Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

“Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм міста Вінниці”

Виконав: студент 2 курсу, групи 2ІСТ-24м
спеціальності 126 – «Інформаційні системи та технології»
_____ Володимир ПАНАСЮК

Керівник: к.т.н., ас. каф. САІТ
_____ Ігор ШТЕЛЬМАХ
«27» 11 2025 р.

Опонент: к.т.н., доц. каф. КН
_____ Володимир ОЗЕРАНСЬКИЙ
«03» 12 2025 р.

Допущено до захисту
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«28» 11 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – другий (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 126 Інформаційні системи та технології
Освітньо-професійна програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
«25» 09 2025 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Панасюку Володимиру Романовичу

1. Тема роботи: "Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм міста Вінниці", керівник роботи: Ігор ШТЕЛЬМАХ, к.т.н., ас. каф. САІТ, затверджені наказом ВНТУ від «24» 09 2025 року № 313
2. Строк подання студентом роботи «28» 11 2025 року
3. Вихідні дані до роботи:
   Відкриті дані порталу моніторингу вод Держводагентства України.
4. Зміст текстової частини:
   - Аналіз предметної області, існуючих систем екологічного моніторингу та обґрунтування вибору технологій.
   - Проєктування архітектури системи, реляційної моделі бази даних, підсистем безпеки та динамічного налаштування пристроїв.
   - Програмна реалізація серверної частини (Backend), клієнтського веб-додатку (Frontend) та механізмів обміну даними в реальному часі.
   - Експериментальне дослідження роботи системи на реальних даних.
   - Економічне обґрунтування розробки та впровадження системи.
5. Перелік ілюстративного матеріалу:
   - Діаграма варіантів використання (Use Case Diagram).
   - Структурна схема архітектури Clean Architecture.
   - Діаграма послідовності обробки даних (Sequence Diagram).
   - ER-діаграма бази даних.
   - Алгоритми роботи системи (конвертація даних, виявлення аномалій).
   - Інтерфейс користувача та візуалізація даних (скріншоти системи).
   - Графіки результатів експериментальних досліджень.

6. Консультанти розділів МКР

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
3	Крижановський Є.М., доцент каф. САІТ	06.10.2025	17.10.2025
6	Олександр ЛЕСЬКО, к. е. н., проф. каф. ЕПВМ	05.11.2025	15.11.2025

7. Дата видачі завдання «25» 09 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області та існуючих рішень	15.09.2025	26.09.2025	виконано
2	Вибір оптимальних інформаційних технологій та стеку	26.09.2025	06.10.2025	виконано
3	Проєктування архітектури та бази даних системи	06.10.2025	17.10.2025	виконано
4	Програмна реалізація серверної та клієнтської частин	17.10.2025	26.10.2025	виконано
5	Тестування та експериментальна перевірка роботи	26.10.2025	05.11.2025	виконано
6	Виконання економічних розрахунків	05.11.2025	15.11.2025	виконано
7	Оформлення пояснювальної записки та графічних матеріалів	15.11.2025	25.11.2025	виконано

Студент _____ Володимир ПАНАСЮК

Керівник роботи _____ Ігор ШТЕЛЬМАХ

АНОТАЦІЯ

УДК 004.94:502.3(477.44)

Панасюк В. Р. Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм міста Вінниці. Магістерська кваліфікаційна робота зі спеціальності 126 – інформаційні системи та технології, освітньо-професійна програма – інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2025. 118 с.

На укр. мові. Бібліогр.: 20 назва; рис.: 19; табл.: 8.

В магістерській кваліфікаційній роботі проаналізовано сучасний стан моніторингу водних ресурсів та розроблено інформаційну систему для автоматизованого збору, обробки та візуалізації даних якості води в реальному часі з використанням технологій IoT, .NET 8 та SignalR. Об’єктом досліджень є процес автоматизованого моніторингу та управління даними про екологічний стан водних ресурсів.

Ілюстративна частина складається з 17 плакатів із результатами проєктування та розробки системи.

У розділі економічної частини розглянуто питання про доцільність розробки та впровадження інформаційної системи моніторингу, розраховано витрати та термін окупності проєкту.

Ключові слова: інформаційна система, екологічний моніторинг, якість води, IoT, Clean Architecture, візуалізація даних, SignalR.

ABSTRACT

Panasiuk V. R. Information system for analysis and visualization of monitoring data of the state of natural water bodies of the city of Vinnytsia. Master's thesis in specialty 126 - information systems and technologies, educational and professional program - information technology data and image analysis. Vinnytsia: VNTU, 2025. 118 p.

In Ukrainian language. Bibliogr .: 20 titles; fig .: 19; table: 8.

The master's qualification work analyzes the current state of water resources monitoring and develops an information system for automated collection, processing, and visualization of water quality data in real-time using IoT, .NET 8, and SignalR technologies. The object of research is the process of automated monitoring and data management regarding the ecological state of water resources.

The illustrative part consists of 17 posters with the results of analysis of of system design and development.

In the section of the economic part, the question of the expediency of development and implementation of the monitoring information system is considered, costs and the payback period of the project are calculated.

Key words: information system, ecological monitoring, water quality, IoT, Clean Architecture, data visualization, SignalR.

ЗМІСТ

ВСТУП

Актуальність теми. Проблема забруднення природних водойм є критичною для екологічної безпеки України, зокрема для міста Вінниці, де річка Південний Буг є основним джерелом водопостачання. Існуючі методи державного моніторингу, що базуються на періодичному ручному відборі проб, не забезпечують необхідної оперативності для виявлення пікових забруднень та аварійних скидів у режимі реального часу . Впровадження технологій Інтернету речей (IoT) дозволяє вирішити цю проблему шляхом безперервного контролю параметрів води. Проте на ринку відсутні доступні універсальні інформаційні системи, які б поєднували гнучкість налаштування різнорідних датчиків, сучасні стандарти безпеки та можливості візуалізації даних у реальному часі, що робить розробку такої системи актуальним науково-прикладним завданням.

Мета і завдання роботи. Метою роботи є підвищення ефективності моніторингу стану вод шляхом створення сучасної інформаційної системи, що забезпечує автоматизований збір, надійне збереження, аналіз та оперативну візуалізацію показників якості води.

Для досягнення мети вирішено такі задачі:

- проаналізувати предметну область та існуючі рішення для моніторингу водних ресурсів;
- обґрунтувати вибір архітектурного підходу Clean Architecture та технологічного стеку;
- спроєктувати базу даних та програмну архітектуру системи з урахуванням вимог до безпеки та гнучкості;
- розробити програмне забезпечення серверної та клієнтської частин, включаючи модулі автентифікації, динамічного мапінгу даних та сповіщень.

Об’єктом дослідження є процес автоматизованого моніторингу та управління даними про екологічний стан водних ресурсів.

Предметом дослідження є методи та інформаційні технології проєктування розподілених систем збору, обробки та візуалізації телеметричних даних у реальному часі.

Методи дослідження. У роботі використано методи системного аналізу для формування вимог; принципи об’єктно-орієнтованого проєктування та патерн Clean Architecture для побудови архітектури ПЗ; методи реляційних баз даних для організації зберігання інформації; технології асинхронного програмування та веб-сокети (SignalR) для передачі даних у реальному часі.

Новизна одержаних результатів. Удосконалено метод інтеграції IoT-пристроїв в інформаційні системи екологічного моніторингу шляхом реалізації механізму динамічного мапінгу даних, що дозволяє підключати сенсори з довільною структурою даних без зміни програмного коду.

Набуло подальшого розвитку застосування архітектури Clean Architecture для систем екологічного спрямування, що забезпечує незалежність бізнес-логіки від інфраструктурних компонентів та підвищує надійність системи.

Практичне значення Розроблена інформаційна система є повністю функціональним програмним продуктом, готовим до впровадження у комунальних підприємствах та громадських організаціях для оперативного контролю стану водойм, виявлення аномалій та прийняття управлінських рішень.

Апробація результатів магістерської кваліфікаційної роботи. Результати роботи будуть представлені на LV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2026).

Публікації результатів магістерської кваліфікаційної роботи. Опубліковано тези на LV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2026) [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ПОБУДОВИ СИСТЕМ ЕКОЛОГІЧНОГО МОНІТОРИНГУ

1.1 Характеристика стану водних ресурсів та проблеми їх моніторингу в сучасних умовах

Водні ресурси є не лише стратегічним природним активом будь-якої держави, але й фундаментальною основою існування біосфери, що визначає можливість сталого розвитку суспільства та економіки. Для України, яка згідно з класифікацією ООН належить до країн з відносно обмеженими запасами прісної води на душу населення, питання збереження, раціонального використання та відновлення водних об'єктів набуває особливої гостроти. Природні водойми — річки, озера, водосховища та підземні водоносні горизонти — виконують цілий ряд критично важливих функцій, що виходять далеко за межі простого забезпечення потреб у питній воді. Вони є транспортними артеріями, джерелами енергії для гідроелектростанцій, середовищем існування для унікальних екосистем, а також виконують незамінну рекреаційну та естетичну роль для населення [2].

Для міста Вінниці та Вінницької області головною водною артерією та джерелом життя є річка Південний Буг. Вона виступає безальтернативним джерелом централізованого питного водопостачання для обласного центру, що накладає виняткову відповідальність на органи місцевого самоврядування та екологічні служби щодо контролю її екологічного стану. Сабарівське водосховище, розташоване в межах міста, акумулює запаси води, які після відповідної очистки подаються споживачам. Проте, в умовах інтенсивної урбанізації, неконтрольованого розростання приміських зон, зростання промислового виробництва та активного використання агрохімікатів у сільському господарстві, водні об'єкти регіону зазнають безпрецедентного антропогенного тиску. Неконтрольовані або недостатньо очищені скиди стічних вод від промислових підприємств, змив азотних та фосфорних добрив

із полів під час злив, незаконна забудова прибережних захисних смуг — усе це призводить до поступової, але неухильної деградації якості води.

Екологічний стан басейну Південного Бугу характеризується низкою системних проблем, які вимагають постійного контролю та реагування:

- Евтрофікація водойм: Надмірне збагачення води біогенними елементами (сполуками азоту та фосфору) провокує бурхливий ріст фітопланктону та вищої водної рослинності. Це призводить до масового розмноження синьо-зелених водоростей, «цвітіння» води, зниження її прозорості та появи неприємного запаху.
- Гіпоксія: Внаслідок розкладу надмірної біомаси водоростей відбувається інтенсивне споживання розчиненого у воді кисню, що призводить до його дефіциту (гіпоксії) та може спричинити масовий мор риби та інших гідробіонтів, особливо у літній період.
- Хімічне забруднення: Потрапляння у воду нафтопродуктів, синтетичних поверхнево-активних речовин (СПАР), фенолів та важких металів створює пряму загрозу токсичного отруєння водних організмів та погіршує якість питної води [4].

Оцінка екологічного стану водойм є складним, багатофакторним процесом, що базується на комплексному аналізі широкого спектру гідрофізичних, гідрохімічних та гідробіологічних показників. Серед фізичних параметрів першочергове значення мають температура, прозорість (або каламутність) та електропровідність. Температурний режим води є фундаментальним фактором, що визначає швидкість протікання всіх хімічних реакцій та біологічних процесів у водоймі. Підвищення температури внаслідок скидів підігрітих вод (теплова забрудненість) або глобальних кліматичних змін призводить до зменшення розчинності газів, зокрема кисню, та прискорення розвитку патогенної мікрофлори. Каламутність, яка вимірюється в нефелометричних одиницях (NTU), свідчить про наявність у воді завислих часток органічного та неорганічного походження, які можуть бути носіями адсорбованих токсичних речовин.

Хімічні показники якості води є найбільш інформативними індикаторами забруднення та дозволяють точно визначити характер впливу на екосистему. Водневий показник (pH) визначає кислотно-лужний баланс водного середовища і впливає на токсичність багатьох речовин та їхню здатність мігрувати. Вміст розчиненого кисню (Dissolved Oxygen, DO) є критичним показником життєздатності водної екосистеми; його зниження нижче рівня 4 мг/дм³ є сигналом тривоги для екологів. Концентрація сполук азоту (амонійний азот, нітрити, нітрати) та фосфору (фосфати) є прямим індикатором органічного забруднення комунальними стоками або змивами з сільськогосподарських угідь.

Таблиця 1.1 – Основні показники якості води та їх нормативні значення для рибогосподарських водойм

Показник	**Одиниця виміру**	**ГДК (Норматив)**	**Наслідки перевищення (або відхилення)**
Розчинений кисень	мгO_2/дм³	Не менше 6,0 (літній період) Не менше 4,0 (зимовий період)	Критичний дефіцит кисню (гіпоксія), що призводить до явищ задухи та масового мору риби та інших гідробіонтів.
Водневий показник (pH)	од. pH	6,5 – 8,5	Порушення кислотно-лужного балансу, пошкодження зябер та шкірних покривів риб, підвищення токсичності важких металів та аміаку.
Амоній-іони	мг/дм³	0,5	Висока токсичність для риб (особливо мальків), ураження центральної нервової системи гідробіонтів, порушення обміну речовин.
Нітрит-іони	мг/дм³	0,08	Утворення метгемоглобіну в крові риб, що блокує перенесення кисню, спричиняючи токсичне отруєння навіть при нормальному вмісті кисню у воді.
Нітрат-іони	мг/дм³	40,0	Хоча менш токсичні, ніж нітрити, високі концентрації свідчать про давнє органічне забруднення та пригнічують імунну систему риб.

Продовження таблиці 1.1

Показник	**Одиниця виміру**	**ГДК (Норматив)**	**Наслідки перевищення (або відхилення)**
Фосфат-іони	мг/дм³	3,5 (для поліфосфатів)	Основний фактор евтрофікації («цвітіння» води). Надлишок провокує бурхливий ріст синьо-зелених водоростей, що вторинно забруднюють воду токсинами.
БСК-5 (Біохімічне споживання кисню)	мгO_2/дм³	Не більше 3,0	Свідчить про високий вміст органічних речовин, що легко окислюються. Призводить до швидкого вичерпання запасів кисню у водоймі.
Завислі речовини (каламутність)	мг/дм³	Збільшення не більше ніж на 0,25 - 0,75 (від фону)	Погіршення світлопроникності води (зниження фотосинтезу), замулення дна та ікри, механічне пошкодження зябер.

Відповідно до чинного законодавства України, зокрема Водного кодексу та Постанови Кабінету Міністрів України про порядок здійснення державного моніторингу вод, моніторинг є складовою частиною державної системи моніторингу довкілля. Його основною метою є збір, обробка, збереження, узагальнення та аналіз інформації про стан водних об’єктів, прогнозування його змін та розробка науково обґрунтованих рекомендацій для прийняття управлінських рішень [3]. Проте, існуюча система державного моніторингу, яка базується переважно на традиційних лабораторних методах контролю, має низку системних недоліків, що суттєво знижують її ефективність у сучасних динамічних умовах.

