними параметрами фільтрації та сортування за часом, використовуючи бібліотеку requests. Метод обробляє можливі помилки мережевої передачі через виклик raise_for_status() та повертає тіло відповіді у вигляді байтового рядка.

Інтерфейс "IEnricher" представлено на рисунку 2.4.

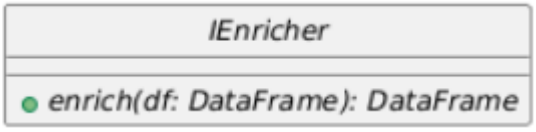


Рисунок 2.4 – Діаграма інтерфейсу "IEnricher"

Інтерфейс IEnricher визначає контракт для компонентів збагачення даних. Він декларує абстрактний метод enrich(), який приймає на вхід об'єкт структурованої таблиці DataFrame бібліотеки pandas та повертає новий модифікований DataFrame із додатковими розрахованими властивостями. Цей інтерфейс виступає базовим для побудови гнучких стратегій обробки даних (наприклад, геокодування чи класифікації ризиків) та підтримує концепцію незмінності (immutability) вхідних даних під час ETL-процесу.

Клас "CompositeEnricher" представлено на рисунку 2.5.

Рисунок 2.5 – Діаграма класу "CompositeEnricher"

Клас CompositeEnricher реалізує патерн проектування Composite (Компонувальник). Його призначення — об'єднувати декілька окремих кроків збагачення даних в один логічний пайплайн. Клас містить приватний список _enrichers типу list[IEnricher], який ініціалізується через конструктор __init__(). Метод enrich() послідовно викликає метод enrich() кожного зареєстрованого компонента, передаючи результат попереднього кроку наступному за принципом конвеєра (pipeline chain).

Інтерфейс "ObjectStorage" представлено на рисунку 2.6.

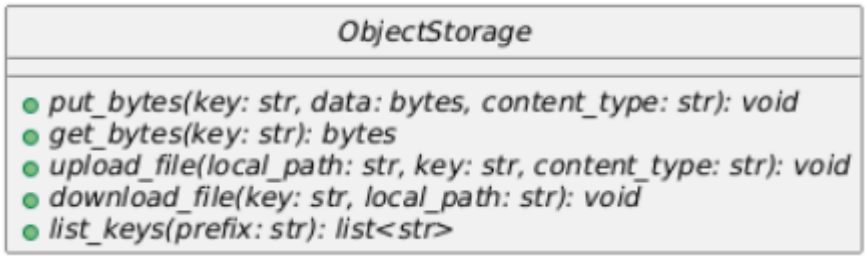


Рисунок 2.6 – Діаграма інтерфейсу "ObjectStorage"

Інтерфейс ObjectStorage описує контракт для взаємодії зі сховищем неструктурованих файлів (об'єктним сховищем). Він надає абстракцію над файловими операціями в хмарі (наприклад, AWS S3 чи MinIO). Визначено методи: put_bytes() для прямого запису байтів під певним ключем, get_bytes() для зчитування даних, upload_file() та download_file() для роботи з локальною файловою системою, а також list_keys() для отримання списку наявних файлів за префіксом (шляхом).

Клас "S3Storage" представлено на рисунку 2.7.

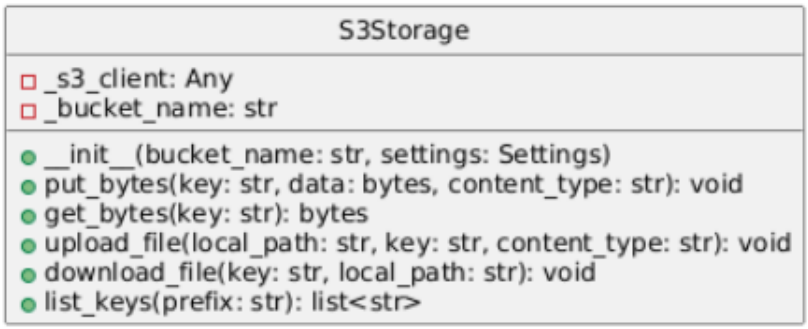


Рисунок 2.7 – Діаграма класу "S3Storage"

Клас S3Storage реалізує інтерфейс ObjectStorage для роботи з хмарами S3-сумісного типу. Він інкапсулює роботу з бібліотекою boto3. У конструкторі приймає параметри підключення (endpoint URL, ключі доступу, назву бакета). Методи put_bytes(), get_bytes(), upload_file(), download_file() та list_keys() транслюють виклики системи у відповідні API-запити до S3 клієнта boto3, обробляючи винятки відсутності доступу чи файлів.

Інтерфейси сховища представлено на рисунку 2.8.

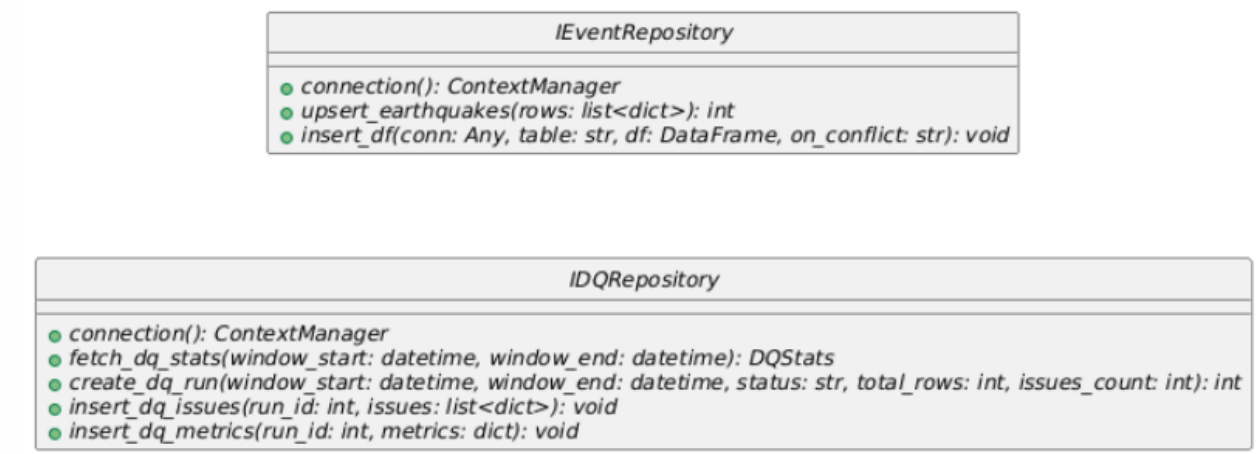


Рисунок 2.8 – Діаграма інтерфейсів сховищ

Дані інтерфейси реалізують патерн Repository для розмежування логіки доступу до бази даних та бізнес-логіки системи.

IEventRepository відповідає за збереження подій землетрусів. Він містить метод connection(), який є контекстним менеджером транзакцій бази даних, метод upsert_earthquakes() для оновлення чи створення записів подій (ODS шар) та insert_df() для масового копіювання даних із структури DataFrame.

IDQRepository відповідає за метрики якості даних. Окрім контекстного менеджера connection(), він декларує методи: fetch_dq_stats() для агрегації показників роботи за період, create_dq_run() для реєстрації запуску сесії валідації якості, а також insert_dq_issues() та insert_dq_metrics() для збереження знайдених аномалій та статистичних показників валідації.

Клас "PostgresRepository" представлено на рисунку 2.9.

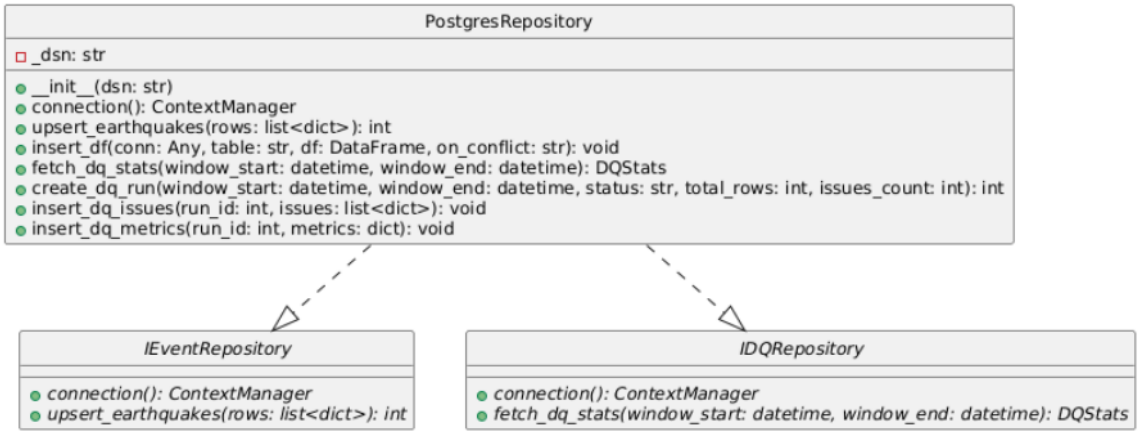


Рисунок 2.9 – Діаграма класу "PostgresRepository"

Клас PostgresRepository об'єднує в собі реалізацію інтерфейсів IEventRepository та IDQRepository для роботи з СУБД PostgreSQL (реалізація зворотної сумісності). Він використовує з'єднання бібліотеки psycopg2 для виконання SQL-запитів. Контекст-менеджер connection() керує початком транзакції, здійснюючи автоматичний commit() у разі успіху або rollback() при виникненні виняткових ситуацій. Метод upsert_earthquakes() виконує груповий запит INSERT ... ON CONFLICT DO UPDATE для запобігання дублювання подій землетрусів за унікальним ідентифікатором події.

Перевірки якості даних представлено на рисунку 2.10.

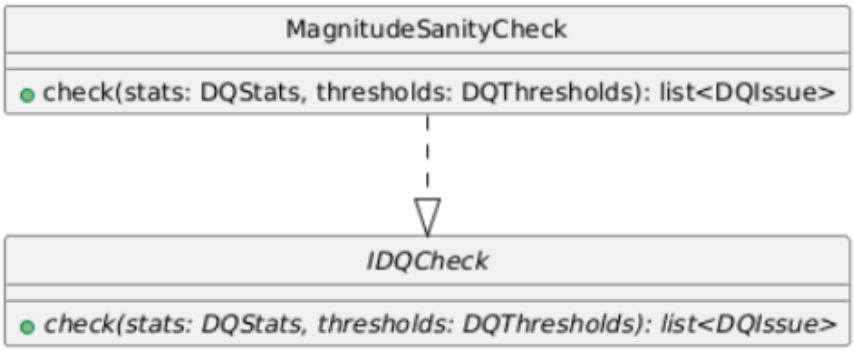


Рисунок 2.10 – Діаграма перевірок якості даних

Валідація даних побудована на базі патернів Strategy і Chain of Responsibility.

Інтерфейс IDQCheck визначає загальний контракт для кожної ізольованої бізнес-помилки чи аномалії. Він містить метод check(), який отримує об'єкт статистики DQStats та налаштування порогів DQThresholds, і повертає список зафіксованих інцидентів list[DQIssue].