Головним і найбільш критичним недоліком традиційної системи є низька дискретність вимірювань у часі та просторі. Відбір проб здійснюється вручну мобільними групами з суворо регламентованою періодичністю (наприклад, щомісяця, щокварталу або навіть рідше) у чітко визначених створах спостереження. Такий підхід забезпечує високу метрологічну точність визначення хімічного складу конкретної проби завдяки використанню сертифікованого високоточного лабораторного обладнання (спектрофотометрів, хроматографів) та стандартних методик виконання вимірювань. Однак, він абсолютно не дозволяє фіксувати динамічні,

короткочасні зміни якості води, які можуть відбуватися протягом годин або навіть хвилин [4].

Проблема полягає в тому, що значна частина забруднень водних об'єктів має характер так званих «залпових скидів». Недобросовісні промислові підприємства або комунальні служби часто здійснюють несанкціоновані скиди концентрованих забруднюючих речовин у нічний час, вихідні дні або під час інтенсивних опадів, коли змив забруднюючих речовин з міських територій є максимальним, розраховуючи на відсутність контролю з боку екологічної інспекції в цей час. Традиційний лабораторний моніторинг фізично не здатний зафіксувати ці короткочасні пікові навантаження. Проба води, відібрана через кілька днів або тижнів після події, може показати цілком нормативні значення показників внаслідок процесів розбавлення, течії та природного самоочищення річки, хоча шкода біоценозу вже була завдана.

Відсутність оперативної інформації в режимі реального часу призводить до ряду негативних наслідків для управління водними ресурсами:

1) Неможливість швидкого реагування: Екологічні служби та водоканали дізнаються про забруднення із запізненням, коли вжити превентивних заходів вже неможливо.
2) Складність ідентифікації джерел: Важко довести вину конкретного підприємства, якщо забруднююча пляма вже пройшла створ спостереження.
3) Недостатність даних для прогнозування: Розріджені масиви даних не дозволяють будувати точні математичні моделі поведінки екосистеми та прогнозувати її реакцію на зміни клімату.

У зв'язку з цим виникає нагальна необхідність модернізації системи екологічного моніторингу шляхом впровадження автоматизованих засобів вимірювання на базі технологій Інтернету речей (Internet of Things, IoT). Такі системи здатні забезпечити безперервний потік даних у режимі 24/7, що дозволить отримати повну картину динаміки якості води.

1.2 Аналіз існуючих інформаційних систем, платформ та підходів до автоматизації екологічного контролю

Сучасний етап розвитку інформаційного суспільства характеризується стрімкою цифровізацією всіх сфер життєдіяльності, включаючи екологічний менеджмент та моніторинг навколишнього природного середовища. Ринок програмних та апаратних рішень пропонує широкий спектр інструментарію, що варіюється від найпростіших аматорських датчиків до складних державних геоінформаційних систем національного масштабу. Для обґрунтованого визначення функціональних та нефункціональних вимог до розроблюваної системи, а також задля уникнення дублювання вже існуючих рішень, було проведено глибокий порівняльний аналіз наявних аналогів. Умовно їх можна класифікувати на три великі групи, кожна з яких займає свою нішу: офіційні державні портали, універсальні IoT-платформи комерційного або відкритого типу, та спеціалізовані громадські системи моніторингу.

Безперечним лідером та найбільш вагомим джерелом даних у сфері моніторингу водних ресурсів в Україні є «Система моніторингу та екологічної оцінки водних ресурсів», що адмініструється Державним агентством водних ресурсів України. Ця система являє собою масштабний веб-портал, який акумулює результати лабораторних досліджень з розгалуженої мережі басейнових лабораторій моніторингу вод та ґрунтів по всій території країни [5]. Ключовою та беззаперечною перевагою даної системи, яка вигідно відрізняє її від будь-яких інших аналогів, є офіційний юридичний статус даних. Вимірювання проводяться виключно атестованими лабораторіями з використанням стандартизованих, метрологічно повірених методик, які гармонізовані з європейськими стандартами, зокрема з вимогами Водної рамкової директиви ЄС. Це гарантує високу точність та достовірність показників, що дозволяє використовувати їх для формування державної екологічної політики, накладання штрафних санкцій на підприємства-забруднювачі та підготовки офіційних звітів про стан довкілля. Інтерфейс

порталу дозволяє переглядати дані в розрізі річкових басейнів, суббасейнів та водогосподарських ділянок, що є зручним для професійних гідрологів та державних службовців.

Проте, незважаючи на свою фундаментальність та метрологічну точність, державна система має низку критичних недоліків у контексті потреб оперативного реагування на екологічні загрози. Основним бар'єром є значна часова затримка (так званий «time lag») між моментом фізичного відбору проби води та моментом публікації верифікованих результатів на веб-порталі. Цей складний адміністративний процес, що включає транспортування зразків до лабораторії, проведення тривалого хімічного аналізу (який для деяких показників, наприклад БСК-5, регламентовано триває 5 діб), перевірку протоколів та ручне внесення даних у базу, може займати від кількох тижнів до місяця [4]. В умовах виникнення аварійних ситуацій, таких як несанкціоновані залпові скиди промислових стоків або аварії на очисних спорудах, така затримка є абсолютно неприпустимою, оскільки критично важлива інформація стає доступною громадськості та органам місцевого самоврядування вже тоді, коли забруднююча пляма пройшла водозабори або розчинилася, завдавши незворотної шкоди екосистемі.

Крім того, державна система є архітектурно закритою та консервативною. Вона не передбачає сучасних механізмів для автоматизованої інтеграції даних (наприклад, публічного API) для сторонніх розробників або науковців, що ускладнює створення похідних аналітичних продуктів та мобільних застосунків. Також на сьогодні відсутня технічна та організаційна можливість підключення до державної системи потоків даних з автоматизованих постів моніторингу, встановлених муніципалітетами, комунальними підприємствами, громадськими організаціями або відповідальним бізнесом. Це штучно обмежує щільність покриття моніторингової мережі лише офіційними створами, залишаючи значні «сліпі зони» на малих річках та у місцях потенційного техногенного впливу, що є

суттєвою проблемою для таких міст, як Вінниця, де малі річки часто стають колекторами стічних вод.

На противагу спеціалізованим та регламентованим державним рішенням, на світовому ринку широко представлені універсальні хмарні платформи для Інтернету речей (IoT), яскравим прикладом яких є платформа ThingSpeak від компанії MathWorks. Ця платформа здобула значну популярність у світі, особливо в освітньому та науково-дослідницькому середовищі, завдяки своїй доступності та простоті інтеграції з широким спектром мікроконтролерів та апаратних платформ, таких як Arduino, ESP8266/ESP32, Raspberry Pi [6]. ThingSpeak надає розробникам готовий інструментарій для збору, хмарного зберігання, візуалізації та первинного аналізу даних, що надходять з датчиків через стандартні протоколи HTTP або MQTT. Важливою конкурентною перевагою платформи є вбудована підтримка потужного математичного пакету MATLAB, що дозволяє виконувати складну статистичну обробку даних, фільтрацію шумів, інтерполяцію пропусків та візуалізацію результатів безпосередньо на стороні сервера, не навантажуючи обчислювальні потужності кінцевих пристроїв або клієнтських додатків.

Однак, при спробі побудови на базі ThingSpeak повноцінної регіональної системи екологічного моніторингу, розробники неминуче стикаються з низкою суттєвих обмежень, які роблять це рішення менш привабливим для професійного використання у довгостроковій перспективі. По-перше, безкоштовна версія платформи накладає досить жорсткі ліміти на частоту оновлення даних (зазвичай не частіше одного запиту на 15 секунд) та кількість каналів, що суттєво ускладнює масштабування системи при збільшенні кількості точок спостереження. Для розгортання мережі з десятків або сотень датчиків необхідна покупка комерційної ліцензії, вартість якої може бути значною для бюджетних установ або неприбуткових громадських організацій. По-друге, і що є більш критичним недоліком для корпоративних систем, ThingSpeak не має розвиненої системи управління користувачами та гнучкого

розмежування прав доступу (Role-Based Access Control - RBAC). У базовій архітектурі платформи дані каналу є або повністю публічними для всього інтернету, або приватними, доступними лише власнику облікового запису через спеціальний API-ключ. Це фактично унеможливлює створення ієрархічної структури доступу, яка є необхідною для муніципальних систем, де потрібно чітко розділяти права адміністраторів (які налаштовують пристрої), науковців-аналітиків (які мають доступ до «сирих» даних та звітів) та широкої громадськості (яка має право лише на перегляд агрегованої інформації на картах). Крім того, інтерфейс платформи є технічно орієнтованим і важко піддається кастомізації (white-labeling) під бренд міста або специфічні потреби замовника, що знижує його цінність як кінцевого продукту для непідготовлених користувачів.

Третім важливим вектором розвитку екологічного моніторингу є поява та стрімке поширення громадських систем, таких як відомі українські проєкти EcoCity та SaveEcoBot. Ці ініціативи стали справжнім феноменом останніх років, успішно реалізувавши концепцію «громадянської науки» (Citizen Science) на практиці. Вони змогли побудувати масштабні розподілені мережі моніторингу якості повітря та радіаційного фону, залучивши до цього процесу тисячі активних волонтерів, які встановлюють недорогі станції моніторингу у своїх домівках, офісах та на підприємствах. Ці системи пропонують надзвичайно зручні, сучасні мобільні додатки та веб-інтерфейси, орієнтовані на пересічного громадянина, з інтуїтивно зрозумілою інфографікою (наприклад, кольорове кодування рівня небезпеки), надійною системою сповіщень про перевищення гранично допустимих концентрацій та інтеграцією з популярними месенджерами (чат-боти в Telegram, Viber). Вони також послідовно дотримуються принципів відкритості даних (Open Data), надаючи безперешкодний доступ до своїх архівів через публічні API, що стимулює незалежні наукові дослідження та розробку сторонніх сервісів [7].

Проте, незважаючи на їхній беззаперечний успіх, пряме перенесення досвіду та технологічних рішень EcoCity або SaveEcoBot на сферу

моніторингу водних ресурсів стикається зі специфічними проблемами предметної області. Архітектура, протоколи обміну даними та алгоритми аналізу цих систем історично оптимізовані насамперед під специфіку моніторингу атмосферного повітря, зокрема вимірювання концентрації дрібнодисперсного пилу (PM2.5, PM10), газів (NO2, CO, SO2) та радіаційного фону. Датчики повітря є відносно автономними і не вимагають частого складного обслуговування. Натомість, автоматизований моніторинг якості води має свою унікальну і складну специфіку. Занурені у воду датчики (pH, розчинений кисень, електропровідність, каламутність) піддаються інтенсивному впливу агресивного водного середовища та явищу біообрастання (утворення біологічної плівки з водоростей та бактерій на чутливих елементах сенсора), що призводить до швидкого дрейфу показників і необхідності регулярного фізичного очищення та калібрування за еталонними розчинами.

Існуючі платформи громадського моніторингу часто не надають необхідної функціональної гнучкості для управління життєвим циклом водних сенсорів, фіксації фактів їх технічного обслуговування та програмної корекції калібрувальних коефіцієнтів у часі. Крім того, гідрологічні процеси поширення забруднень мають зовсім іншу просторово-часову динаміку порівняно з атмосферними процесами, що вимагає інших підходів до візуалізації та аналізу трендів (наприклад, врахування напрямку та швидкості течії річки для прогнозування руху плями забруднення, чого немає в «точкових» моделях моніторингу повітря).

У таблиці 1.2 наведено узагальнений порівняльний аналіз розглянутих підходів, що дозволяє наочно оцінити їх сильні та слабкі сторони в контексті поставленого завдання.

Таблиця 1.2 – Порівняльний аналіз функціональних можливостей існуючих систем моніторингу

Критерій порівняння	**Державна система (Держводагентство)**	**Універсальні платформи (ThingSpeak)**	**Громадські системи (EcoCity)**	**Власна розробка (IoT-System)**
Джерело даних	Сертифіковані лабораторії	Будь-які IoT-пристрої	Громадські станції (повітря)	Гетерогенні джерела (IoT + лабораторії)
Оперативність (Latency)	Низька (затримка дні/місяці)	Висока (Real-time)	Висока (Real-time)	Висока (Real-time, SignalR)
Гнучкість налаштування	Відсутня (жорсткий регламент)	Середня (API, обмежені поля)	Низька (стандартні пристрої)	Висока (Динамічний мапінг JSONB)
Управління доступом (RBAC)	Закрита відомча система	Обмежене (API Key)	Публічний доступ	Розширена рольова модель (Admin, User, Scientist)
Спеціалізація	Водні ресурси	Універсальна	Атмосферне повітря	Водні ресурси (з урахуванням калібрування)
Вартість впровадження	Висока (бюджетне фінансування)	Середня/Висока (залежить від ліцензії)	Низька	Низька (Open Source стек)

Таким чином, проведений комплексний аналіз дозволяє зробити обґрунтований висновок про наявність вільної ніші на ринку інформаційних технологій екологічного спрямування. Існує об'єктивна та нагальна потреба у створенні спеціалізованої інформаційної системи, яка б гармонійно поєднувала в собі переваги всіх розглянутих підходів, але була позбавлена їх критичних недоліків. Зокрема, така система має бути архітектурно відкритою для підключення різнорідних IoT-пристроїв (на відміну від консервативних державних систем); гнучкою в налаштуванні параметрів моніторингу та мапінгу даних (що перевершує можливості жорстко регламентованих платформ); мати розвинену систему безпеки та рольового доступу (на відміну від простих IoT-конструкторів); та бути адаптованою саме до специфіки водного моніторингу з урахуванням особливостей обслуговування гідрохімічних датчиків. Саме ці вимоги лягли в основу концепції розроблюваної системи IoT-System, яка покликана забезпечити надійний,

оперативний та доступний інструментарій для контролю стану водних ресурсів міста Вінниці та регіону в цілому [4, 8].

1.3 Огляд та обґрунтування архітектурних принципів проєктування сучасних IoT-систем

Процес проєктування архітектури програмного забезпечення для систем екологічного моніторингу, що базуються на концепції Інтернету речей (IoT), є фундаментальним етапом розробки, який визначає не лише поточну працездатність комплексу, але і його життєздатність, гнучкість та здатність до еволюційного розвитку в довгостроковій перспективі. Специфіка таких систем полягає у необхідності обробки гетерогенних, високочастотних потоків даних, що надходять від різноманітних сенсорів, забезпеченні високої доступності сервісів для кінцевих користувачів та здатності до горизонтального масштабування при розширенні мережі моніторингу. Вибір неоптимального архітектурного патерну на початкових етапах проєктування неминуче призводить до накопичення так званого «технічного боргу», коли внесення будь-яких, навіть незначних змін у функціонал або додавання підтримки нового обладнання стає надмірно трудомістким, ризикованим та дорогим процесом через сильну зв'язність компонентів системи. У сучасній інженерії програмного забезпечення виділяють кілька домінуючих парадигм побудови веб-орієнтованих систем, серед яких найбільш дискусійними є монолітна архітектура, мікросервісна архітектура та різні варіації шаруватих (багатоланкових) архітектур, зокрема «Чиста архітектура» (Clean Architecture).

Традиційна монолітна архітектура (Monolithic Architecture) історично була стандартом де-факто для розробки веб-застосунків. У рамках цього підходу всі функціональні модулі системи — інтерфейс користувача, бізнес-логіка, доступ до даних, модулі автентифікації та обробки подій — об'єднані в єдину кодову базу, компілюються разом та розгортаються як один цілісний виконуваний файл або сервіс. Безперечними перевагами такого підходу на

старті проєкту є простота розробки, легкість налагодження (дебагінгу), відсутність накладних витрат на міжпроцесну комунікацію та проста процедура розгортання (деплою). Однак, у контексті складних IoT-систем, які характеризуються високою динамікою змін вимог та технологічного стеку, класичний моноліт швидко демонструє свої обмеження. Феномен тісної пов'язаності компонентів (tight coupling) призводить до того, що зміна реалізації одного модуля, наприклад, оновлення бібліотеки протоколу обміну даними з датчиком, може непередбачувано вплинути на роботу зовсім інших частин системи, скажімо, модуля генерації звітності. Крім того, масштабування моноліту можливе лише шляхом дублювання всього додатку на кількох серверах, що є вкрай неефективним використанням обчислювальних ресурсів, якщо вузьким місцем (bottleneck) є лише один специфічний компонент, що відповідає за обробку вхідного потоку даних [10].

Діаметрально протилежною альтернативою є мікросервісна архітектура (Microservices), яка передбачає декомпозицію системи на набір невеликих, автономних сервісів, кожен з яких виконує чітко окреслену бізнес-функцію (Bounded Context) та спілкується з іншими через легковажні мережеві протоколи (зазвичай HTTP/REST або черги повідомлень типу RabbitMQ чи Kafka). Такий підхід забезпечує максимальну гнучкість, дозволяючи використовувати різні мови програмування та технології зберігання даних для різних підзадач, а також незалежно масштабувати окремі сервіси відповідно до навантаження. Проте, для проєкту рівня магістерської кваліфікаційної роботи або стартапу на етапі створення мінімально життєздатного продукту (MVP), мікросервіси вносять надлишкову, часто невиправдану складність в інфраструктуру. Необхідність реалізації надійних механізмів виявлення сервісів (service discovery), балансування навантаження, забезпечення узгодженості розподілених транзакцій та налаштування складного моніторингу вимагає високої кваліфікації команди та значних часових витрат, що може відволікати від вирішення основної прикладної задачі — екологічного моніторингу водних ресурсів.

Враховуючи необхідність знайти оптимальний баланс між структурованістю коду, гнучкістю, тестопридатністю та розумною складністю реалізації, для даної роботи було обрано та обґрунтовано використання архітектурного стилю Clean Architecture («Чиста архітектура»), систематизованого та популяризованого Робертом Мартіном (відомим як Uncle Bob). Цей підхід, який концептуально перегукується з ідеями «Гексагональної архітектури» (Ports and Adapters) Алістера Кокберна та «Цибулевої архітектури» (Onion Architecture) Джеффрі Палермо, базується на фундаментальному принципі інверсії залежностей (Dependency Inversion Principle). Головна ідея полягає у розділенні програмної системи на концентричні шари, де правило залежностей є суворим і однозначним: залежності вихідного коду завжди повинні бути спрямовані виключно від зовнішніх шарів (деталі реалізації, інфраструктура) до внутрішніх (бізнес-правила, політики). Така структура гарантує, що ядро системи залишається «чистим» і незалежним від зовнішніх фреймворків, конкретних баз даних, інтерфейсів користувача та сторонніх сервісів, що дозволяє замінювати ці компоненти без необхідності переписування бізнес-логіки [11].

Структурна схема обраного архітектурного підходу, що ілюструє поділ на шари та напрямок залежностей, наведена на рисунку 1.1.

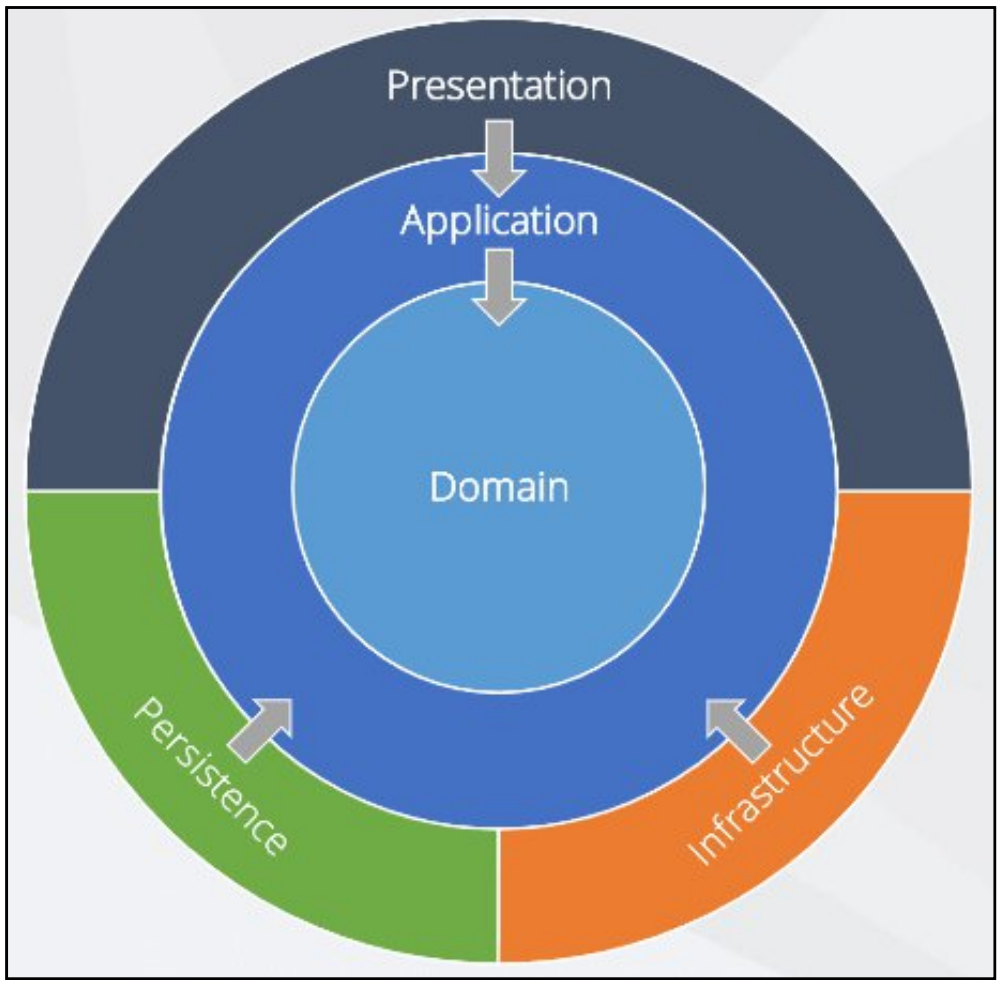


Рисунок 1.1 – Концептуальна схема рівнів «Чистої архітектури» із відображенням правила залежностей

У геометричному центрі архітектури знаходиться шар Домену (Domain Layer). Цей шар є найбільш стабільним компонентом системи, який містить сутності (Entities) — програмні об'єкти, що моделюють реальні поняття предметної області. У контексті нашої системи це такі класи як «Пристрій» (Device), «Вимірювання» (Measurement), «Користувач» (User), «Повідомлення про тривогу» (Alert). Доменний шар інкапсулює найзагальніші бізнес-правила та логіку, яка є істинною незалежно від того, чи використовується система через веб-сайт, мобільний додаток чи командний рядок. Критично важливою характеристикою цього шару є те, що він не має жодних залежностей від інших проєктів у рішенні, не містить посилань на бібліотеки доступу до даних або веб-фреймворки, що робить його абсолютно портативним і придатним для повторного використання.

Безпосередньо навколо домену розташовується шар Застосунку (Application Layer). Цей рівень містить сценарії використання (Use Cases) та специфічні бізнес-правила конкретного додатку. Шар застосунку виступає в ролі оркестратора, який керує потоками даних між сутностями та зовнішнім світом, забезпечуючи виконання конкретних завдань користувача, таких як «Зареєструвати новий датчик», «Отримати історію вимірювань за період» або «Перевірити перевищення порогових значень». Саме тут визначаються інтерфейси (абстракції) для доступу до даних (Repository Interfaces) та зовнішніх сервісів, але не їхня конкретна реалізація. Наприклад, шар застосунку може декларувати інтерфейс IMeasurementRepository з методом Save(), але він «не знає», чи будуть дані збережені в реляційну базу SQL, NoSQL документ або звичайний текстовий файл. Це дозволяє відкласти прийняття важливих технологічних рішень на пізніші етапи розробки.

Зовнішні шари архітектури — Інфраструктура (Infrastructure) та Презентація (Presentation/API) — містять деталі реалізації та механізми взаємодії із зовнішнім світом. Шар інфраструктури відповідає за технічну реалізацію інтерфейсів, визначених у шарі застосунку. Тут розміщується код для роботи з конкретною системою управління базами даних (наприклад, конфігурація контексту Entity Framework Core для PostgreSQL), клієнти для відправки електронної пошти, драйвери для роботи з файловою системою та адаптери для взаємодії зі сторонніми API. Шар презентації відповідає за взаємодію з користувачами або клієнтськими програмами. У розробленій системі це реалізовано через набір контролерів Web API, які приймають HTTP-запити, виконують валідацію вхідних даних, викликають відповідні сценарії з шару застосунку та конвертують отриманий результат у форматі DTO (Data Transfer Object) для передачі клієнту (зазвичай у форматі JSON).

Імплементація Clean Architecture надає розроблюваній системі екологічного моніторингу ряд стратегічних переваг. По-перше, забезпечується повна незалежність від фреймворків та баз даних. Рішення про міграцію з MS SQL Server на PostgreSQL, яке було прийнято в процесі роботи над

магістерською дисертацією для оптимізації роботи з JSON-даними, вимагало змін лише в одному шарі (Інфраструктура), залишивши бізнес-логіку та API контролери абсолютно недоторканими. По-друге, досягається високий рівень тестопридатності (testability). Оскільки бізнес-правила не залежать від інтерфейсу користувача чи бази даних, їх можна легко покрити автоматичними модульними тестами (Unit Tests) без необхідності підняття важкого оточення або розгортання бази даних, використовуючи мок-об'єкти (mocks) для імітації зовнішніх залежностей. Це значно підвищує надійність системи та впевненість розробника при внесенні змін та рефакторингу. По-третє, чітка структурованість коду полегшує навігацію проектом та введення нових розробників у команду, оскільки кожен клас має своє чітко визначене місце та зону відповідальності, що запобігає ентропії коду та перетворенню системи на неструктуровану масу.

1.4 Аналіз та обґрунтування вибору технологічного стеку для реалізації системи

Процес вибору технологічного стеку для розробки сучасної інформаційної системи є одним із найкритичніших етапів життєвого циклу проєкту, оскільки прийняті на цьому етапі рішення визначають не лише швидкість та вартість початкової реалізації, але й, що значно важливіше, здатність продукту до масштабування, його надійність, безпеку та вартість подальшої підтримки (Total Cost of Ownership). Враховуючи специфіку даної магістерської роботи, яка полягає у створенні системи екологічного моніторингу на базі технологій Інтернету речей (IoT), до програмного забезпечення висувається низка жорстких нефункціональних вимог. Серед них ключовими є здатність обробляти високочастотні потоки гетерогенних даних від великої кількості розподілених сенсорів, забезпечувати мінімальну затримку (latency) при візуалізації показників у реальному часі та гарантувати цілісність накопиченої історії спостережень. На основі проведеного аналізу

архітектурних патернів та особливостей предметної області було сформовано комплексний технологічний стек, що охоплює серверну платформу (Backend), систему управління базами даних (Database), клієнтську частину (Frontend) та протоколи обміну даними.

Першим стратегічним рішенням став вибір платформи для реалізації серверної частини (Backend), яка виступає ядром системи та відповідає за бізнес-логіку, обробку даних та взаємодію з базою даних. У сучасному веб-девелопменті домінуючими платформами є Node.js, Python (Django/FastAPI), Java (Spring) та .NET. Хоча Node.js демонструє високу ефективність у I/O-bound задачах завдяки своїй асинхронній природі, а Python є стандартом де-факто у сфері Data Science, для даного проєкту було обрано платформу .NET 8 (мова програмування C#). Цей вибір був зумовлений комплексом наступних факторів:

- Висока продуктивність: .NET 8, завдяки вдосконаленому JIT-компілятору (Just-In-Time) та оптимізованому механізму управління пам'яттю (Garbage Collection), стабільно входить до топу найшвидших веб-фреймворків у світі, значно випереджаючи інтерпретовані мови у сценаріях обробки HTTP-запитів.
- Кросплатформність: Це рішення з відкритим вихідним кодом дозволяє розгортати серверне ПЗ на операційних системах Linux, що суттєво знижує витрати на хостинг порівняно з традиційними Windows-серверами.
- Надійність та типізація: C# є мовою із суворою статичною типізацією, що дозволяє виявляти більшість потенційних помилок ще на етапі компіляції, а вбудований у платформу контейнер впровадження залежностей (Dependency Injection) ідеально узгоджується з принципами Clean Architecture [12].

Наступним критичним етапом став вибір системи управління базами даних (СУБД). У попередніх прототипах системи, розроблених на етапі бакалаврської роботи, використовувалася СУБД Microsoft SQL Server. Проте, аналіз вимог до гнучкості моделі даних змусив переглянути це рішення на користь об'єктно-реляційної СУБД PostgreSQL. Головним аргументом на

користь міграції стала унікальна для реляційних баз даних підтримка типу даних JSONB (Binary JSON). Специфіка IoT-систем полягає у високій гетерогенності даних: різні моделі датчиків від різних виробників можуть передавати абсолютно різний набір параметрів (наприклад, один пристрій вимірює лише температуру, інший — температуру, pH та каламутність, третій додатково надсилає службову інформацію про заряд батареї). Використання класичної жорсткої реляційної схеми, де кожному параметру відповідає окремий стовпець, призвело б до необхідності постійної зміни схеми бази даних (Schema Migration) при додаванні кожного нового типу пристрою, що є ризикованим процесом на працюючій системі.

Переваги використання PostgreSQL у даному проєкті:

- Гнучкість структури: Технологія JSONB дозволяє зберігати напівструктуровані дані телеметрії у вигляді документів всередині однієї таблиці, забезпечуючи при цьому швидкість читання, порівнянну з NoSQL рішеннями.
- Геопросторова аналітика: Наявність розширення PostGIS, яке надає потужні інструменти для роботи з геопросторовими даними, відкриває можливості для реалізації складних запитів, наприклад, пошуку джерел скидів у радіусі дії водозабору.
- Економічна ефективність: PostgreSQL є повністю безкоштовним продуктом з відкритим кодом, що знижує загальну вартість володіння системою (TCO) [13].