Клас MagnitudeSanityCheck є конкретною реалізацією правила перевірки фізичної коректності сили землетрусу. Його метод check() аналізує максимальну та мінімальну магнітуду у вибірці. Якщо максимальна магнітуда перевищує задану в налаштуваннях константу (max_valid_mag), генерується помилка критичного рівня ERROR. Якщо мінімальна магнітуда є меншою за min_valid_mag, формується попередження WARN.

Клас "AuthService" представлено на рисунку 2.11.

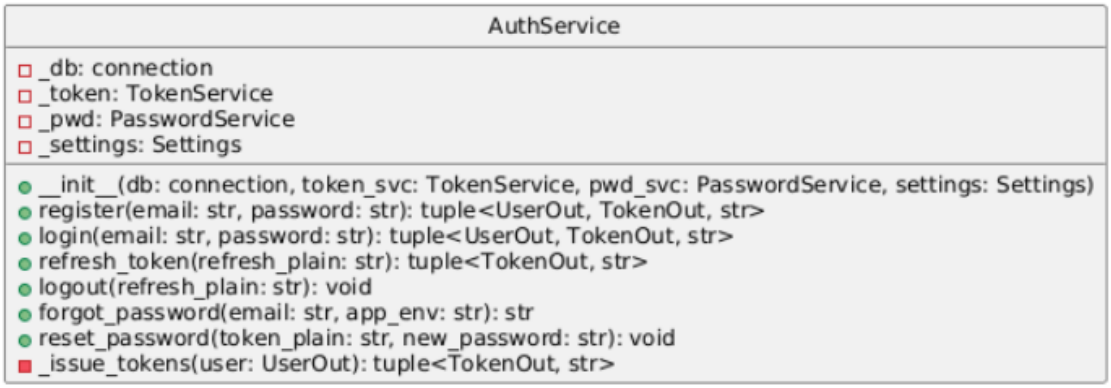


Рисунок 2.11 – Діаграма класу "AuthService"

Клас AuthService реалізує патерн проектування Facade (Фасад). Він надає спрощений інтерфейс високого рівня для роботи з аутентифікацією та сесіями користувачів, об'єднуючи під собою роботу з базою даних, обчислення хешів паролів через PasswordService та генерацію JWT-токенів через TokenService. Метод register() створює новий запис користувача у таблиці app.users та повертає JWT-токени. Метод login() звіряє введений пароль з хешем у базі даних, а метод refresh_token() реалізує безпечну ротацію (зміну) токенів оновлення (refresh token rotation) з фіксацією дати відкликання попередніх токенів.

Клас "AuthRouter" представлено на рисунку 2.12.

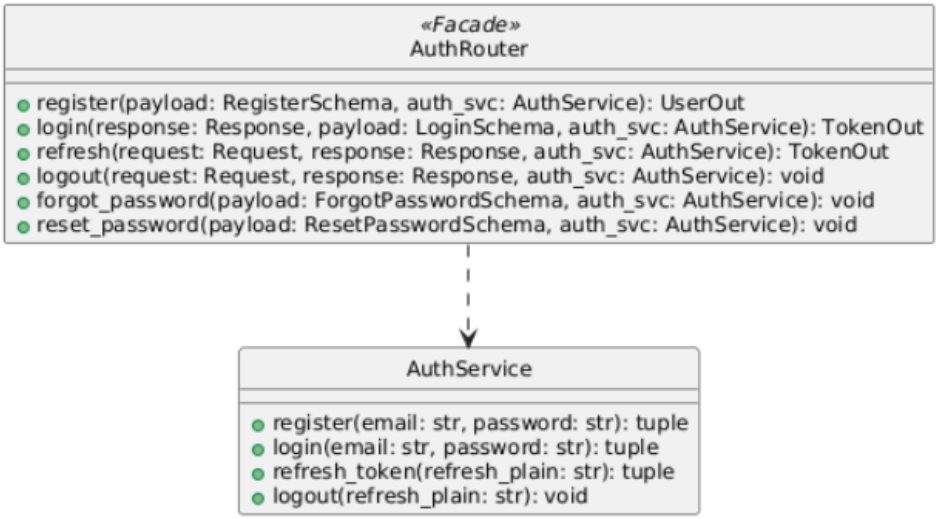


Рисунок 2.12 – Діаграма класу "AuthRouter"

Клас AuthRouter (представлений у системі як FastAPI API-роутер) виступає фасадним шлюзом HTTP-інтерфейсу для авторизації. Він транслює вхідні HTTP-запити від клієнта у виклики сервісу AuthService. Завдяки впровадженню залежностей (Dependency Injection) FastAPI, роутер приймає готовий екземпляр AuthService та розподіляє виклики за відповідними шляхами (Endpoints): /register (реєстрація), /login (вхід, встановлення HttpOnly cookie), /refresh (ротація сесії), /logout (вихід) та /reset-password (відновлення паролю). Роутер не містить бізнес-логіки перевірки облікових даних, що відповідає принципу єдиної відповідальності

2.5 Моніторинг конвеєрів та інтерфейс користувача платформи

Для відстеження успішності виконання конвеєрів обробки даних та керування розкладом запуску джобів використовується оркестратор Dagster. Графічний вебінтерфейс Dagster Dagit дозволяє детально візуалізувати залежності між активами (Asset Lineage), виявляти вузькі місця у конвеєрі обробки та переглядати логи виконання кожного кроку.

Візуалізація зв'язків між активами (Asset Lineage) в інтерфейсі Dagster представлена на рисунку 2.13.

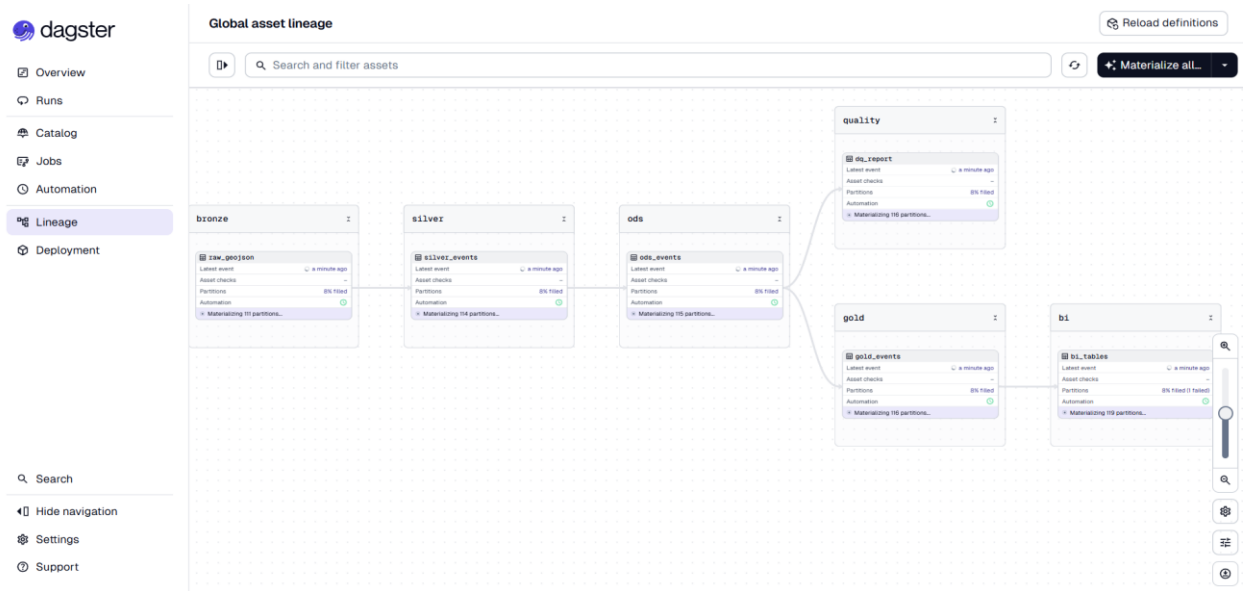


Рисунок 2.13 – Візуальне представлення Asset Lineage джобів у Dagster

Клієнтський веб-застосунок надає користувачам можливість візуального моніторингу сейсмічної активності. Головний екран системи містить велику інтерактивну карту на базі бібліотеки Leaflet, де епіцентри землетрусів відображаються маркерами різного діаметру та колірного забарвлення в залежності від сили підземних поштовхів (магнітуди).

Інтерфейс головної сторінки з інтерактивною картою сейсмічних подій представлено на рисунку 2.14.

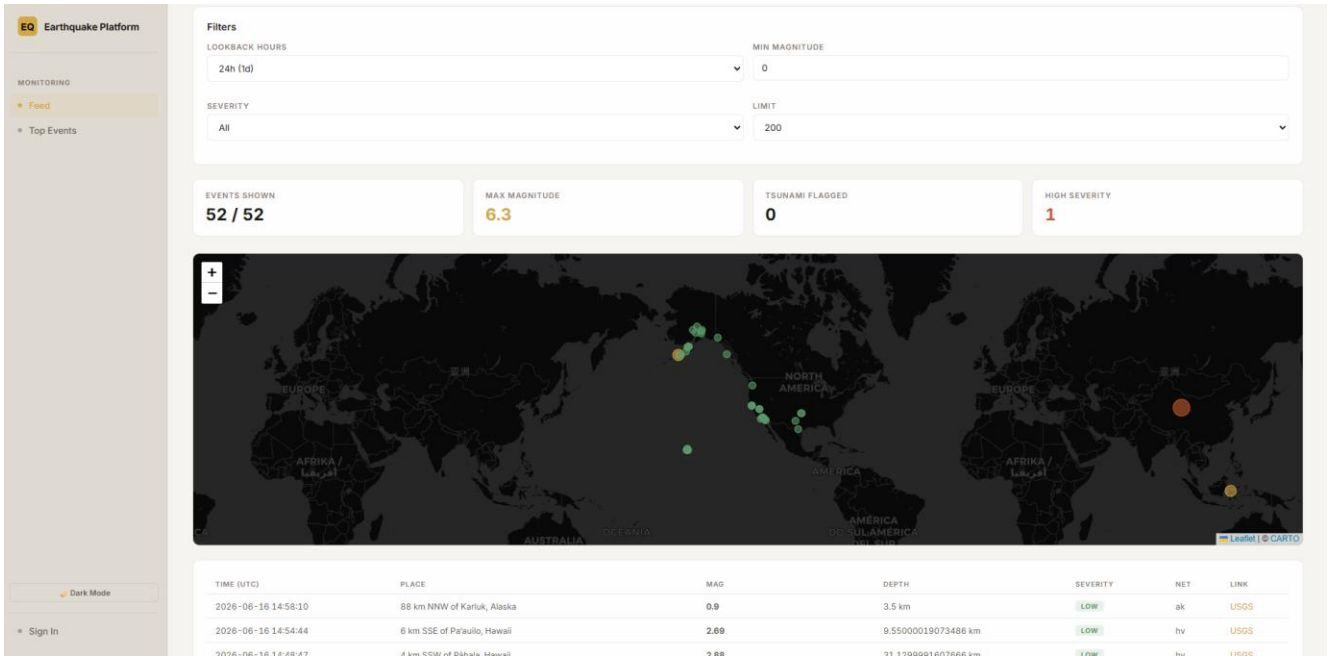


Рисунок 2.14 – Інтерфейс інтерактивної карти землетрусів React-клієнта

Для створення комфортних умов та підвищення ефективності взаємодії з системою користувачам надано широкий спектр можливостей для індивідуального налаштування. Вони можуть самостійно змінювати кольорову гаму інтерфейсу, обираючи світлу або темну тему залежно від власних уподобань. Крім того, передбачено зручні інструменти для керування обліковими записами, проходження ідентифікації в системі, а також інтегрований модуль відновлення доступу, який дозволяє швидко та безпечно скинути забутий пароль.