Для реалізації клієнтської частини (Frontend) було обрано бібліотеку React у поєднанні з мовою TypeScript. Цей вибір обумовлений необхідністю створення швидкодіючого, інтерактивного односторінкового додатку (SPA — Single Page Application), який би забезпечував користувачам досвід, максимально наближений до роботи з нативними десктопними програмами. React використовує концепцію віртуального DOM (Virtual DOM), що дозволяє мінімізувати кількість "дорогих" операцій перерисування сторінки в браузері, забезпечуючи плавне оновлення інтерфейсу навіть при високій частоті

надходження нових даних від сенсорів, що є критичним для дашбордів моніторингу. Компонентний підхід React дозволяє створювати модульний код, який легко підтримувати та розширювати. Використання TypeScript додає до гнучкості JavaScript строгу типізацію, що є необхідним для забезпечення надійності клієнтського коду та спрощення взаємодії з типізованим API бекенду (використання спільних DTO-моделей). Для візуалізації часових рядів екологічних показників було обрано спеціалізовану бібліотеку Recharts, яка оптимізована для роботи в екосистемі React і дозволяє будувати адаптивні графіки з підтримкою масштабування та деталізації [15].

Критично важливою вимогою до системи моніторингу є здатність відображати зміни екологічної ситуації в режимі реального часу (Real-time). Класичний підхід HTTP, що базується на моделі «запит-відповідь», не є ефективним для таких задач, оскільки вимагає постійного періодичного опитування сервера клієнтом (polling). Це створює надлишкове навантаження на мережу та серверні ресурси, а також вносить штучну затримку в оновленні даних. Для вирішення цієї проблеми було впроваджено технологію SignalR, яка є високорівневою абстракцією над протоколом WebSockets для платформи .NET. SignalR дозволяє встановити постійне двонаправлене з'єднання між сервером та браузером клієнта. Це дає можливість серверу ініціювати передачу даних (push-повідомлення) миттєво після отримання нового пакету телеметрії від датчика та його збереження в базі даних. Такий підхід забезпечує субсекундну затримку відображення інформації, що є критичним для систем оперативного реагування на аварійні ситуації [14].

Питання безпеки доступу до даних та API було вирішено шляхом впровадження стандарту JSON Web Token (JWT). На відміну від традиційної сесійної автентифікації, де стан сесії зберігається в оперативній пам'яті сервера, JWT дозволяє реалізувати stateless-архітектуру (без збереження стану). Це означає, що сервер не зобов'язаний зберігати інформацію про активні сесії, що значно спрощує горизонтальне масштабування системи шляхом додавання нових серверів за балансувальником навантаження. Токен

містить всю необхідну інформацію про користувача та його права (claims), підписану криптографічним ключем, що гарантує захист від підробки. Для запобігання атакам типу XSS (Cross-Site Scripting) було реалізовано механізм зберігання токенів оновлення (Refresh Tokens) у захищених HttpOnly Cookies, які недоступні для читання клієнтськими скриптами, забезпечуючи високий рівень захисту облікових записів [16].

Для наочної ілюстрації взаємодії обраних технологій та їх місця в загальній структурі системи розроблено узагальнену схему технологічного стеку, яка представлена на рисунку 1.2.

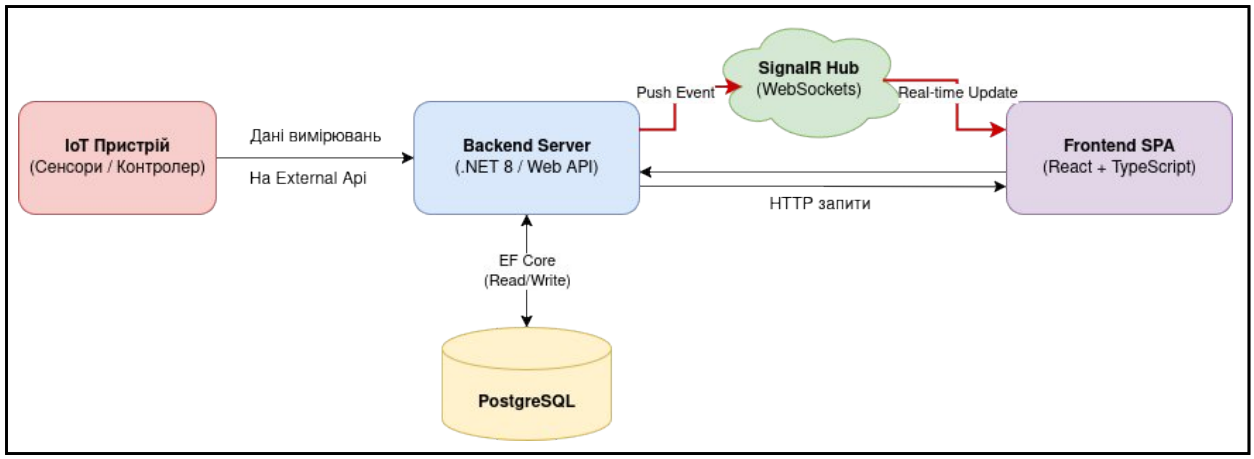


Рисунок 1.2 – Узагальнена схема технологічного стеку та потоків даних в інформаційній системі

На схемі відображено повний шлях проходження даних: від генерації первинних показників IoT-пристроями, їх передачі через захищений канал на Backend-сервер під управлінням .NET 8, збереження у структурованому сховищі PostgreSQL та миттєвої ретрансляції через SignalR Hub на клієнтські інтерфейси React для візуалізації кінцевим користувачам. Такий підхід забезпечує цілісність, безпеку та високу швидкість обробки інформації на всіх етапах її життєвого циклу.

1.5 Узагальнення результатів аналізу та концептуалізація рішення

Для наочного узагальнення логіки дослідження та взаємозв'язку виявлених проблем із запропонованими технологічними рішеннями розроблено концептуальну схему, наведену на рисунку 1.3.

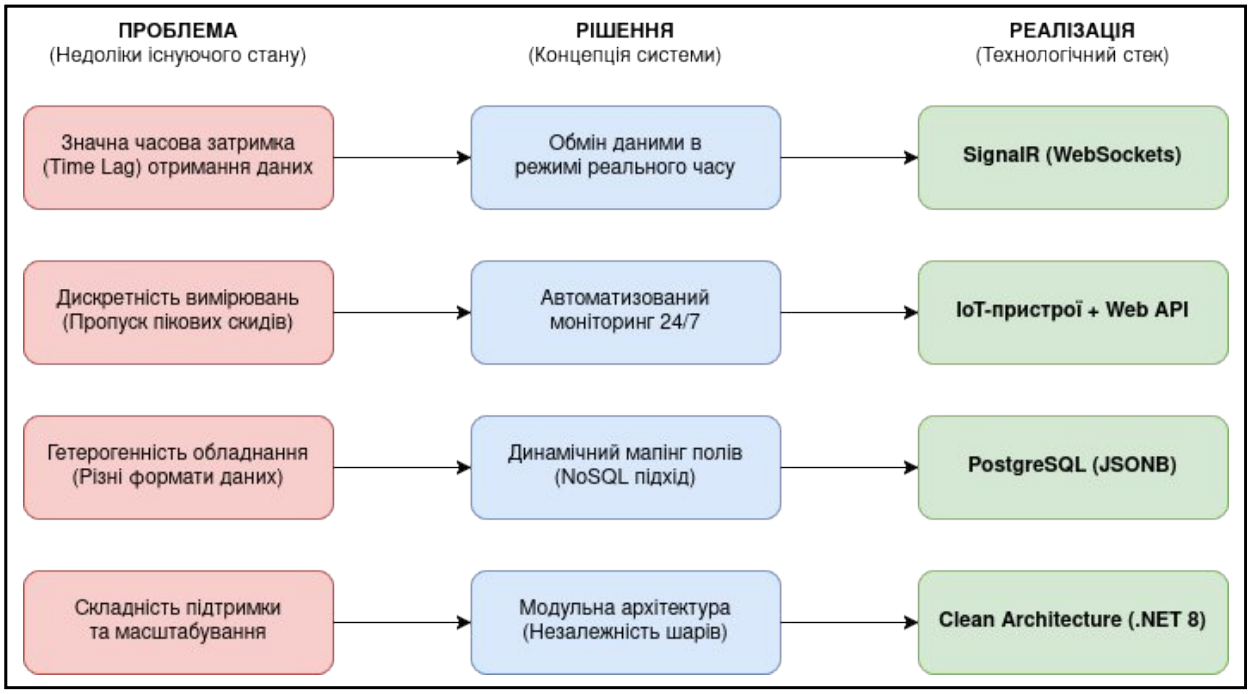


Рисунок 1.3 – Концептуальна схема переходу від проблематики предметної області до технологічних рішень

Підсумовуючи результати проведеного аналізу, можна сформулювати основні концептуальні засади подальшого проєктування:

1) Цілеспрямованість: Система повинна фокусуватися на вирішенні проблеми оперативного виявлення забруднень шляхом обробки потокових даних.

2) Відкритість: Архітектура має дозволяти легке підключення нових типів пристроїв через механізм динамічного мапінгу.

3) Надійність: Використання сучасних стандартів безпеки (JWT) та відмовостійких технологій зберігання даних є обов'язковим.

4) Ефективність: Технологічний стек повинен забезпечувати мінімальну затримку при візуалізації даних.

Отримані результати аналізу формують надійне теоретичне та методологічне підґрунтя для переходу до наступного етапу роботи — безпосереднього проєктування архітектури та компонентів інформаційної системи.

1.6 Висновки

У першому розділі проведено комплексний аналіз предметної області та виявлено критичну невідповідність між динамікою забруднення р. Південний Буг та інерційністю існуючих методів державного моніторингу. Встановлено, що дискретний ручний відбір проб не дозволяє фіксувати пікові забруднення та аварійні скиди, що обґрунтовує необхідність переходу до автоматизованого моніторингу в режимі реального часу (Real-time) на базі IoT-технологій [4, 9].

Критичний огляд аналогів (державні портали, ThingSpeak, EcoCity) засвідчив відсутність універсального продукту, який би поєднував гнучкість налаштування гетерогенних датчиків, відкритість архітектури та сучасні стандарти безпеки. Виявлені функціональні обмеження існуючих систем стали підґрунтям для прийняття рішення про розробку власної інформаційної системи, яка заповнить цю нішу [6, 7].

Науково обґрунтовано вибір архітектурного стилю Clean Architecture для забезпечення масштабованості та тестопридатності системи. У якості технологічного фундаменту обрано платформу .NET 8 та бібліотеку React. Для ефективного зберігання різнорідних даних визначено СУБД PostgreSQL з використанням типу JSONB, а для миттєвої доставки даних користувачам — технологію SignalR [12, 14].

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА КОМПОНЕНТІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Розробка загальної архітектури системи та діаграм взаємодії компонентів

Етап проєктування архітектури є фундаментальним у життєвому циклі розробки будь-якої складної інформаційної системи, оскільки саме на цьому етапі закладаються основи для забезпечення її надійності, масштабованості та здатності до еволюційного розвитку. Враховуючи результати аналізу предметної області, проведеного у першому розділі, та виявлену необхідність створення гнучкого рішення, незалежного від конкретних фреймворків чи баз даних, за основу проєктування було взято архітектурний стиль Clean Architecture («Чиста архітектура»). Цей підхід, формалізований Робертом Мартіном, базується на принципі розділення відповідальності та інверсії залежностей (Dependency Inversion Principle), що дозволяє створити систему, в якій бізнес-логіка є ядром, ізольованим від деталей реалізації, таких як інтерфейс користувача, зовнішні API або система управління базами даних [11].

Головною метою застосування Clean Architecture у даному проєкті є мінімізація технічного боргу та забезпечення можливості легкої заміни або модернізації окремих компонентів без впливу на загальну стабільність системи. Архітектура розробленої системи IoT-System складається з чотирьох концентричних шарів, взаємодія між якими суворо регламентована правилом залежностей: залежності вихідного коду можуть вказувати лише всередину, у напрямку до високорівневих політик.

Першим і найважливішим рівнем є Шар Домену (Domain Layer). Це серце системи, яке містить сутності (Entities), об'єкти-значення (Value Objects), перелічувані типи (Enums) та винятки (Exceptions), що описують предметну область екологічного моніторингу. Сутності інкапсулюють у собі найбільш

загальні та високорівневі правила бізнесу, які залишаються незмінними незалежно від того, як змінюються зовнішні технології. У проєкті виділено такі ключові сутності, як Device (Пристрій), що представляє фізичний сенсор або станцію моніторингу; Measurement (Вимірювання), що зберігає часові ряди показників якості води; User (Користувач), що описує суб'єктів системи та їх права доступу; та AlertRule (Правило сповіщення), яке визначає умови виникнення тривожних подій. Критично важливою особливістю цього шару є його повна незалежність: він не має посилань на жодні інші проєкти в рішенні, не залежить від бібліотек доступу до даних (Entity Framework) чи веб-фреймворків (ASP.NET Core), що робить його ідеальним для написання модульних тестів (Unit Tests) [17].

Навколо доменного шару розташовується Шар Застосунку (Application Layer). Цей рівень виступає в ролі оркестратора, який керує потоками даних між інтерфейсом користувача та доменними об'єктами. Він містить визначення інтерфейсів (Interfaces) для сервісів та репозиторіїв, реалізація яких буде здійснена на зовнішніх рівнях, а також об'єкти передачі даних (DTO — Data Transfer Objects). Саме тут реалізуються сценарії використання системи (Use Cases), такі як «Зареєструвати новий пристрій», «Отримати історію вимірювань за вказаний період» або «Оновити налаштування мапінгу». Шар застосунку відповідає за валідацію вхідних даних, перевірку прав доступу на рівні бізнес-логіки та координацію транзакцій. Він залежить виключно від шару домену і не знає деталей про те, як дані зберігаються в базі або як вони відображаються користувачеві.

Схематичне зображення структури проєкту та взаємозв'язків між шарами наведено на рисунку 2.1.