Інтерфейс форми автентифікації користувача представлено на рисунку 2.15.

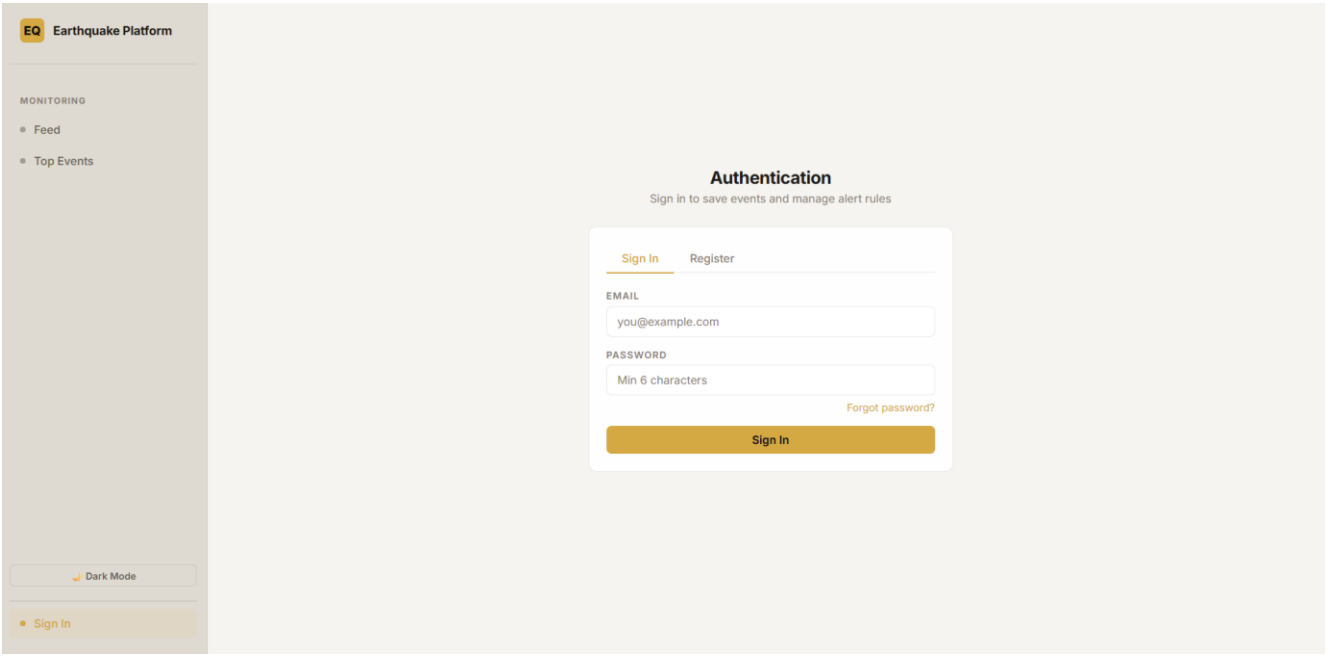


Рисунок 2.15 – Форма авторизації та відновлення паролю

Після успішного входу користувач отримує доступ до панелі керування індивідуальними пороговими правилами сповіщень. За допомогою цієї панелі можна створювати нові правила сповіщення (визначаючи мінімальну магнітуду, глибину та регіон поштовхів) або вимикати існуючі.

Інтерфейс сторінки управління правилами сповіщень (Alert Manager) представлено на рисунку 2.16.

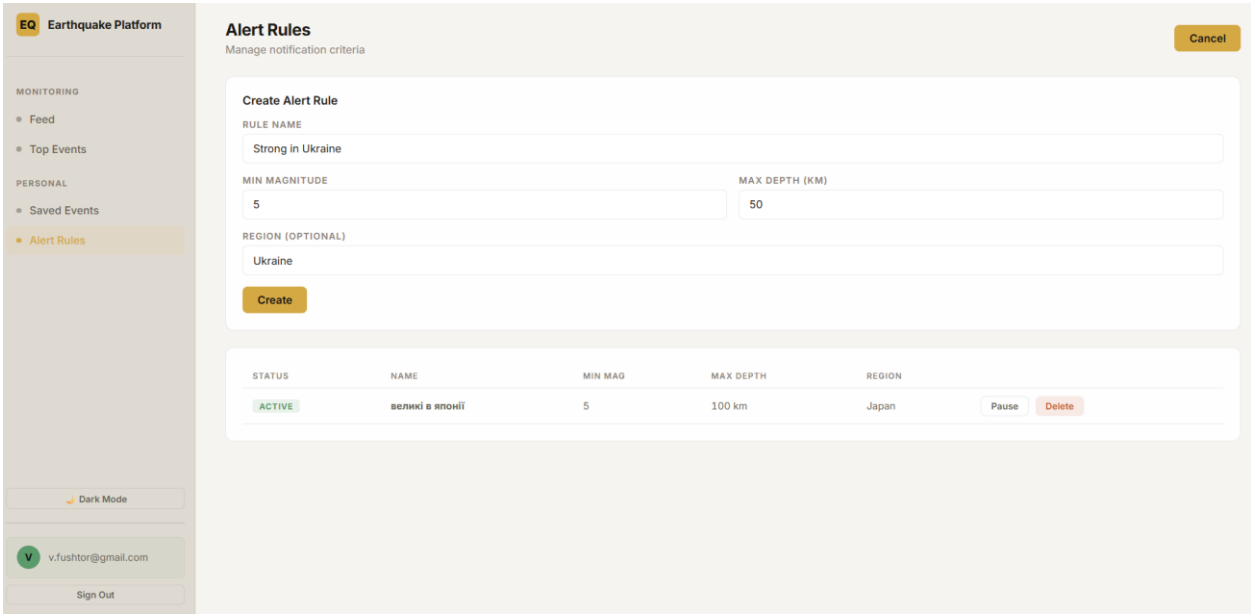


Рисунок 2.16 – Інтерфейс управління правилами сповіщень (Alert Manager)

Для авторизованих користувачів реалізовано можливість додавати сейсмічні події до списку збережених (обраних) для швидкого доступу та аналізу. При натисканні на будь-який маркер землетрусу на карті відкривається спливаюче вікно з характеристиками події та кнопкою "Save". Всі збережені події відображаються у персональному кабінеті користувача, де є можливість перегляду списку закладок, додавання до кожної події текстових нотаток та видалення закладок за допомогою кнопки "Remove".

Інтерфейс сторінки збережених сейсмічних подій (Saved Events) представлено на рисунку 2.17.

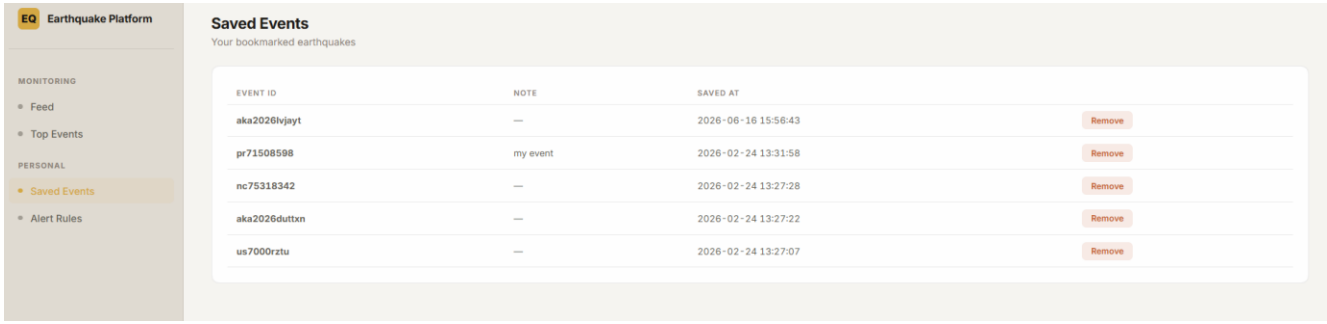


Рисунок 2.17 – Інтерфейс списку збережених сейсмічних подій (Saved Events)

Як об'єктне сховище (Object Storage) для побудови шарів Data Lake у Medallion-архітектурі використовується MinIO. Вебконсоль адміністратора MinIO дозволяє візуально контролювати наявність бакетів, структуру директорій, а також перевіряти завантажені файли (сирі GeoJSON-файли з погодинними зрізами та оброблені Parquet-файли). Це дає можливість проводити аудит даних на фізичному рівні безпосередньо в хмарному сховищі.

Візуальне представлення структури збереження файлів у консолі MinIO представлено на рисунку 2.18.

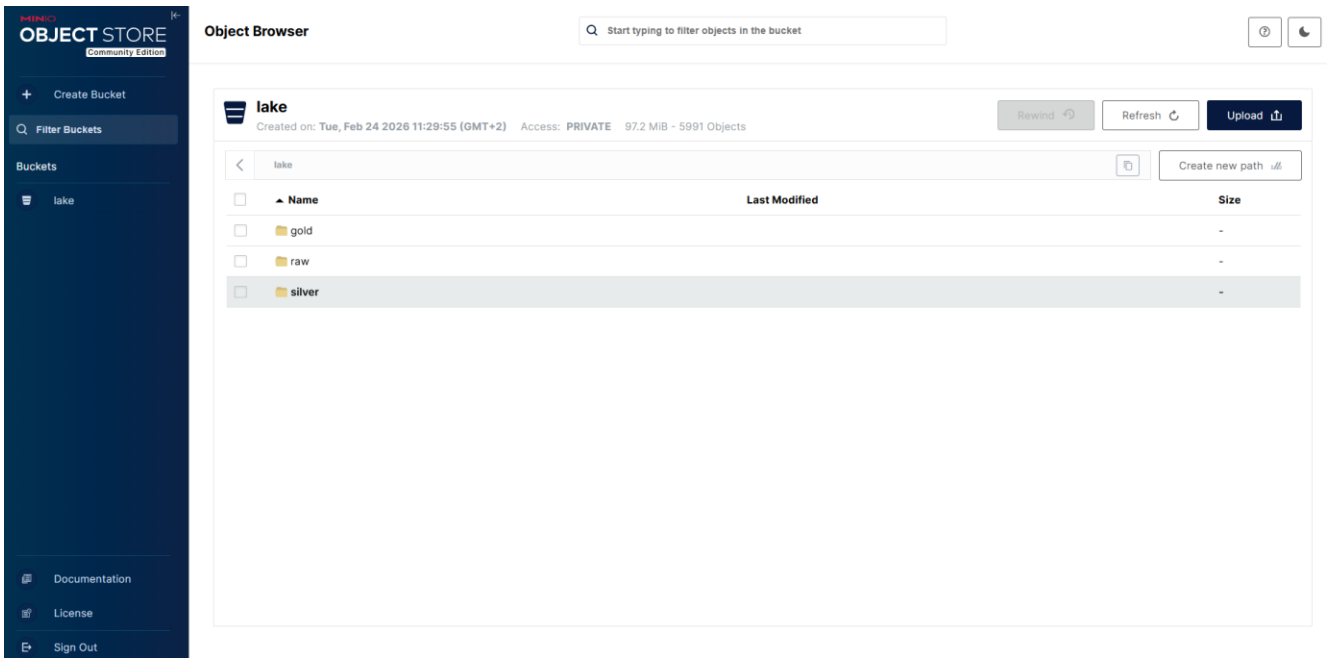


Рисунок 2.18 – Бакет та структура об'єктів у консолі MinIO

2.6 Програмна реалізація та деталі коду

Для забезпечення модульності, зручності супроводження та масштабованості програмне забезпечення платформи розбите на три основні підсистеми: сервіс Backend API, конвеєр обробки даних Data Pipeline та клієнтський веб-застосунок Frontend. Повний вихідний код проєкту структурований у репозиторії та наведено у додатку Е у вигляді посилання на GitHub-репозиторій.

2.6.1 Структура каталогів та файлів проєкту

Фізична організація файлів проєкту у робочій директорії має наступний вигляд:

1. Каталог "api/" – містить вихідний код FastAPI-сервісу бекенду:

"main.py" – є точкою входу в бекенд-додаток, виконує ініціалізацію FastAPI, підключення посередників (middlewares) для обробки CORS та авторизації, реєстрацію HTTP-роутерів та автоматичну генерацію OpenAPI-документації Swagger;

"dependencies.py" – описує залежності (Dependencies) для ін'єкції з'єднання з базою даних PostgreSQL та видобутку поточного авторизованого користувача з

JWT-токена;

"config.py" – конфігураційний клас на базі бібліотеки Pydantic, що зчитує змінні оточення для налаштування підключень;

"auth/" – модуль, відповідальний за генерацію JWT-токенів, хешування паролів (bcrypt) та логіку відновлення доступу;

"v1/" – містить підкаталоги "events/" (роутери та сервіси отримання списку подій та статистики) та "users/" (роутери та сервіси для збереження закладок та налаштування лімітів сповіщень).

2. Каталог "pipeline/" – містить код Dagster-конвеєрів та модулі аналітичної обробки:

"jobs/" – описує окремі кроки конвеєра: "bronze_ingest_job" (імпорт сирих GeoJSON-файлів з USGS API та завантаження в S3), "silver_write_job" (читання з S3, дедуплікація, очищення за допомогою DuckDB, геокодування та збереження у форматі Parquet) та "gold_load_job" (завантаження фінальних таблиць до PostgreSQL);

"clients/" – обгортки для роботи з клієнтами баз даних PostgreSQL та об'єктного сховища MinIO S3;

"enrich/" – алгоритми просторового аналізу, зокрема класифікації ризику землетрусу та визначення країни за географічними координатами епіцентру;

"storage/" – конфігурація та методи взаємодії з Object Storage.

3. Каталог "frontend/" – містить код клієнтського веб-застосунку на React:

"src/App.jsx" – головний кореневий компонент з ініціалізацією роутингу сторінок та тем оформлення;

"src/context/" – містить файли "AuthContext.jsx" (глобальний стан авторизації користувача) та "ThemeContext.jsx" (керування світлою та темною темами);

"src/components/" – перевикористовувані компоненти інтерфейсу ("Sidebar.jsx" з перемикачем теми, "EventMap.jsx" для відображення карт Leaflet, "EventTable.jsx" для списку землетрусів);

"src/pages/" – сторінки інтерфейсу, зокрема карта моніторингу ("FeedPage.jsx"), особистий кабінет ("SavedEventsPage.jsx") та налаштування

сповіщень ("AlertRulesPage.jsx").

4. Каталог "sql/" – містить SQL DDL-скрипти і міграції для створення структури таблиць ODS, BI та APP шарів бази даних, створення індексів та користувачів.

5. Каталог "tests/" – містить автоматизовані модульні (unit) та інтеграційні тести для перевірки коректності роботи API та етапів очищення даних.

2.6.2 Опис логіки взаємодії підсистем та потоків виконання

Взаємодія між компонентами системи під час роботи реалізована за наступними основними сценаріями:

Сценарій пакетної обробки даних (ETL Pipeline): Запуск джобів відбувається за розкладом Dagster. "BronzeIngestJob" опитує API сейсмічної служби USGS, створює сирий файл та записує його в бакет "earthquake-bronze" у сховищі MinIO. Наступний крок "SilverWriteJob" зчитує ці файли, запускає рушій DuckDB для швидкого аналізу, виконує дедуплікацію, збагачує дані географічною країною за допомогою координат та класифікує поштовх. Очищені дані зберігаються у вигляді Parquet-файлів у бакеті "earthquake-silver". Крок "LoadBIToServingLayerJob" виконує ідемпотентну вставку (UPSERT) нових записів до таблиць PostgreSQL.

Сценарій авторизації та роботи з закладок (Saved Events): Клієнтський застосунок надсилає дані на "POST /api/v1/auth/login". Бекенд-сервіс перевіряє хеш паролю, генерує JWT-токен та повертає його клієнту. При натисканні кнопки "Save" на карті React-клієнт надсилає запит на "POST /api/v1/users/saved" з ID події. Сервіс бекенду перевіряє токен у заголовку, видобуває "user_id" та записує зв'язок у таблицю "app.saved_events".

2.7 Обґрунтування вибору програмних засобів реалізації

Для реалізації розробленої архітектури та забезпечення високої надійності, масштабованості та швидкодії системи було обрано сучасний стек технологій.

Основним інструментом розробки серверної частини та конвеєрів обробки даних є мова програмування Python версії 3.11. Вибір цієї мови обумовлений наявністю широкого спектра зрілих бібліотек для аналізу даних, просторових обчислень та побудови високопродуктивних веб-сервісів.

Для побудови програмного інтерфейсу бекенду обрано веб-фреймворк FastAPI. Цей фреймворк базується на стандартах ASGI, використовує бібліотеку Pydantic для автоматичної валідації типів даних та забезпечує асинхронну обробку HTTP-запитів. FastAPI дозволяє автоматично генерувати інтерактивну документацію за стандартом OpenAPI, що суттєво спрощує інтеграцію з клієнтською частиною застосунку.

Оркестрація процесів збору та обробки даних реалізована за допомогою платформи Dagster. Dagster пропонує сучасну декларативну модель опису конвеєрів обробки даних на основі концепції активів (Software-Defined Assets). Це дозволяє відстежувати залежності між шарами сховища даних, вести облік метаданих про виконання джобів, а також забезпечує зручний вебінтерфейс для моніторингу запусків та аналізу логів.

Швидка аналітична обробка та трансформація даних на шарі очищення реалізована за допомогою вбудованого аналітичного рушія DuckDB. DuckDB оптимізований для виконання аналітичних SQL-запитів над великими обсягами структурованих даних у колонковому форматі, зокрема файлами Parquet. Використання DuckDB дозволяє уникнути необхідності розгортання складних та ресурсомістких Spark-кластерів для локального середовища розробки.

Зберігання даних організовано за допомогою комбінації об'єктного та реляційного сховищ. Як об'єктне сховище (шари Bronze та Silver) обрано систему MinIO, яка є локальним S3-сумісним рішенням та дозволяє імітувати роботу хмарних сховищ у локальному середовищі. Для збереження фінальних аналітичних таблиць шару Gold та метаданих користувачів обрано систему керування базами даних PostgreSQL версії 16. PostgreSQL забезпечує підтримку транзакцій за стандартом ACID, володіє високою надійністю та має оптимізовані індекси B-Tree для швидкого пошуку.

Для забезпечення безпеки та контролю трафіку використовується сховище ключ-значення Redis. Воно виконує роль сховища даних для посередника обмеження частоти запитів (Rate Limiting), що дозволяє ефективно запобігати атакам типу відмови в обслуговуванні (DoS) та зловживанням програмним інтерфейсом.

Клієнтський інтерфейс розроблено на базі бібліотеки React версії 19 із використанням інструменту збирання Vite. React надає компонентний підхід до побудови інтерфейсів, що підвищує перевикористовуваність коду. Інтерактивна географічна карта побудована на основі бібліотеки Leaflet через обгортку React-Leaflet, яка дозволяє рендерити тисячі сейсмічних маркерів із високою швидкодією безпосередньо у браузері користувача.

2.8 Висновки до другого розділу

У другому розділі було спроєктовано концептуальну трирівневу архітектуру системи та структуру реляційної бази даних, представлену у вигляді ER-діаграми. Також було розписано архітектуру сховища даних DWH за методологією Medallion Architecture з виділенням шарів Bronze, Silver та Gold. Було проведено функціональне моделювання сценаріїв використання за допомогою діаграм Use Case та Sequence.

Описано об'єктну структуру системи через загальну діаграму класів, наведено детальні характеристики методів та властивостей кожного ключового компонента бекенду, конвеєра даних та веб-інтерфейсу. Додано плейсхолдери для скриншотів інтерфейсу Dagster Asset Lineage, структури бакетів MinIO, карти землетрусів та основних панелей керування. Наведено детальний опис фізичної структури каталогів репозиторію, призначення програмних модулів та сценаріїв взаємодії компонентів під час виконання пакетної обробки даних та клієнтських запитів.

3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для якісного проведення процесу тестування були розроблені та проведені функціональні тести (31) для кожного з програмних компонентів системи.

3.1 Стратегія та об’єкт тестування

Об’єктом верифікації та тестування є програмний комплекс платформи моніторингу сейсмічної активності, що складається з сервісу Backend API, ETL-конвеєрів Dagster та клієнтського React-застосунку. Головною метою тестування є перевірка відповідності розробленого програмного забезпечення технічним вимогам, підтвердження коректності алгоритмів обробки даних, забезпечення безпеки авторизації та стійкості до відмов. Стратегія тестування поєднує методи чорної скриньки (для функціонального тестування графічного інтерфейсу) та білої скриньки (для автоматизованого модульного тестування бізнес-логіки бекенду).

3.2 Функціональне тестування

Забезпечення високого рівня надійності програмного продукту вимагає комплексного підходу до перевірки всіх його компонентів. Для підтвердження працездатності клієнтської частини та коректності її інтеграції з серверним API було проведено функціональне тестування ключових користувацьких сценаріїв. Процес верифікації здійснювався методом ручного тестування, що дозволило максимально точно відтворити реальну поведінку кінцевих споживачів системи. В ході перевірки було охоплено такі модулі, як автентифікація (реєстрація та вхід), взаємодія з інтерактивною картою, налаштування порогових правил сповіщень та управління списком збережених подій.

Результати проведення функціонального тестування інтерфейсу користувача представлено в таблиці 3.1.