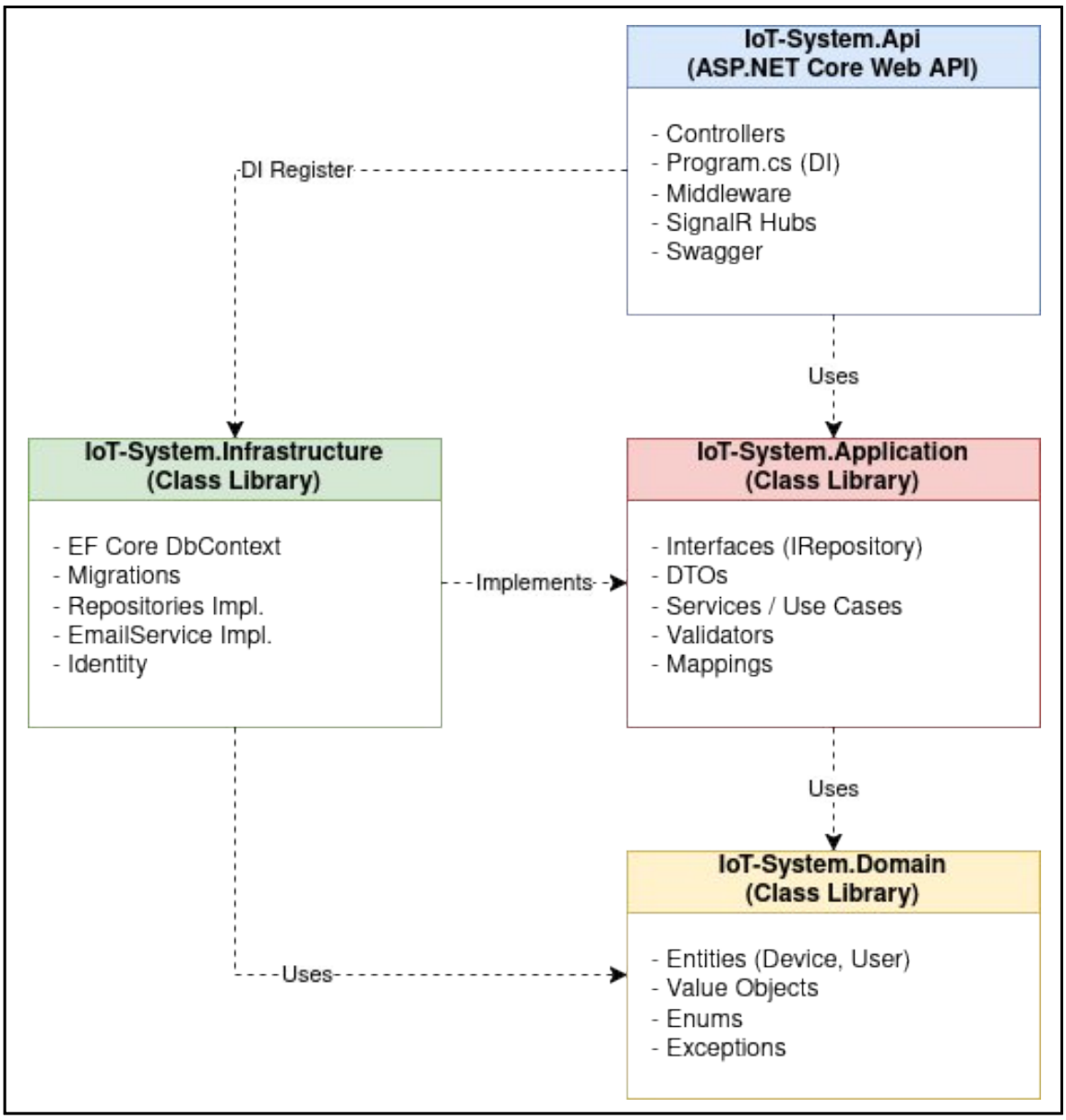


Рисунок 2.1 – Структурна схема архітектури рішення на базі Clean Architecture

Зовнішніми по відношенню до ядра є шари Інфраструктури та Представлення. Шар Інфраструктури (Infrastructure Layer) відповідає за технічну реалізацію інтерфейсів, визначених у шарі застосунку. Тут розміщується код, що забезпечує взаємодію із зовнішнім світом: доступ до бази даних PostgreSQL через ORM Entity Framework Core, реалізація клієнтів для відправки електронної пошти (Email Service), адаптери для роботи з файловою системою або хмарними сховищами. Саме в цьому шарі відбувається конфігурація контексту бази даних, налаштування міграцій та

реалізація патерну «Репозиторій» (Repository Pattern), який абстрагує логіку вибірки даних від бізнес-логіки.

Шар Представлення (Presentation/API Layer) є точкою входу в систему для зовнішніх клієнтів. У розробленій системі він реалізований як набір RESTful контролерів на базі платформи ASP.NET Core Web API. Завданням цього шару є прийняття HTTP-запитів, розпарсинг тіла запиту, виклик відповідних сервісів з шару застосунку та формування коректної HTTP-відповіді (Status Code + JSON Body). Цей шар також містить Middleware — проміжні компоненти для обробки глобальних виключень, логування запитів та налаштування CORS політик. Важливо зазначити, що контролери не містять бізнес-логіки; вони лише делегують виконання завдань шару застосунку, що забезпечує «тонкість» контролерів і спрощує їх тестування.

Для формалізації функціональних вимог та визначення меж системи було розроблено модель прецедентів, яка відображена на діаграмі варіантів використання (Use Case Diagram). Система передбачає розмежування прав доступу на основі рольової моделі (RBAC), де виділено чотири основні актори:

- Гість (Unregistered User): Має обмежений доступ, може лише переглядати сторінки входу, реєстрації, головну, а також переглядати дашборди, графіки та карту з точками моніторингу, але не може вносити зміни в конфігурацію
- Спостерігач (Viewer): Авторизований користувач, який має право переглядати дашборди, графіки та карту з точками моніторингу, але не може вносити зміни в конфігурацію.
- Науковець (Scientist): Користувач з розширеними правами, який може іспортувати «сирі» історичні дані, налаштовувати персональні сповіщення про перевищення порогових значень та вносити зміни в конфігурації пристроїв.
- Адміністратор (Admin): Повноправний керуючий системою, який відповідає за реєстрацію нових пристроїв, налаштування мапінгу полів, управління користувачами своєї групи та моніторинг стану здоров'я системи.

- Головний адміністратор (Super Admin): Повний доступ до всіх частин системи.

Взаємодія цих акторів із основними функціональними модулями системи зображена на рисунку 2.2.

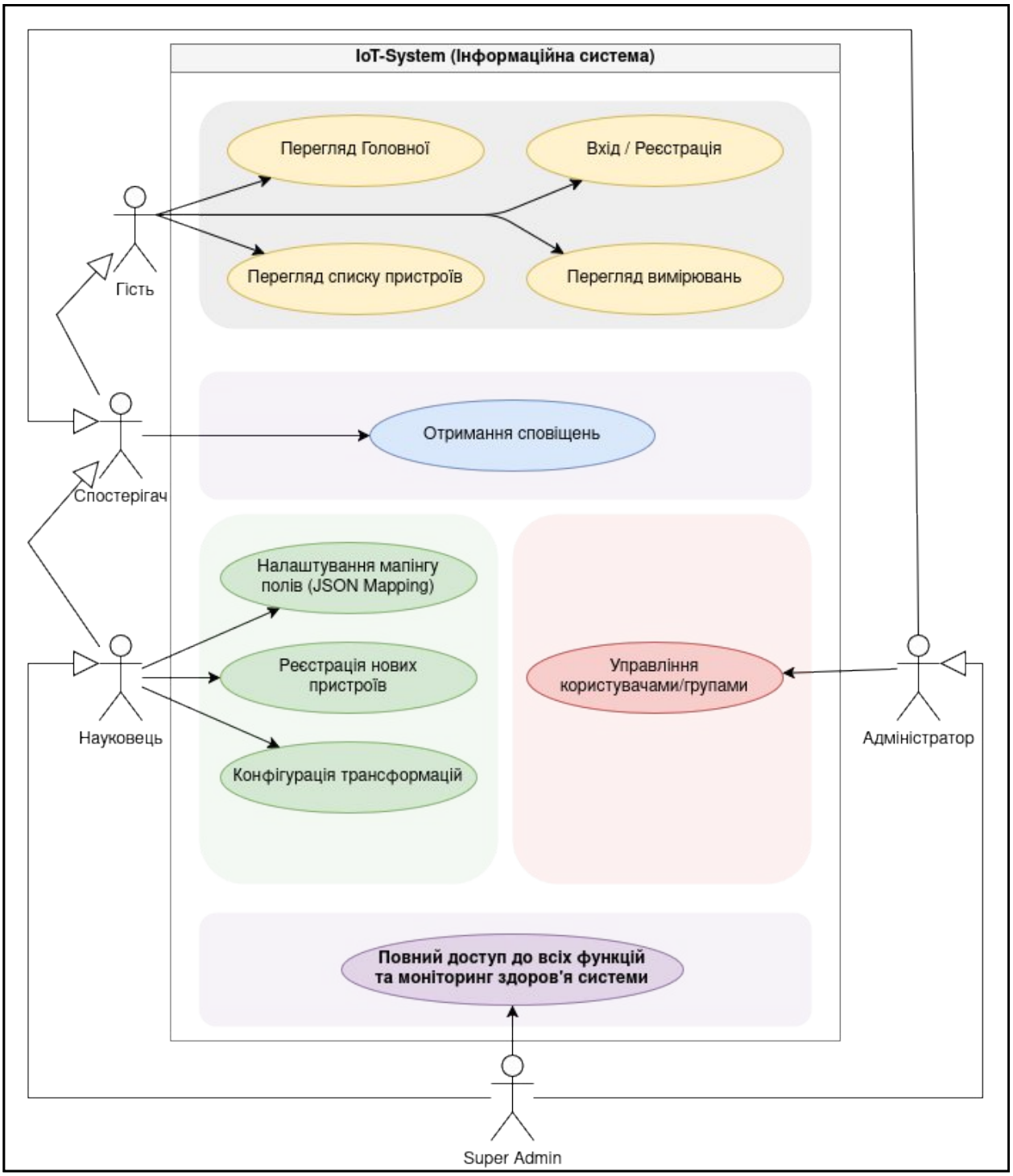


Рисунок 2.2 – Діаграма варіантів використання (Use Case Diagram) інформаційної системи

Окремої уваги заслуговує проєктування динамічної взаємодії компонентів системи при обробці потоків телеметричних даних. Оскільки система працює в режимі, наближеному до реального часу, важливо було забезпечити мінімальну затримку між надходженням даних від сенсора та їх відображенням на клієнті. Процес обробки даних можна описати наступним алгоритмом, який реалізовано через взаємодію компонентів різних шарів:

- IoT-пристрій ініціює HTTP POST запит на ендпоінт API, передаючи JSON-пакет з вимірюваннями.
- API Controller (Presentation Layer) приймає запит та передає DTO (Data Transfer Object) у сервіс обробки вимірювань (Application Layer).
- Сервіс виконує валідацію даних, перевіряє наявність зареєстрованого пристрою за унікальним ключем (API Key) та, використовуючи доменну логіку, конвертує вхідні дані у сутність Measurement.
- Далі сервіс звертається до репозиторію (Infrastructure Layer) для збереження сутності в базі даних PostgreSQL.
- Після успішного збереження сервіс ініціює подію (Domain Event), на яку підписаний SignalR Hub.
- SignalR Hub (Presentation Layer) миттєво транслює оновлені дані всім підключеним через WebSocket клієнтам (веб-браузерам).

Такий підхід забезпечує чітке розділення відповідальності: база даних відповідає за надійне зберігання історії, а підсистема SignalR — за оперативну доставку «гарячих» даних користувачам. Використання асинхронної моделі програмування (async/await) на всіх етапах обробки запиту дозволяє серверу ефективно використовувати потоки виконання, не блокуючи їх під час очікування відповіді від бази даних або мережевих операцій, що є критичним для забезпечення високої пропускної здатності системи під навантаженням [12].

Таким чином, спроєктована архітектура створює надійний фундамент для подальшої реалізації системи. Вона поєднує в собі строгість і впорядкованість Clean Architecture з гнучкістю та продуктивністю сучасних

технологій .NET та PostgreSQL, що дозволяє задовольнити як функціональні вимоги щодо моніторингу якості води, так і нефункціональні вимоги щодо надійності, безпеки та зручності супроводу.

2.2 Проєктування реляційної моделі бази даних PostgreSQL з урахуванням специфіки IoT-даних

Проєктування схеми бази даних є одним із найбільш критичних етапів розробки інформаційної системи, оскільки від правильності прийнятих на цьому етапі рішень залежить продуктивність системи, цілісність даних та зручність їх подальшої обробки. Враховуючи специфіку предметної області, а саме необхідність роботи з великими масивами часових рядів (Time-Series Data), що надходять від різнорідних IoT-пристроїв, до підсистеми зберігання даних висуваються підвищені вимоги щодо швидкості запису (write throughput) та гнучкості структури. У попередніх версіях прототипу системи використовувалася СУБД Microsoft SQL Server із класичною нормалізованою схемою. Однак, у рамках даної магістерської роботи, на основі аналізу технологічного стеку, проведеного у першому розділі, було прийнято стратегічне рішення про міграцію на СУБД PostgreSQL. Це дозволило використати потужні можливості цієї системи для роботи з напівструктурованими даними та геопросторовою інформацією.

Ключовим викликом при проєктуванні схеми даних для IoT-систем є проблема гетерогенності (різлрідності) даних. Різні моделі датчиків від різних виробників можуть передавати абсолютно різний набір параметрів. Наприклад, проста станція моніторингу може передавати лише температуру води, тоді як комплексна станція — температуру, рівень pH, вміст розчиненого кисню, каламутність та електропровідність. Використання класичного підходу «одна колонка — один параметр» (Entity-Attribute-Value або широкі таблиці з безліччю стовпців, що допускають NULL) у реляційній базі даних призвело б до створення неефективної, розрідженої структури, яку важко підтримувати

та масштабувати. Додавання підтримки нового типу датчика вимагало б виконання операції ALTER TABLE для додавання нових стовпців, що на великих таблицях з мільйонами записів є дорогою та блокуючою операцією.

Для вирішення цієї проблеми у спроєктованій схемі бази даних було застосовано гібридний підхід, який поєднує строгість реляційної моделі для метаданих та гнучкість NoSQL для телеметрії. Центральною таблицею, що зберігає інформацію про джерела даних, є таблиця Devices (Пристрої). Вона спроєктована за класичними принципами нормалізації і містить фіксований набір атрибутів, спільних для всіх типів обладнання: унікальний ідентифікатор (Id), назву (Name), текстовий опис (Description), географічні координати (Latitude, Longitude) для відображення на карті, унікальний ключ доступу (UniqueKey), який використовується для автентифікації пристрою при передачі даних, та зовнішній ключ GroupId, що визначає приналежність пристрою до певної групи користувачів або організації.

Зберігання безпосередньо вимірювань реалізовано у таблиці Measurements (Вимірювання). Замість створення окремих стовпців для кожного фізико-хімічного показника, було використано специфічний для PostgreSQL тип даних JSONB (Binary JSON). Структура таблиці включає первинний ключ, зовнішній ключ DeviceId для зв'язку з таблицею пристроїв, мітку часу Timestamp (з точністю до мілісекунд) та поле Data типу JSONB. Саме у полі Data зберігається корисне навантаження від датчика у вигляді JSON-об'єкта (наприклад, {"t": 24.5, "ph": 7.2, "do": 8.4}). Такий підхід забезпечує колосальну гнучкість: система може приймати та зберігати дані будь-якої структури без необхідності зміни схеми бази даних. При цьому, завдяки бінарному формату зберігання, PostgreSQL дозволяє ефективно виконувати запити по ключах всередині JSON-документа, а також будувати GIN-індекси (Generalized Inverted Index) для прискорення пошуку, що робить продуктивність роботи з такими даними співмірною з традиційними стовпцями.

Важливим архітектурним компонентом, що забезпечує інтерпретацію цих «сирих» JSON-даних, є таблиця DeviceValueMappings (Мапінг значень). Ця таблиця реалізує логіку динамічного мапінгу, описану в попередніх розділах. Вона містить правила відповідності між технічними ключами, які надсилає пристрій (наприклад, короткі імена змінних "t", "v1", "o2" для економії трафіку), та зрозумілими для користувача семантичними назвами показників ("Температура води", "Рівень pH", "Розчинений кисень") та їх одиницями виміру ("°C", "од.", "мг/дм³"). Кожен запис у цій таблиці пов'язаний з конкретним пристроєм, що дозволяє індивідуально налаштовувати парсинг даних для кожного сенсора. Також у цій таблиці зберігаються налаштування порогових значень (WarningThreshold, CriticalThreshold), які використовуються підсистемою сповіщень для виявлення аномалій.