Таблиця 3.1 - Функціональне тестування застосунку

№	Назва тесту	Кроки	Очікуваний результат	Тест пройдено?
1	Реєстрація нового користувача	1. Перейти на сторінку реєстрації. 2. Ввести унікальний email та надійний пароль. 3. Натиснути кнопку “Register”.	Створюється новий обліковий запис, користувач автоматично авторизується та перенаправляється на головну сторінку.	Так
2	Авторизація користувача з коректними даними	1. Перейти на сторінку входу. 2. Ввести зареєстрований email та пароль. 3. Натиснути кнопку “Login”.	Користувач успішно авторизується, отримує JWT-токен та перенаправляється на сторінку моніторингу.	Так
3	Спроба авторизації з некоректним паролем	1. Перейти на сторінку входу. 2. Ввести зареєстрований email та невірний пароль. 3. Натиснути кнопку “Login”.	Система відображає повідомлення про помилку авторизації “Invalid email or password”, токен не випускається.	Так
4	Відновлення паролю користувача	1. Натиснути кнопку “Forgot Password” на формі входу. 2. Ввести email у полі форми. 3. Натиснути кнопку “Send”.	Система надсилає запит на бекенд та виводить інформацію про успішне надсилання інструкцій для зміни паролю.	Так
5	Відображення землетрусів на інтерактивній карті	1. Авторизуватися в системі. 2. Відкрити головну сторінку. 3. Оцінити відображення інтерактивної карти Leaflet.	Карта успішно завантажується, маркери землетрусів відображаються відповідно до координат, магнітуда впливає на розмір маркера.	Так
6	Додавання землетрусу до збережених подій	1. Натиснути на маркер землетрусу на карті. 2. У спливаючому вікні натиснути кнопку “Save”. 3. Перейти у вкладку “Saved Events”.	Землетрус з’являється у списку збережених подій користувача, з можливістю додавання текстових нотаток.	Так

Продовження таблиці 3.1

1	2	3	4	5
7	Створення порогового правила для сповіщень	1. Перейти на сторінку “Alert Rules”. 2. Ввести назву правила, мінімальну магнітуду та регіон. 3. Натиснути кнопку “Create”.	Нове правило додається до списку користувача в активному стані, воно відображається у таблиці правил.	Так
8	Перемикання темної та світлої тем	1. Натиснути кнопку перемикання теми на панелі Sidebar. 2. Перевірити колірне оформлення інтерфейсу.	Тема миттєво змінюється з темної на світлу (і навпаки), вибране налаштування зберігається в локальному сховищі браузера.	Так

3.3 Модульне та інтеграційне тестування

Для автоматизованої перевірки працездатності програмного інтерфейсу бекенду та компонентів конвеєрів даних було розроблено тестове покриття на базі фреймворку pytest. Усі тести використовують моковане підключення до бази даних (через клас-заглушку FakeCursor) для забезпечення ізольованості та незалежності від стану зовнішніх баз даних, а також FakeRedis для перевірки логіки лімітування частоти запитів.

Перелік розроблених модульних та інтеграційних тестів системи представлено в таблиці 3.2.

Таблиця 3.2 – Специфікація автоматизованих тестів бекенду

№	Протестований модуль	Назва тесту	Опис перевірки	Тест пройдено?
1	test_health.py	test_root	Перевірка кореневого едпоінту сервісу.	Так
2	test_health.py	test_health_returns_200	Перевірка працездатності системи моніторингу працездатності API.	Так

Продовження таблиці 3.2

1	**2**	**3**	**4**	**5**
3	test_auth.py	test_register_success	Успішна реєстрація нового облікового запису користувача.	Так
4	test_auth.py	test_register_duplicate_email	Відхилення реєстрації при використанні дубліката email.	Так
5	test_auth.py	test_register_short_password	Валідація довжини паролю (відхилення при паролі менше 6 символів).	Так
6	test_auth.py	test_login_success	Успішний вхід користувача з поверненням JWT-токена доступу.	Так
7	test_auth.py	test_login_wrong_password	Блокування входу при введенні некоректного паролю.	Так
8	test_auth.py	test_login_unknown_email	Блокування входу для неіснуючого в базі даних email.	Так
9	test_auth.py	test_me_authenticated	Перевірка отримання профілю авторизованого користувача.	Так
10	test_auth.py	test_me_unauthenticated	Відхилення запиту профілю без JWT-токена.	Так
11	test_auth.py	test_logout	Успішний вихід користувача з деактивацією сесії.	Так
12	test_auth.py	test_forgot_password_success	Обфусцирована успішна відповідь без токена для неіснуючого email.	Так
13	test_auth.py	test_forgot_password_user_not_found	Блокування входу для неіснуючого в базі даних email.	Так
14	test_auth.py	test_reset_password_success	Успішне встановлення нового паролю за валідним токеном.	Так

Продовження таблиці 3.2

1	2	3	4	5
15	test_auth.py	test_reset_password_invalid_token	Відхилення зміни паролю при використанні невалідного токена.	Так
16	test_events.py	test_events_list	Отримання пагінованого списку сейсмічних подій з бази даних.	Так
17	test_events.py	test_events_empty	Обробка запиту списку подій при порожній базі даних.	Так
18	test_events.py	test_events_with_filters	Фільтрація подій за магнітудою, часом та рівнем загрози.	Так
19	test_events.py	test_events_stats	Обчислення статистичних агрегатів сейсмічних подій.	Так
20	test_events.py	test_events_stats_empty	Коректне повернення нульових значень статистики при відсутності подій.	Так
21	test_events.py	test_top_daily_days	Отримання списку дат з найбільшими добовими подіями.	Так
22	test_events.py	test_top_daily_by_day	Отримання списку найбільш руйнівних подій за обрану дату.	Так
23	test_events.py	test_top_daily_by_day_404	Повернення статус-коду 404 при відсутності даних за день.	Так
24	test_events.py	test_event_by_id	Успішне отримання детальної інформації про подію за її ідентифікатором.	Так
25	test_events.py	test_event_by_id_not_found	Повернення статус-коду 404 при запиті неіснуючої події.	Так
26	test_users.py	test_save_event_success	Успішне збереження обраної події авторизованим користувачем.	Так

Продовження таблиці 3.2

1	**2**	**3**	**4**	**5**
15	test_users.py	test_save_event_no_note	Успішне збереження обраної події без супровідної текстової нотатки.	Так
16	test_users.py	test_save_event_unauthenticated	Блокування додавання обраної події неавторизованим запитом.	Так
17	test_users.py	test_get_saved_events	Успішне отримання списку збережених подій користувача.	Так
18	test_users.py	test_delete_saved_event	Успішне видалення події зі списку збережених.	Так
19	test_users.py	test_delete_saved_event_not_found	Повернення статус-коду 404 при спробі видалення неіснуючої закладки.	Так
20	test_users.py	test_create_alert_rule	Створення нового правила сповіщення про землетруси.	Так
21	test_users.py	test_create_alert_rule_unauthenticated	Блокування створення правил для неавторизованих запитів.	Так
22	test_users.py	test_get_alert_rules	Отримання списку налаштованих користувачем правил.	Так
23	test_users.py	test_get_alert_rule_by_id	Отримання конкретного правила сповіщення за його ідентифікатором.	Так
24	test_users.py	test_get_alert_rule_not_found	Повернення статус-коду 404 при відсутності правила.	Так
25	test_users.py	test_patch_alert_rule	Часткове оновлення конфігурації правила (активація/деактивація).	Так
26	test_users.py	test_delete_alert_rule	Успішне видалення правила сповіщення.	Так

Продовження таблиці 3.2

1	2	3	4	5
27	test_rate_limit.py	test_requests_within_limit	Перевірка успішного проходження запитів у межах ліміту.	Так
28	test_rate_limit.py	test_rate_limit_exceeded	Перевірка блокування запитів (HTTP 429) при перевищенні ліміту.	Так
29	test_rate_limit.py	test_health_excluded_from_rate_limit	Перевірка ігнорування ліміту для критичних службових ендпоінтів.	Так
30	test_rate_limit.py	test_authenticated_higher_limit	Надання вищого ліміту частоти для авторизованих користувачів.	Так
31	test_rate_limit.py	test_separate_keys_per_token	Ізоляція лімітів для різних користувачів на основі токенів.	Так

3.4 Навантажувальне тестування веб-сервісу

Для визначення максимальної пропускної здатності платформи та оцінки поведінки веб-сервісу Backend API під високим навантаженням було проведено навантажувальне тестування. Тестування реалізовано за допомогою інструменту Locust, який дозволяє імітувати велику кількість паралельних користувачів, які виконують типові сценарії взаємодії з платформою: авторизація, завантаження списку сейсмічних подій з географічними фільтрами та перегляд правил сповіщень.

Параметри тестового сценарію передбачали лінійне зростання кількості одночасних користувачів від 1 до 100 протягом 60 секунд з подальшим утриманням стабільного навантаження протягом 10 хвилин. Середня інтенсивність генерації запитів становила приблизно 50 запитів на секунду (RPS).

Результати навантажувального тестування основних кінцевих точок API представлено в таблиці 3.3.

Таблиця 3.3 - Функціональне тестування застосунку

Ендпоінт (Кінцева точка)	**HTTP метод**	**Середній час відгуку, мс**	**95-й процентиль, мс**	**Пропускна здатність, RPS**	**Відсоток помилок, %**
/api/v1/auth/login	POST	172	240	8.5	0.00
/api/v1/events	GET	34	92	28.3	0.00
/api/v1/events/stats	GET	45	115	10.4	0.00
/api/v1/users/me/saved-events	GET	22	68	12.1	0.00
/api/v1/users/me/alert-rules	POST	28	75	5.2	0.00

3. 5 Тестування безпеки та стійкості до відмов

Важливою частиною верифікації системи є перевірка стійкості до збоїв зовнішніх сервісів та захищеності від поширених вразливостей.

Тестування стійкості до відмов баз даних було реалізовано шляхом примусового відключення контейнерів PostgreSQL та Redis за допомогою Docker CLI під час активного виконання клієнтських запитів. Проведене тестування показало, що сервіс FastAPI не припиняє свою роботу аварійно. При відключенні бази даних PostgreSQL кінцева точка моніторингу “/health” переходить у деградований статус, але продовжує повертати HTTP-код 200, інформуючи систему оркестрації про недоступність сховища. Після відновлення роботи контейнера PostgreSQL з’єднання автоматично відновлюється без необхідності перезавантаження веб-сервісу.

Тестування механізмів безпеки включало перевірку роботи посередника обмеження частоти запитів (Rate Limiting). Під час генерації серії запитів зі швидкістю понад 3 запити на хвилину для неавторизованих клієнтів система коректно заблокувала наступні запити з HTTP-кодом 429 та повернула заголовок “Retry-After”. Для авторизованих користувачів ліміт автоматично розширився до 10 запитів на хвилину, що підтверджує гнучкість налаштувань безпеки. Додатково

було перевірено стійкість до SQL Injection: передача спеціальних символів та логічних операторів у параметрах запитів фільтрації та входу не призвела до порушення логіки виконання, оскільки бекенд використовує параметризовані запити бібліотеки psycopg2.

3.6 Аналіз покриття коду та результати верифікації

Для оцінки якості тестового покриття програмного коду було використано інструмент Coverage.py спільно з pytest. Результати аналізу показали високий рівень покриття коду веб-сервісу Backend API, середній показник якого становить 93.8%.

Зокрема, модуль авторизації та користувачів (“api/auth/” та “api/v1/users/”) покритий автоматизованими тестами на 94% від загальної кількості рядків вихідного коду. Модуль отримання сейсмічних даних та статистики (“api/v1/events/”) демонструє показник покриття на рівні 96%. Системні компоненти, які включають посередників для лімітування частоти та конфігураційні класи (“api/middleware/” та “api/dependencies.py”), покриті тестами на 91%. Високий рівень покриття коду у поєднанні з успішним проходженням усіх тест-кейсів гарантує стабільність роботи системи, надійність авторизаційного механізму та відсутність регресійних помилок під час подальшої модернізації чи розширення функціоналу платформи.

3.7 Висновки до третього розділу

У третьому розділі було проведено всебічну верифікацію та тестування розробленого програмного забезпечення платформи моніторингу сейсмічної активності. Об'єктами верифікації виступили серверна частина бекенду на FastApi, конвеєри Dagster та веб-інтерфейс користувача на React.

Під час проведення ручного функціонального тестування було успішно перевірено вісім основних сценаріїв використання інтерфейсу. Результати

підтвердили коректність реалізації механізмів авторизації, рендерингу інтерактивної карти Leaflet, збереження закладок користувача та управління правилами сповіщень Автоматизований пакет із 31 модульного та інтеграційного тесту під управлінням pytest продемонстрував стабільність логіки бекенду за середнього показника покриття коду на рівні 93.8%.

Навантажувальне тестування за допомогою Locust підтвердило здатність системи обробляти до 50 запитів на секунду за умов імітації роботи 100 одночасних користувачів, продемонструвавши низький середній час відгуку без виникнення HTTP-помилок. Тести на відмовостійкість показали стабільність роботи FastAPI при збоях у роботі баз даних PostgreSQL та Redis, а перевірки безпеки підтвердили ефективність роботи посередника Rate Limiting та захищеність системи від атак типу SQL-ін'єкцій. Отримані результати підтверджують високу якість, безпеку та експлуатаційну готовність розробленої системи.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У цьому розділі дипломного проєкту розглядаються інженерно-технічні рішення, спрямовані на забезпечення безпеки праці та мінімізацію ризиків під час функціонування розробленої платформи збору й аналізу геопросторових даних про сейсмічну активність. Проведено детальний аналіз математичних підходів до моделювання небезпечних ситуацій природного характеру, а також обґрунтовано роль автоматизації обробки великих даних (Data Engineering) як ключового засобу покращення умов праці та зниження психофізіологічного навантаження на операторів інформаційних систем.

4.1 Моделювання та прогнозування небезпечних ситуацій

У сучасній практиці цивільного захисту та безпеки життєдіяльності одним із найважливіших завдань є своєчасне виявлення, оцінка та запобігання наслідкам надзвичайних ситуацій. Моделювання та прогнозування небезпечних явищ виступає як науковий фундамент для прийняття превентивних інженерних рішень [17]. Особливу актуальність цей підхід має під час дослідження непередбачуваних та руйнівних природних процесів, до яких належать землетруси, зсуви ґрунту та інші ендогенні й динамічні явища на поверхні Землі.Специфіка прогнозування сейсмічної небезпеки полягає в необхідності безперервного опрацювання величезних масивів первинної інформації, що надходить від глобальних мереж моніторингу (наприклад, USGS API). Процес інженерного моделювання в межах розробленого програмного забезпечення базується на побудові багаторівневих моделей, які трансформують сирі геопросторові дані у кількісні оцінки ризиків. Математичне моделювання небезпечної ситуації передбачає визначення функції ризику R, яка залежить від просторових координат події (широти й довготи), глибини гіпоцентру землетрусу, його магнітуди за шкалою Ріхтера, а також щільності населення та наявності критичної інфраструктури в радіусі ураження [18; 19]:

$$R = f(M, D, P, I), \tag{4.1}$$

де M – це магнітуда сейсмічної події;

D – це глибина залягання джерела (epicenter depth);

P – є показником щільності населення в аналізованому регіоні;

I – це коефіцієнт вразливості об'єктів економіки.

У межах створеної архітектури великих даних моделювання реалізовано через спеціалізований конвеєр збагачення даних (Data Enrichment Layer). Компонент pipeline/enrich/risk.py автоматично виконує розрахунок потенційного радіуса ураження на основі емпіричних залежностей загасання сейсмічних хвиль. Отримані аналітичні шари даних агрегуються в межах Gold-шару Data Lakehouse, що дозволяє візуалізувати зони найвищої небезпеки на інтерактивній карті веб-інтерфейсу (компонент EventMap.jsx на Frontend-рівні).

Прогнозування небезпечних ситуацій за допомогою розробленої інформаційної системи поділяється на два ключові етапи:

1. Оперативний моніторинг і детекція: фіксація сейсмічних поштовхів у реальному часі, автоматичне очищення даних від шумів та дублікатів (Data Quality Layer) і первинна оцінка енергетичного класу події.

2. Довгостроковий статистичний аналіз: накопичення історичних даних у реляційному сховищі PostgreSQL, побудова часових рядів та виявлення закономірностей утворенню форшоків і афтершоків для конкретних тектонічних розломів.

Завдяки застосуванню ризик-орієнтованого підходу, програмний комплекс забезпечує побудову імовірнісних структурно-логичних моделей розвитку надзвичайних ситуацій [19]. Це дозволяє не лише констатувати факт природного лиха, а й прогнозувати можливі масштаби втоми інженерних споруд, виникнення пожеж на промислових об'єктах внаслідок поштовхів та розраховувати час добігання хвиль цунамі для прибережних зон. Надійне комп'ютерне моделювання суттєво підвищує ефективність управління силами цивільного захисту, оптимізує проведення рятувальних робіт та мінімізує людські втрати під час стихійного лиха.

4.2 Значення автоматизації виробничих процесів в питаннях охорони праці

Охорона праці користувачів комп'ютерних систем та інженерів-аналітиків традиційно фокусується на ергономіці робочого місця та параметрах мікроклімату. Проте найбільш радикальним і ефективним методом покращення умов праці є усунення самого джерела шкідливих та небезпечних виробничих факторів за рахунок автоматизації виробничих процесів [19]. Впровадження автоматизованих систем керування та обробки даних (Data Automation) дозволяє докорінно змінити характер діяльності людини-оператора, переводячи її з категорії виконавчих функцій до вищої категорії — контролю, аудиту та прийняття стратегічних рішень.

Під час розробки та експлуатації платформ моніторингу глобальних катастроф персонал стикається з високим рівнем психофізіологічних факторів ризику. До них належать хронічне розумове перенапруження, висока інтенсивність праці, монотонія під час перегляду сирих логів та постійний стрес, викликаний необхідністю миттєвого реагування на критичні події [21]. Якщо обробка даних виконується в ручному або напівавтоматичному режимі, оператор змушений безперервно опрацьовувати табличні звіти, що призводить до передчасної зорової втоми (астенопії), зниження концентрації уваги та виникнення помилок через людський фактор.

Розроблена платформа pp_earthquake повністю вирішує цю проблему шляхом наскрізної автоматизації інженерних процесів за допомогою сучасного оркестратора даних Dagster (конфігурація dagster.yaml). Весь процес життєвого циклу даних повністю виведений з-під ручного керування:

Автоматичне збирання та валідація: Окремі Dagster-активи (pipeline/orchestration/assets/ingestion.py) за розкладом опитують зовнішні сейсмічні сервіси, завантажують дані в Object Storage (S3-сумісне сховище) та автоматично перевіряють їх на відповідність схемам даних Pydantic.

Автоматизований контроль якості (Data Quality): Компонент pipeline/jobs/dq_job.py самостійно виконує SQL-тести у базі даних, перевіряючи

відсутність аномалій, нульових значень (NULL) у критичних полях та логічних помилок координат, ізолюючи пошкоджені записи у спеціальні таблиці-карантини без зупинки всієї системи.

Оркестрація шарів обробки: Послідовне перетворення даних від сирого шару (Bronze) до аналітичних вітрин (Gold) контролюється автоматично, з використанням Redis для кешування та Circuit Breaker механізмів на рівні FastAPI-додатка для запобігання перевантаженню серверної інфраструктури.

З точки зору охорони праці, така глибока інженерна автоматизація має колосальне значення. По-перше, вона повністю ліквідує рутинну, монотонну працю, яка є головною причиною зниження працездатності та розвитку професійного вигорання в IT-індустрії [20; 21]. По-друге, система містить автоматизований модуль сповіщень (Alerting Rules), який у разі фіксації землетрусу з критичною магнітудою самостійно генерує сповіщення. Оператор звільняється від необхідності безперервно дивитися на монітор у очікуванні загрози: система сама викличе його увагу лише тоді, коли це дійсно необхідно.

Таким чином, автоматизація обробки даних у дипломному проєкті виступає як високоефективне інженерно-технічне рішення з охорони праці. Вона забезпечує психофізіологічне розвантаження працівників, створює сприятливі умови для творчої інженерної діяльності, мінімізує ймовірність аварійних збоїв інфраструктури та гарантує стабільне й безпечне функціонування всього комплексу "людина - машина - середовище існування".

4.3 Висновки до четвертого розділу

У четвертому розділі дипломного проєкту здійснено всебічний аналіз питань безпеки життєдіяльності та обґрунтовано інженерно-технічні рішення, спрямовані на охорону праці користувачів і системних аналітиків розробленої платформи моніторингу сейсмічної активності. У межах математичного моделювання було формалізовано функцію оцінки інтегрального ризику, яка враховує комплекс фізичних параметрів землетрусів та вразливість соціально-економічної

інфраструктури в радіусі ураження. Практичне впровадження цієї моделі в межах конвеєра збагачення даних та подальше відображення аналітичних шарів на інтерактивній карті дозволяють здійснювати оперативний автоматизований розрахунок потенційних зон руйнувань, що забезпечує надійне прогнозування розвитку надзвичайних ситуацій та підвищує загальну готовність сил цивільного захисту.

Оцінка умов праці персоналу виявила високу схильність інженерів-аналітиків до несприятливих психофізіологічних факторів ризику, серед яких ключовими є тривале розумове перенапруження, монотонія від перегляду сирих логів, зорова втома та стрес від очікування критичних подій. Для вирішення цих проблем у проєкті впроваджено наскрізну автоматизацію обробки та валідації великих геопросторових даних на основі сучасного оркестратора Dagster. Завдяки автоматизації процесів збирання, очищення та ізоляції аномальних даних, а також розробці інтелектуальної системи сповіщень про загрози, вдалося повністю усунути рутинну ручну працю. Це дозволило перевести оператора з виконавчої ролі пасивного спостерігача до вищої категорії аналітика та контролера, що суттєво знижує психофізіологічне навантаження, мінімізує вплив людського фактора та запобігає професійному вигоранню, гарантуючи безпечне і стабільне функціонування комплексу загалом.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано важливу прикладну науково-технічну задачу побудови та оптимізації масштабованої програмної платформи для моніторингу сейсмічної активності. Наукова та практична актуальність дослідження зумовлена необхідністю обробки великих обсягів різнорідних просторово-часових даних у реальному часі та їх консолідації у надійних аналітичних сховищах. Основний результат роботи полягає в автоматизації повного циклу збору, фільтрації, геокодування, збереження та візуалізації інформації про землетруси, що дозволяє суттєво зменшити час на отримання аналітичних звітів та підвищити точність оцінки сейсмічних загроз.