Система управління користувачами та безпекою базується на стандартних таблицях технології ASP.NET Core Identity (AspNetUsers, AspNetRoles, AspNetUserRoles), які були інтегровані в загальну схему. Таблиця користувачів була розширена додатковими полями, необхідними для бізнес-логіки, зокрема полем GroupId. Це поле дозволяє реалізувати модель мультитенантності (multitenancy) на рівні бази даних, коли користувачі бачать і керують лише тими пристроями, що належать до їхньої організації або робочої групи.

Для забезпечення гнучкості, безпеки та логічного розмежування відповідальності архітектуру зберігання даних було розділено на дві незалежні схеми. Перша схема відповідає за підсистему ідентифікації, автентифікації та управління організаційною структурою (користувачі, ролі, групи). Друга схема призначена безпосередньо для зберігання бізнес-сутностей, конфігурацій пристроїв та масивів телеметричних даних. Такий підхід дозволяє незалежно масштабувати та обслуговувати кожну з частин системи.

Наочно спроєктовані реляційні моделі представлені на діаграмах: Рисунок 2.3 відображає схему підсистеми управління доступом, а Рисунок 2.4 — схему предметної області моніторингу.

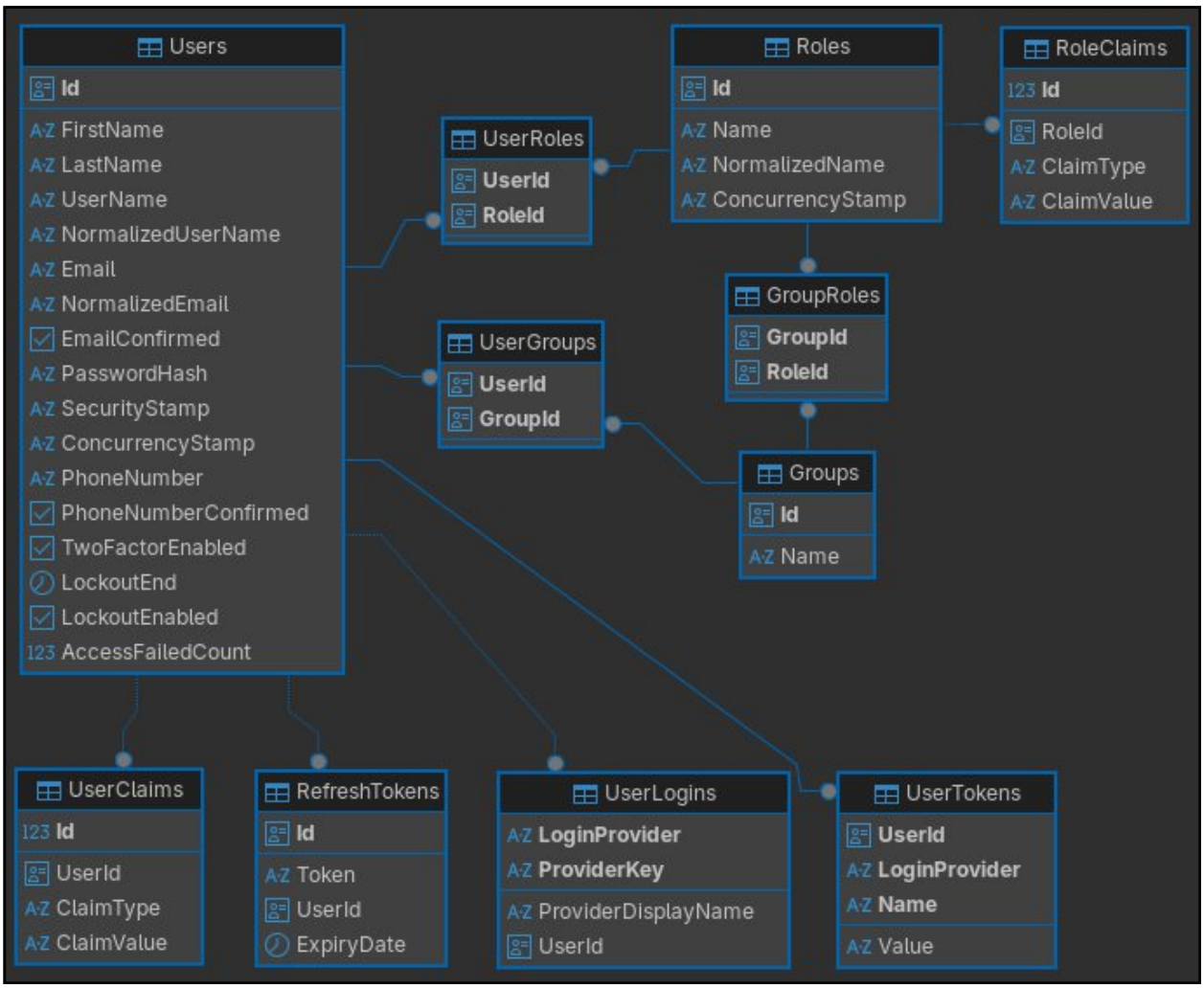


Рисунок 2.3 – ER-діаграма бази даних інформаційної системи ідентифікації та доступу

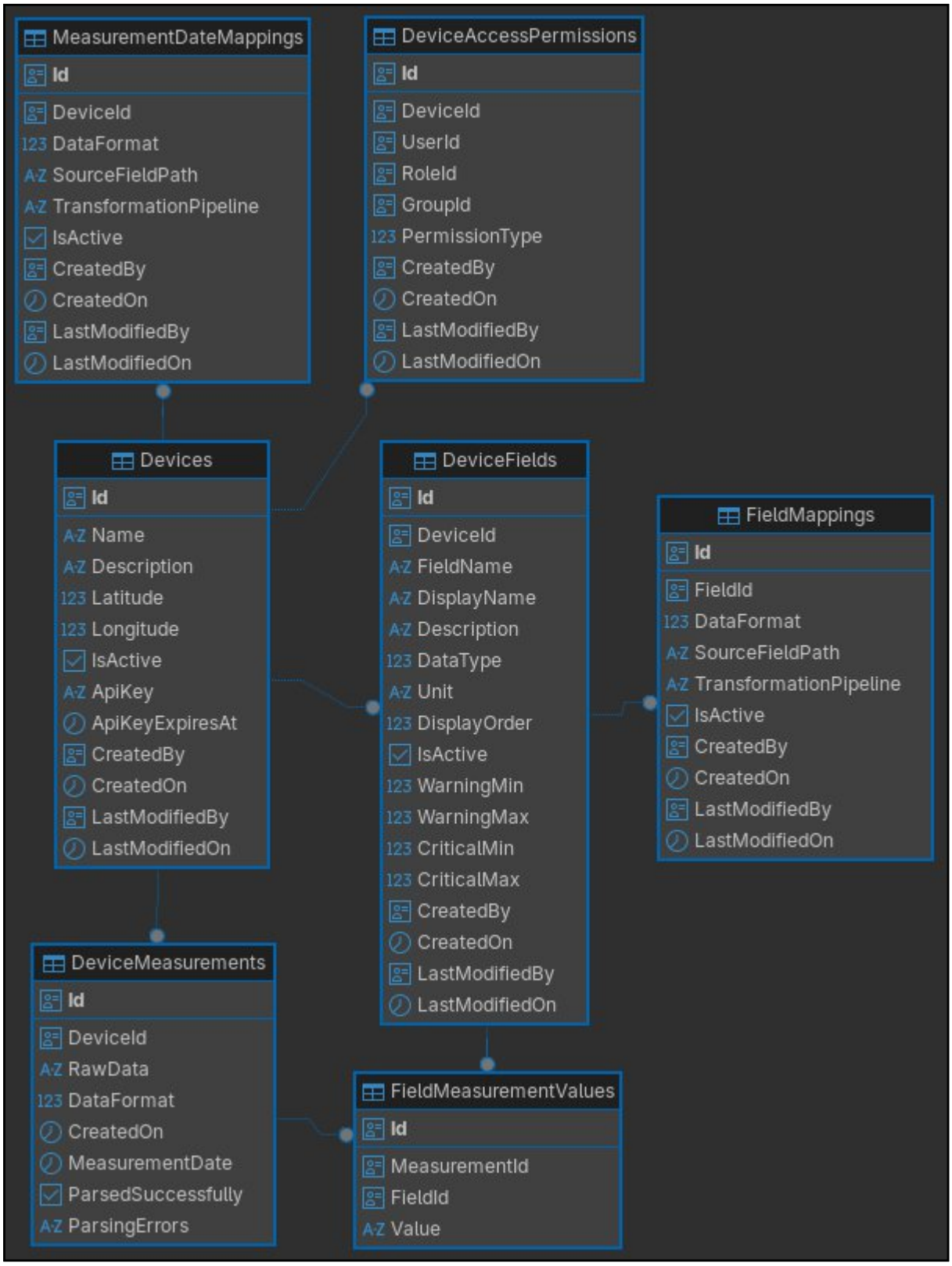


Рисунок 2.4 – ER-діаграма бази даних інформаційної системи предметної області моніторингу

Таким чином, запропонована структура баз даних поєднує надійність стандартизованих таблиць для зберігання критично важливих метаданих

користувачів з гнучкістю документо-орієнтованого підходу для зберігання телеметрії. Це дозволяє системі ефективно працювати з гетерогенними даними, легко масштабуватися та адаптуватися до нових типів обладнання без необхідності складних міграцій структури БД.

2.3 Проєктування комплексної підсистеми безпеки, автентифікації та контролю доступу

У сучасних розподілених інформаційних системах, особливо тих, що працюють з даними екологічного моніторингу, які можуть мати критичне значення для прийняття управлінських рішень та оповіщення населення, питання інформаційної безпеки виходять на перший план. Забезпечення конфіденційності, цілісності та доступності даних є не просто технічною вимогою, а необхідною умовою довіри до системи з боку користувачів та стейкхолдерів. У попередніх версіях прототипу системи (на етапі бакалаврської роботи) механізми безпеки були реалізовані на базовому рівні, що не відповідає сучасним стандартам захисту від кіберзагроз. У рамках даної магістерської роботи було спроєктовано та теоретично обґрунтовано комплексну підсистему безпеки, яка охоплює процеси ідентифікації, автентифікації, авторизації та захисту сесій користувачів.

Фундаментальним елементом системи безпеки є механізм автентифікації, тобто перевірки справжності користувача. Для реалізації цього механізму було обрано відкритий промисловий стандарт JSON Web Token (JWT), описаний у RFC 7519. На відміну від класичної сесійної автентифікації (Cookie-based Session), де сервер зберігає стан сесії (Session State) в оперативній пам'яті або базі даних, JWT дозволяє створити архітектуру без збереження стану («stateless»). Це означає, що сервер не зобов'язаний зберігати інформацію про активні сесії користувачів. Вся необхідна інформація (ідентифікатор користувача, його роль, час видачі та термін дії токена) зашифрована безпосередньо в самому токені, який підписується

криптографічним ключем сервера (HMAC SHA256 або RSA). Такий підхід є критично важливим для забезпечення масштабованості системи: якщо в майбутньому виникне потреба запустити кілька екземплярів API на різних серверах за балансувальником навантаження, їм не потрібно буде синхронізувати між собою сесії користувачів або звертатися до спільного сховища сесій (наприклад, Redis), що значно спрощує архітектуру та підвищує відмовостійкість [16].

Процес автентифікації реалізовано з використанням надійного механізму подвійних токенів, що є стандартом індустрії для забезпечення балансу між зручністю користувача та безпекою.

1. Access Token (Токен доступу): Це короткоживучий токен (наприклад, з терміном дії 15 хвилин), який використовується клієнтським додатком для доступу до захищених ресурсів API. Він передається в заголовку кожного HTTP-запиту (Authorization: Bearer <token>). Короткий термін життя мінімізує ризики у випадку перехоплення токена зловмисником.
2. Refresh Token (Токен оновлення): Це довгоживучий токен (наприклад, з терміном дії 7 днів або 1 місяць). Він використовується виключно для отримання нової пари токенів (нового Access та нового Refresh), коли термін дії поточного Access Token закінчується. Цей процес відбувається прозоро для користувача, без необхідності повторного введення логіна та пароля.

Особливу увагу при проєктуванні було приділено захисту від поширених веб-атак, зокрема міжсайтового скриптингу (Cross-Site Scripting, XSS). При атаці XSS зловмисник впроваджує шкідливий JavaScript-код на сторінку, який може отримати доступ до даних у локальному сховищі браузера (LocalStorage або SessionStorage). Якщо зберігати токени там, вони можуть бути викрадені. Тому було прийнято архітектурне рішення зберігати Refresh Token у спеціальному HttpOnly Cookie. Цей тип cookie має прапорець HttpOnly, який вказує браузеру, що даний cookie не повинен бути доступним для читання через JavaScript (властивість document.cookie). Він автоматично відправляється браузером лише при запитах до сервера, що робить викрадення

токена через XSS практично неможливим. Для захисту від атак типу CSRF (Cross-Site Request Forgery) використовується механізм SameSite=Strict для cookie, що забороняє браузеру відправляти їх при запитах зі сторонніх сайтів.

Схема процесу безпечної автентифікації зображена на рисунку 2.5.