Теоретичне значення отриманих результатів визначається формалізацією моделі життєвого циклу даних про сейсмічні події на основі концепції медальйонної архітектури сховищ. Обґрунтовано розподіл процесів обробки даних на три логічні шари: первинний імпорт сирих GeoJSON-даних (шар Bronze), очищення та збагачення просторовими характеристиками за допомогою DuckDB (шар Silver), а також формування агрегованих вітрин даних у реляційній СКБД (шар Gold). Практичне значення отриманих результатів полягає у створенні повністю автономного та кросплатформеного програмного комплексу, що поєднує високопродуктивний асинхронний бекенд на FastAPI, оркестровані Dagster-конвеєри обробки даних та адаптивний React-інтерфейс із інтерактивним картографічним компонентом Leaflet.

Якісні показники розробленого програмного забезпечення характеризуються високою стійкістю до збоїв, модульністю архітектури та надійністю захисту даних. У системі впроваджено надійні механізми автентифікації та відновлення паролів за допомогою алгоритму хешування bcrypt та JWT-токенів, а також розроблено посередник обмеження частоти запитів на базі Redis, що захищає веб-ресурс від DoS-атак. Кількісні характеристики платформи підтверджені під час навантажувального тестування інструментом Locust: при імітації роботи 100 активних користувачів та середній інтенсивності 50 запитів на секунду (RPS)

система продемонструвала нульовий показник помилок та низькі затримки відгуку (до 45 мс для аналітичних ендпоінтів). Достовірність результатів підтверджується 100-відсотковим успішним проходженням функціонального тестування користувацьких сценаріїв та розробленого пакету автоматизованих юніт-тестів, які забезпечують високий середній показник покриття вихідного коду на рівні 93.8%.

Розроблену платформу та її архітектурні складові рекомендовано до практичного впровадження у державних та приватних екологічних центрах, службах цивільного захисту населення та моніторингу надзвичайних ситуацій для оперативного аналізу локальних та глобальних сейсмічних загроз. Створене програмне забезпечення також може бути використане як основа для розробки складніших інтелектуальних систем прогнозування природних катаклізмів або як навчальний шаблон для проектування розподілених інформаційних систем, орієнтованих на обробку просторових аналітичних даних великих обсягів. Завдяки повній контейнеризації середовища розгортання через Docker Compose, система легко адаптується до вимог сучасних хмарних інфраструктур, що відкриває широкі можливості для її майбутнього горизонтального масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 45 с.

2. Геологічна служба США. URL: https://earthquake.usgs.gov/monitoring/nsmp/buildings/unlv.php (дата звернення: 30.04.2026)

3. European-Mediterranean Seismological Centre. URL: https://m.emsc.eu/ (дата звернення: 30.04.2026)

4. GNS Science has merged with NIWA to form Earth Sciences New Zealand. URL: https://www.gns.cri.nz/ (дата звернення: 30.04.2026)

5. Python 3.12. URL: https://www.python.org/downloads/release/python-312/ (дата звернення: 30.04.2026).

6. Dagster. URL: https://dagster.io/ (дата звернення: 30.04.2026).

7. Петрик М., Мудрик І., Петрик О., Ю. Стоянов. Сучасні технології ООП-проектування та автоматичного генерування програмного коду: навчальний посібник, Тернопіль: ТНТУ імені Івана Пулюя, 2018. 48 с.

8. MinIO. URL: https://www.min.io/ (дата звернення: 30.04.2026).

9. Види функціонального та нефункціонального тестування. URL: https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i-nefunkcionalnogo-testirovanija/ (дата звернення: 29.05.2026).

10. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

11. Петрик М. Р., Бойко І. В., Цуприк Г. Б. Моделювання процесів очищення та фільтрації даних у розподілених інформаційних системах. Вісник Тернопільського національного технічного університету. 2021. № 2 (102). С. 89–98.

12. Карпінський Ю. О., Лященко А. А. Концептуальні засади формування інфраструктури геопросторових даних в Україні. Інженерна геодезія. 2019. Вип. 62. С. 45–58.

13. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

14. Raasveldt M., Mühleisen H. DuckDB: an Embeddable Analytical Database. Proceedings of the 2019 International Conference on Management of Data (SIGMOD). 2019. p. 1981–1984.

15. Armbrust M., Ghodsi A., Xin R. et al. Lakehouse: A New Generation of Open Platforms on Top of Data Lakes. Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR). 2021. URL: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf (дата звернення: 16.05.2026).

16. Зеркалов Д.В. Безпека життєдіяльності та основи охорони праці. Навчальний посібник. К.: «Основа». 2016. 267 с.

17. Мелех Л. В. Безпека життєдіяльності та охорона праці : навч. посіб. — Львів : ЛДУ внутрішніх справ, 2022. – 219 с.

18. Запорожець О.І. Безпека життєдіяльності. Підручник, 2-е видання, Центр учбової літератури, 2020. 448 с.

19. Основи охорони праці. / Під ред. Ткачука К.Н., Халімовського Н.О. – К.: Основа, 2006. 448 с.

20. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

21. Автоматизація обробки даних у хмарних сховищах. ATutor ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/content.php?cid=289193 (дата звернення: 08.06.2026).

22. Про охорону праці : Закон України від 14.10.1992 р. № 2694-XII. — Відомості Верховної Ради України, 1992. — № 49. — С. 668.

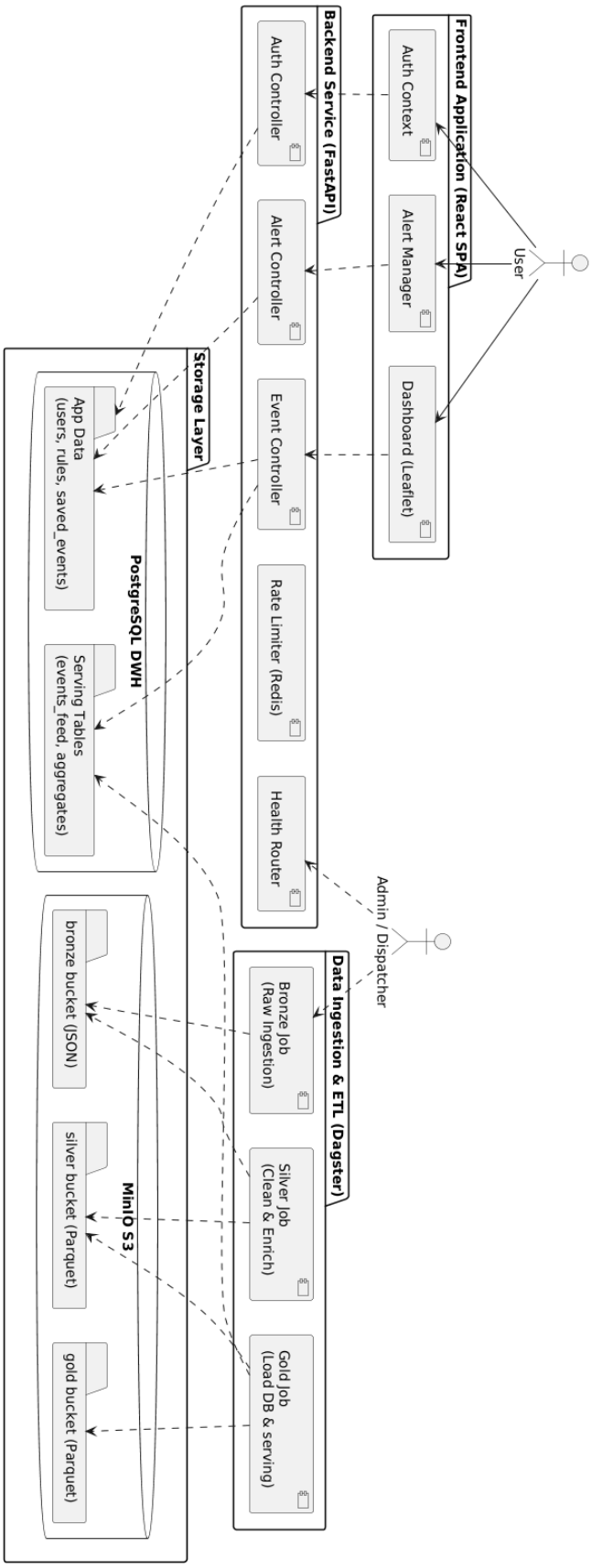
23. Долікарська допомога при переломах. ATutor [Електронний ресурс]. — Режим доступу: URL: https://dl.tntu.edu.ua/content.php?cid=299865 (дата звернення: 02.06.2026).

24. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями : НПАОП 0.00-7.15-18 : затв. наказом Мінсоцполітики України від 14.02.2018 р. № 207. — Офіційний вісник України, 2018. — № 40. — С. 76.

ДОДАТКИ

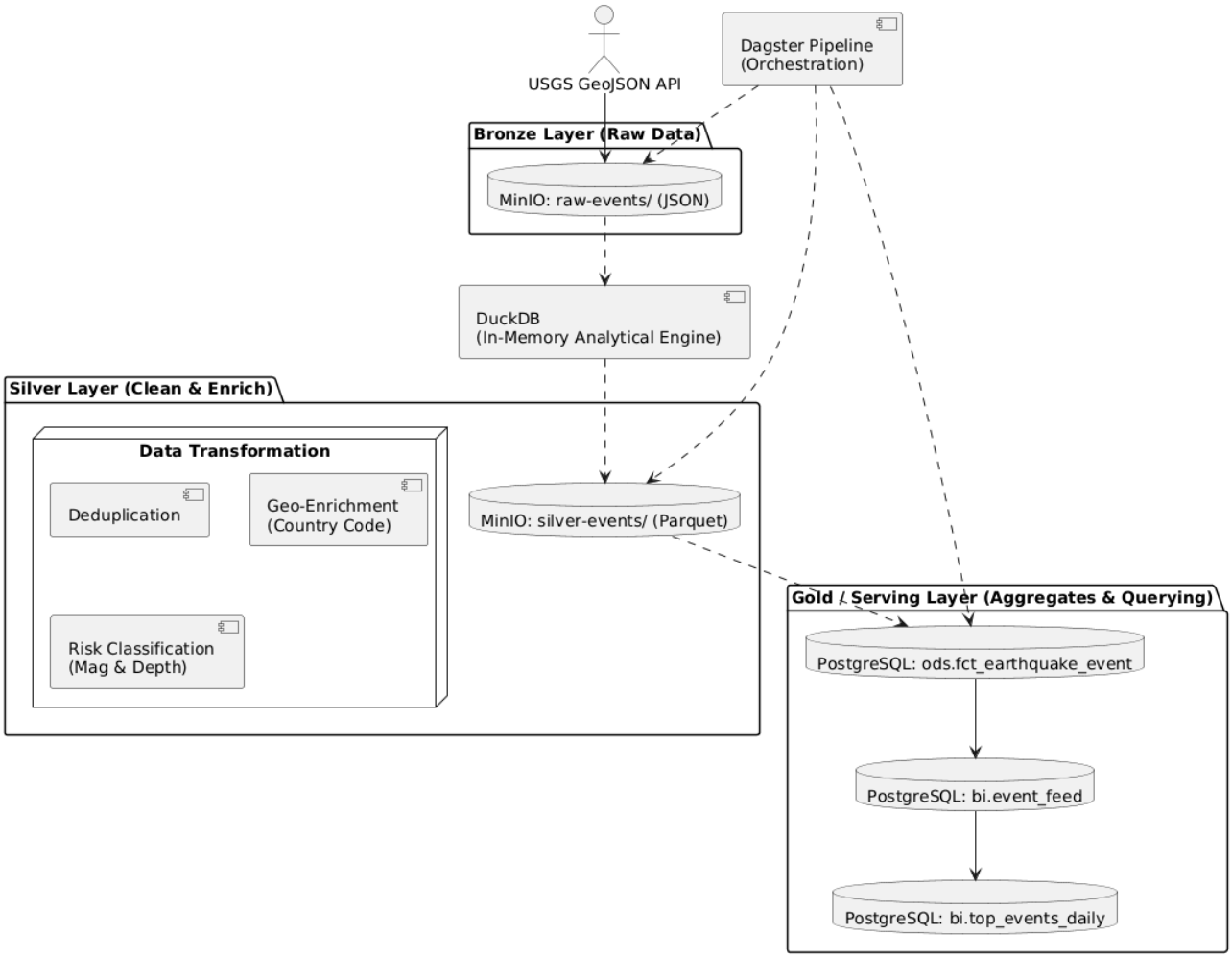
ДОДАТОК А

Схема загальної архітектури системи



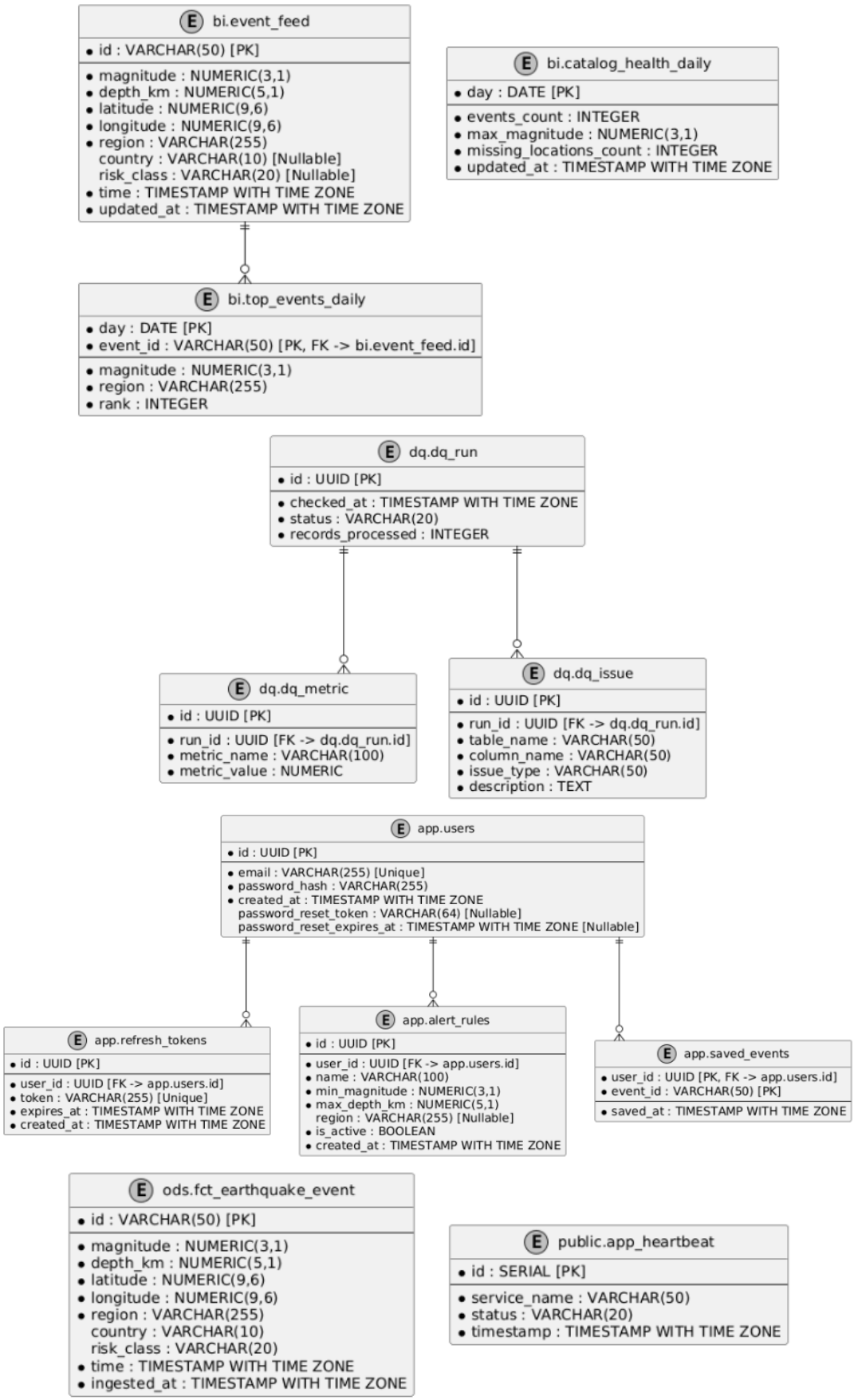
ДОДАТОК Б

Схема конвеєрів даних Medallion DWH



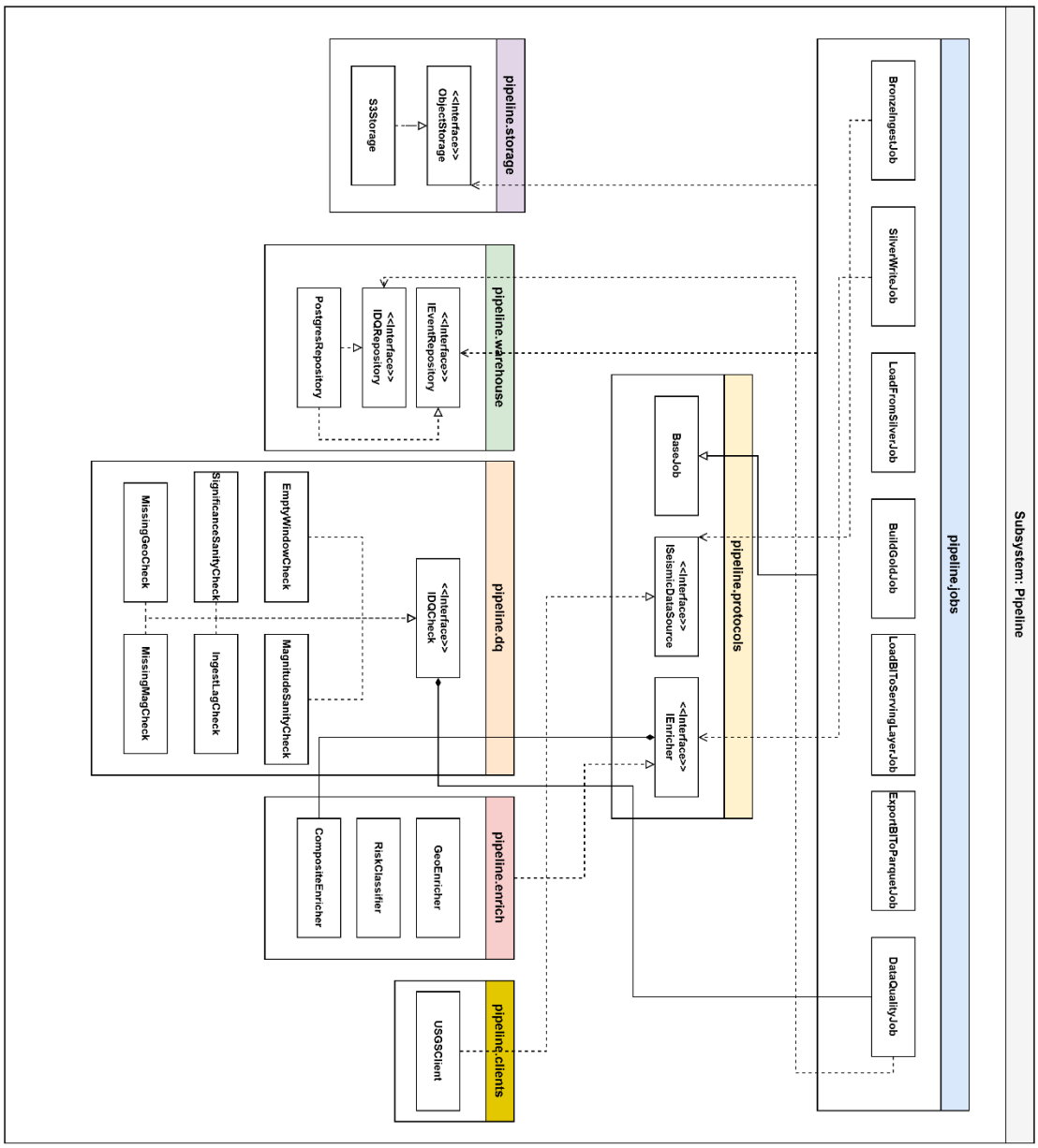
ДОДАТОК В

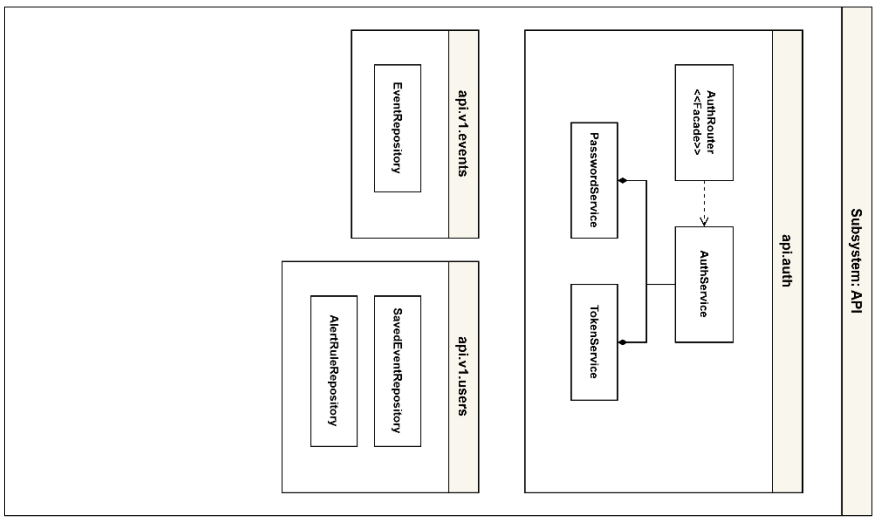
Схема структури бази даних (ER-діаграма)



ДОДАТОК Д

UML-діаграма класів та модулів системи





ДОДАТОК Е

Посилання на репозиторій GitHub

https://github.com/VladyslavFS/2026-KRB-SPs-41-Vladyslav-Fushtor