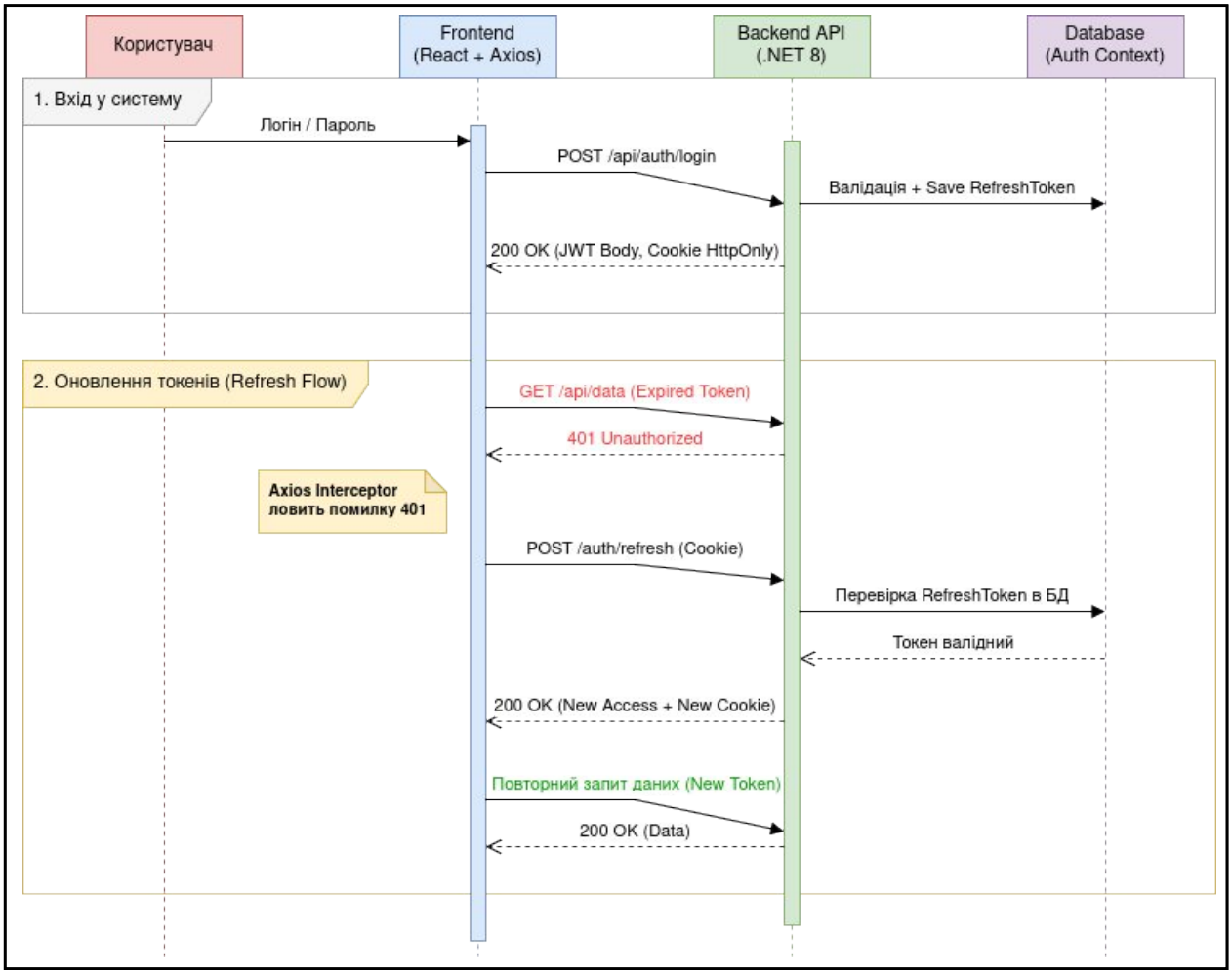


Рисунок 2.5 – Діаграма послідовності процесу безпечної автентифікації користувача (JWT + Refresh Token)

Система авторизації (перевірки прав доступу) базується на рольовій моделі RBAC (Role-Based Access Control). У системі визначено фіксований набір ролей, кожна з яких має чітко окреслений набір дозволів та обмежень:

- SuperAdmin — повний доступ до всіх ресурсів, управління адміністраторами.
- Admin — управління пристроями та користувачами в межах своєї організації.

- Scientist — доступ до аналітичних інструментів та експорту даних.
- Viewer — доступ лише на читання публічних даних.

Перевірка прав здійснюється на двох рівнях захисту (Defense in Depth). Перший рівень — декларативний захист на рівні контролерів API з використанням атрибутів [Authorize(Roles = "...")] платформи .NET. Це дозволяє відхиляти несанкціоновані запити ще до того, як вони потраплять у бізнес-логіку, заощаджуючи ресурси сервера. Другий рівень — імперативна перевірка на рівні сервісів (Resource-based Authorization). Наприклад, навіть якщо користувач має роль «Адміністратор», система на рівні бізнес-логіки перевіряє, чи належить пристрій, який він намагається редагувати, до його організації (GroupId). Це запобігає ситуаціям горизонтального підвищення привілеїв (IDOR — Insecure Direct Object Reference), коли користувач змінює ID ресурсу в URL і отримує доступ до чужих даних.

Такий багаторівневий підхід до проєктування підсистеми безпеки забезпечує надійний захист інформаційної системи від несанкціонованого доступу, витоку даних та компрометації облікових записів, відповідаючи сучасним вимогам кібербезпеки для веб-додатків.

2.4 Розробка моделі та алгоритмів динамічного мапінгу і конфігурації IoT-пристроїв

Однією з найскладніших та найбільш актуальних проблем при проєктуванні універсальних інформаційних систем для Інтернету речей є проблема гетерогенності (різнорідності) обладнання. Ринок сучасних засобів вимірювання пропонує величезний спектр датчиків, контролерів та шлюзів від сотень різних виробників. Кожен з них використовує власні протоколи передачі даних, формати серіалізації та структури повідомлень. Наприклад, один бюджетний мікроконтролер може надсилати дані у форматі JSON вигляду {"temp": 24.5, "hum": 60}, професійна метеостанція може використовувати скорочення {"t": 24.5, "h": 60, "p": 1013}, а кастомний

пристрій на базі Arduino може передавати масив безіменних значень [24.5, 60]. У класичному підході до розробки («Hardcoding») це вимагало б написання окремого програмного адаптера (парсера) для кожного нового типу пристрою, що робить систему вкрай негнучкою, збільшує час виведення нових пристроїв в експлуатацію та ускладнює підтримку кодової бази.

Для вирішення цієї фундаментальної проблеми у рамках даної магістерської роботи було розроблено та теоретично обґрунтовано модель Динамічного мапінгу полів (Dynamic Field Mapping). Суть запропонованого підходу полягає у зміні парадигми обробки даних: замість того, щоб адаптувати код системи під кожен пристрій, система «навчається» розуміти структуру даних пристрою на основі метаданих (конфігурації), які задаються адміністратором через графічний інтерфейс без втручання у програмний код. Цей підхід базується на концепції віртуалізації даних, де фізична структура збереження інформації відділена від її логічного представлення для користувача.

Реалізація цієї моделі стала можливою завдяки використанню специфічних можливостей обраної СУБД PostgreSQL, зокрема типу даних JSONB. Алгоритм роботи підсистеми збору та нормалізації даних можна представити як послідовність наступних етапів. На першому етапі (Data Ingestion) IoT-пристрій ініціює передачу даних на єдиний універсальний ендпоінт API системи (/api/data). При цьому пристрій не зобов'язаний дотримуватися жодної жорсткої схеми, окрім наявності валідного ключа авторизації (UniqueKey). Тіло запиту може містити довільний JSON-об'єкт. Система приймає цей пакет і зберігає його «як є» (as-is) у спеціальному полі RawData таблиці вимірювань. Це забезпечує повну збереженість оригінальних даних, що є критично важливим для можливого майбутнього аудиту або повторної обробки (re-processing) у разі зміни алгоритмів розрахунків [13].

На другому етапі (Mapping Lookup) відбувається інтерпретація даних. Коли користувач (або внутрішній сервіс аналітики) запитує дані для візуалізації, система звертається до таблиці конфігурацій

DeviceValueMappings. Для кожного зареєстрованого пристрою в цій таблиці зберігається набір правил трансформації. Правило являє собою зв'язку: SensorKey (ключ у вхідному JSON, наприклад, "v1") -> DisplayName (семантична назва, наприклад, "Розчинений кисень") -> Unit (одиниця виміру, наприклад, "мг/дм³"). Також тут можуть задаватися коефіцієнти лінійної трансформації (y=kx+b) для калібрування датчиків на льоту, якщо вони мають систематичну похибку.

На третьому етапі (Normalization & Validation) відбувається динамічне перетворення. Сервісний шар системи витягує значення з «сирого» JSON-документа, використовуючи ключі з конфігурації, застосовує калібрувальні коефіцієнти та формує уніфіковану модель даних для фронтенду. Саме на цьому етапі відбувається перевірка значень на відповідність допустимим діапазонам (Thresholds). Для кожного параметра адміністратор може задати нижню та верхню межі норми (наприклад, pH не нижче 6.5 і не вище 8.5). Якщо отримане значення виходить за ці межі, система автоматично генерує подію «Тривога» (Alert), яка обробляється окремим сервісом сповіщень.

Графічне представлення розробленого алгоритму обробки даних від моменту надходження до візуалізації наведено на рисунку 2.6.

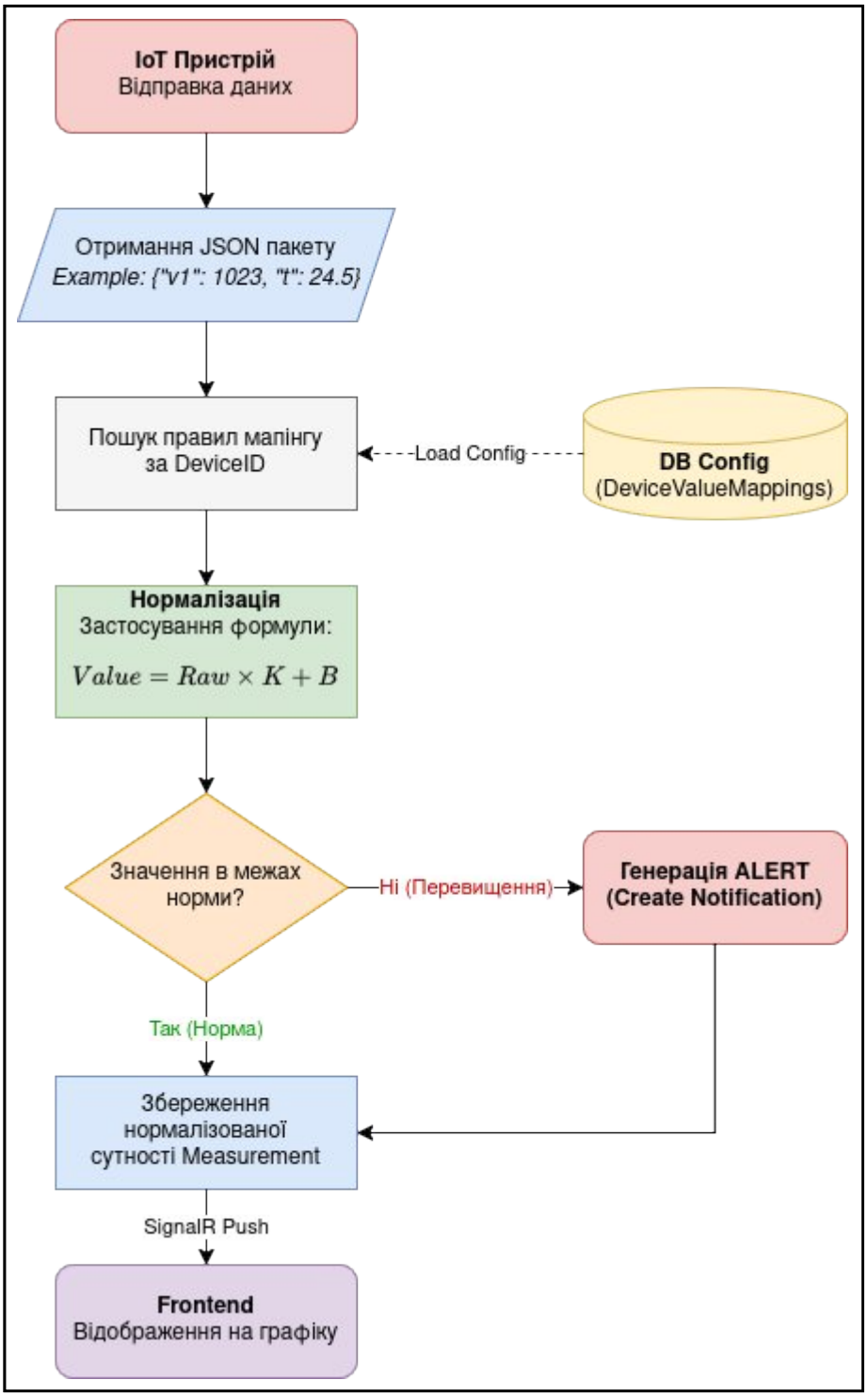


Рисунок 2.6 – Алгоритм динамічної обробки та нормалізації даних від IoT-пристроїв

Такий підхід надає системі ряд стратегічних переваг. По-перше, забезпечується універсальність: система здатна працювати з будь-яким обладнанням, що підтримує стек протоколів TCP/IP та формат JSON, незалежно від його виробника. По-друге, досягається висока відмовостійкість:

зміна формату даних на стороні одного пристрою не призводить до збою всієї системи, а вимагає лише оновлення налаштувань мапінгу в адміністративній панелі. По-третє, це відкриває можливості для «гарячого» калібрування: якщо лабораторний аналіз показує, що датчик почав «дрейфувати», оператор може внести поправку в конфігурацію, і всі наступні дані будуть автоматично скориговані без необхідності перепрошивки самого мікроконтролера, що часто є складним завданням для польового обладнання.

Окрім мапінгу значень, розроблена модель конфігурації дозволяє керувати метаданими відображення. Адміністратор може налаштувати колір графіка для кожного показника, тип діаграми (лінійна, стовпчикова) та пріоритет відображення на дашборді. Всі ці налаштування зберігаються у реляційній базі даних і завантажуються клієнтським додатком при старті, що дозволяє динамічно будувати інтерфейс користувача (UI) на основі даних (Data-Driven UI), повністю відповідаючи принципам гнучкості, закладеним у технічному завданні.

2.5 Висновки

У другому розділі магістерської кваліфікаційної роботи проведено комплексне проєктування архітектури та ключових компонентів інформаційної системи моніторингу природних водойм, що є фундаментом для подальшої програмної реалізації. Результати проєктування базуються на вимогах, сформованих у першому розділі, та враховують специфіку обробки гетерогенних потоків даних від розподіленої мережі IoT-пристроїв.

Першим важливим результатом стало розроблення загальної архітектури системи на основі патерну Clean Architecture («Чиста Архітектура»). Цей вибір дозволив створити чітку, модульну структуру програмного забезпечення, де бізнес-логіка (ядро системи) повністю ізольована від зовнішніх фреймворків, баз даних та інтерфейсів користувача. Така архітектура забезпечує високу стійкість системи до змін, спрощує тестування окремих компонентів та дозволяє легко модернізувати технологічний стек у майбутньому без ризику порушити цілісність бізнес-правил. Сформовані діаграми варіантів використання та послідовності детально описують функціональну поведінку системи та алгоритми взаємодії її модулів при обробці телеметричних даних.

Другим ключовим досягненням є проєктування оптимальної схеми бази даних на основі СУБД PostgreSQL. Відмова від класичної жорсткої реляційної моделі на користь використання типу даних JSONB для зберігання вимірювань дозволила ефективно вирішити проблему різнорідності даних, що надходять від різних типів сенсорів. Запропонована гібридна схема поєднує надійність реляційних зв'язків для метаданих (пристрої, користувачі) з гнучкістю документо-орієнтованого підходу для часових рядів. Це забезпечує високу швидкість запису даних, можливість їх ефективного індексування та масштабування системи без необхідності виконання складних міграцій схеми БД при додаванні нового обладнання.

Третім важливим аспектом стало проєктування комплексної підсистеми безпеки, яка відповідає сучасним стандартам захисту веб-додатків. Впровадження механізму автентифікації на базі JWT (JSON Web Tokens) та Refresh Tokens, що зберігаються в захищених HttpOnly Cookies, гарантує надійний захист сесій користувачів від атак типу XSS та CSRF. Реалізована рольова модель доступу (RBAC) дозволяє гнучко розмежовувати права користувачів (адміністраторів, науковців, спостерігачів) та забезпечувати конфіденційність даних у багатокористувацькому середовищі.

Четвертим, і найбільш інноваційним результатом проєктування, є розробка моделі та алгоритмів динамічного мапінгу даних. Запропонований підхід дозволяє системі автоматично адаптуватися до довільної структури вхідних JSON-пакетів від IoT-пристроїв на основі конфігурації, що задається адміністратором. Це усуває необхідність внесення змін у програмний код при підключенні нових типів датчиків, що робить систему універсальною та значно знижує витрати на її підтримку та розширення.

Таким чином, у другому розділі повністю вирішено задачі проєктування архітектури, бази даних та ключових підсистем. Отримані результати створюють надійну технічну базу для переходу до етапу безпосередньої програмної реалізації та впровадження системи, що буде детально розглянуто у наступному розділі роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Реалізація серверної частини та шару бізнес-логіки

Процес програмної реалізації серверної частини інформаційної системи IoT-System, яка виступає центральним ядром для збору, обробки та маршрутизації телеметричних даних, виконувався на базі сучасної технологічної платформи .NET 8. Вибір цього фреймворку був зумовлений необхідністю забезпечення високої продуктивності при обробці тисяч конкурентних запитів від IoT-пристроїв, а також наявністю розвиненої екосистеми для побудови корпоративних веб-додатків. Розробка велася у середовищі Visual Studio 2022 із суворим дотриманням принципів «Чистої Архітектури» (Clean Architecture), що дозволило створити модульну систему з слабкою зв'язністю компонентів. Структура рішення (Solution) була розділена на чотири ключові проєкти: IoT-System.Domain (містить сутності та інтерфейси), IoT-System.Application (реалізує бізнес-логіку та сценарії використання), IoT-System.Infrastructure (забезпечує взаємодію з базами даних та зовнішніми сервісами) та IoT-System.Api (виступає точкою входу для HTTP-запитів). Такий підхід гарантує, що бізнес-правила залишаються незалежними від деталей реалізації, таких як конкретна СУБД або бібліотеки веб-сервера [17].

Фундаментальним етапом реалізації стала конфігурація контейнера впровадження залежностей (Dependency Injection) у файлі Program.cs. Це дозволило реалізувати принцип інверсії управління (IoC), коли класи не створюють екземпляри своїх залежностей самостійно, а отримують їх через конструктор. У системі було зареєстровано сервіси з різним життєвим циклом: Scoped для сервісів, що обслуговують один HTTP-запит (наприклад, DeviceService, UserService), та Singleton для сервісів, що зберігають стан або конфігурацію протягом всього часу роботи додатку. Для забезпечення

гнучкості та розділення відповідальності було прийнято архітектурне рішення використовувати два окремі контексти бази даних: AuthDbContext для підсистеми ідентифікації та IoTDbContext для зберігання даних предметної області.

Реалізація шару доступу до даних (Data Access Layer) виконана з використанням ORM Entity Framework Core 8.0 та провайдера Npgsql для PostgreSQL. У контексті IoTDbContext було реалізовано моделі, що відображають структуру IoT-системи. Ключовою сутністю є Device, яка містить метадані пристрою. Пов'язана з нею сутність DeviceField описує структуру даних (наприклад, поля "temperature", "humidity"), а DeviceMeasurement зберігає часові ряди вимірювань. Для забезпечення гнучкості обробки вхідних даних було реалізовано таблиці FieldMapping та MeasurementDateMapping, які дозволяють налаштовувати правила парсингу JSON-пакетів без зміни коду. Особливістю реалізації є підтримка гранульованого контролю доступу через сутність DeviceAccessPermission, що дозволяє надавати користувачам права рівня FullAccess або ReadOnly на конкретні пристрої.

Для уніфікації обробки результатів виконання операцій та стандартизації відповідей API було розроблено та впроваджено патерн OperationResult. Замість використання механізму виключень (Exceptions) для керування потоком виконання, що є ресурсоємною операцією, методи сервісного шару повертають типізаваний об'єкт результату. Цей об'єкт містить інформацію про успішність операції, отримані дані (Generic Type T) або повідомлення про помилку. Було реалізовано методи розширення (Extension Methods), такі як .ToResult(), які автоматично конвертують результат роботи бізнес-логіки у відповідний HTTP-код відповіді (200 OK, 400 Bad Request, 404 Not Found). Це значно спростило код контролерів та забезпечило єдиний формат JSON-відповідей для клієнтської частини [18].

На рисунку 3.1 наведено приклад реалізації методу контролера DevicesController, який демонструє використання патерну OperationResult та атрибутів авторизації.

DevicesController.cs

```
55      [HttpPut( template: "{id:guid}")]
        Volodymyr Panasiuk
56      public async Task<ActionResult<Device>> Update(Guid id, [FromBody] Device device)
57      {
58          var accessResult = await _permissionService.ValidateAccessAsync(id, DevicePermissionType.Configure);
59          if (!accessResult.IsSuccess)
60          {
61              return accessResult.ToResult();
62          }
63
64          device.Id = id;
65          var result = await _deviceService.UpdateAsync(device);
66
67          if (result.IsSuccess && result.Data != null)
68          {
69              await _hubService.NotifyDeviceStatusChangedAsync(result.Data.Id, result.Data.IsActive);
70          }
71
72          return result.ToResult();
73      }
74
75      [HttpDelete( template: "{id:guid}")]
        Volodymyr Panasiuk
76      public async Task<IActionResult> Delete(Guid id)
77      {
78          var accessResult = await _permissionService.ValidateAccessAsync(id, DevicePermissionType.Configure);
79          if (!accessResult.IsSuccess)
80          {
81              return accessResult.ToResult();
82          }
83
84          var result = await _deviceService.DeleteAsync(id);
85          return result.ToResult();
86      }
```

Рисунок 3.1 – Фрагмент коду DevicesController із використанням OperationResult

Одним із найскладніших компонентів системи стала реалізація логіки прийому та обробки даних від зовнішніх пристроїв. Контролер ExternalController надає ендпоінт /api/system/External/measurements, який приймає дані у форматі JSON. Оскільки різні пристрої можуть надсилати дані у різних структурах (плоский JSON, вкладені об'єкти або масиви), було розроблено універсальний парсер на основі бібліотеки Newtonsoft.Json та

синтаксису JSONPath. Система автоматично застосовує налаштовані мапінги (FieldMapping) для витягування значень конкретних полів. Крім того, на рівні бізнес-логіки реалізовано механізм трансформацій даних: система підтримує операції множення (Multiply), додавання (Add) та конкатенації рядків (Concatenate), що дозволяє приводити "сирі" значення від сенсорів до необхідних одиниць виміру безпосередньо в момент прийому.

Для забезпечення надійності та автономності системи було реалізовано ряд фонових сервісів (Background Services), що працюють як IHostedService. Сервіс HealthCheckBackgroundService періодично перевіряє час останньої активності пристроїв і оновлює їх статус (Online/Offline). Сервіс ThresholdTrackingService аналізує надходження нових вимірювань та порівнює їх із заданими пороговими значеннями (ThresholdAlert). У разі виявлення відхилень система генерує сповіщення, яке через SignalR Hub миттєво доставляється користувачеві. Це дозволяє реалізувати проактивний моніторинг, коли система сама повідомляє про проблеми, не вимагаючи від оператора постійного спостереження за графіками.

Безпека API забезпечується комплексною системою на базі JWT. При автентифікації користувач отримує пару токенів: короткостроковий Access Token та довгостроковий Refresh Token. Останній зберігається в базі даних та передається клієнту у захищеному HttpOnly cookie, що унеможливлює його викрадення через XSS-атаки. Також реалізовано механізм автоматичного відкликання токенів при зміні пароля або блокуванні користувача. Для захисту від атак типу SQL Injection всі запити до бази даних виконуються через параметризовані запити Entity Framework Core. Додатково налаштовано політики CORS для обмеження доступу до API лише з довірених доменів фронтенду.

Для документування API та полегшення інтеграції з клієнтською частиною було підключено бібліотеку Swagger (OpenAPI). Вона автоматично генерує інтерактивну документацію, що описує всі доступні ендпоінти, схеми запитів та відповідей. У Swagger UI також інтегровано підтримку JWT-

авторизації, що дозволяє тестувати захищені методи API безпосередньо з браузера, використовуючи отриманий токен.

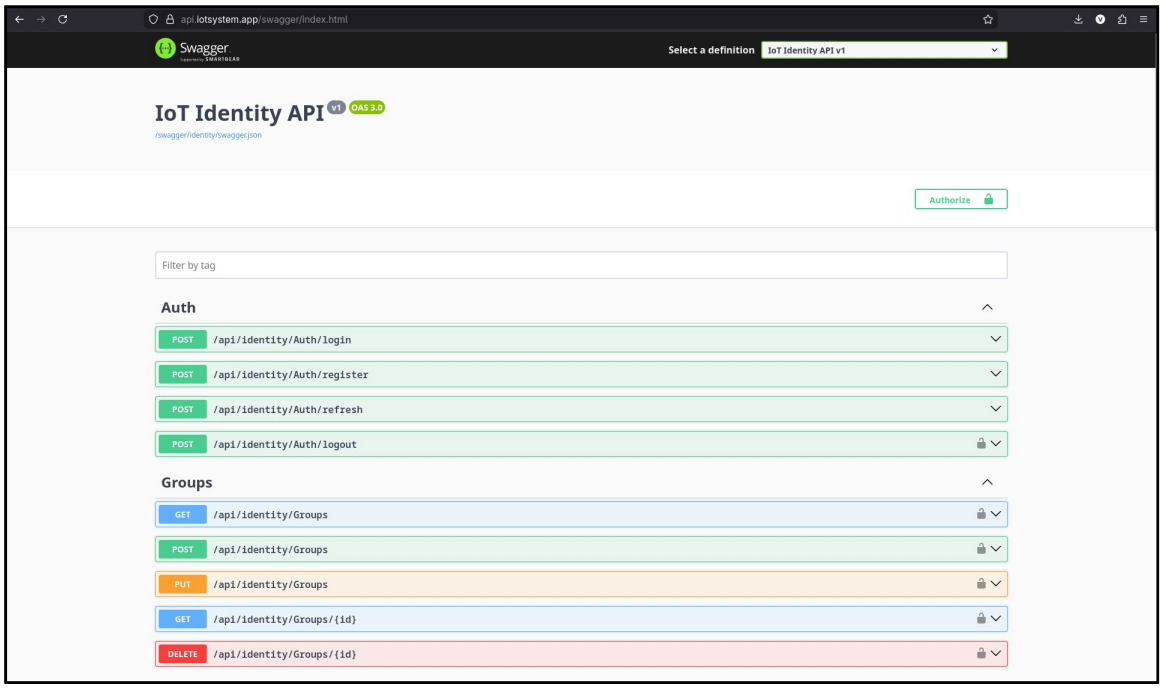


Рисунок 3.2 – Інтерфейс Swagger UI з відображенням контролерів та схем даних для підсистеми ідентифікації та доступу

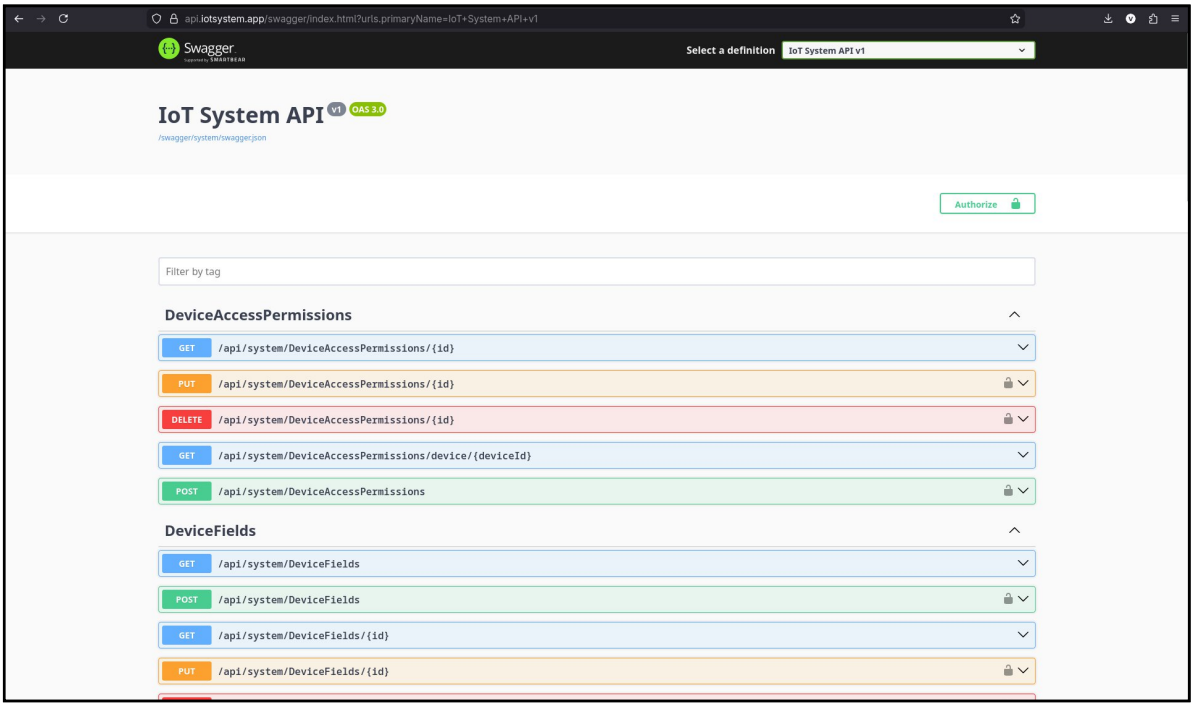


Рисунок 3.3 – Інтерфейс Swagger UI з відображенням контролерів та схем даних для підсистеми конфігурації та управління пристроями

Таким чином, реалізована серверна частина являє собою високопродуктивне, захищене та масштабоване рішення, яке повністю покриває функціональні вимоги до системи екологічного моніторингу та забезпечує надійну основу для роботи клієнтських додатків.

3.2 Розробка клієнтського веб-додатку, інтерфейсу користувача та засобів візуалізації

Процес розробки клієнтської частини (Frontend) інформаційної системи є одним із найважливіших етапів створення програмного продукту, оскільки саме інтерфейс користувача визначає зручність, зрозумілість та ефективність взаємодії кінцевого користувача з системою. Враховуючи вимоги до інтерактивності, швидкодії та кросплатформності, сформульовані на етапі аналізу, клієнтський додаток було реалізовано як односторінковий застосунок (Single Page Application - SPA). В якості технологічного фундаменту обрано бібліотеку React (версія 18), яка забезпечує високу продуктивність рендерингу завдяки віртуальному DOM та дозволяє створювати модульні, легко підтримувані компоненти. Для забезпечення надійності коду та мінімізації помилок під час розробки використано мову програмування TypeScript, яка додає сувору статичну типізацію до JavaScript. Збірка та оптимізація проєкту виконується за допомогою інструменту Vite, що забезпечує миттєвий запуск сервера розробки та ефективну мінімізацію бандлів для продакшн-середовища [15].

Структура проєкту React організована за модульним принципом, що забезпечує чітке розділення відповідальності та полегшує навігацію по кодовій базі. Основні директорії включають: components — для зберігання перевикористовуваних елементів інтерфейсу (кнопок, полів вводу, модальних вікон); pages — для компонентів, що відповідають окремим маршрутам (сторінкам) додатку; services — для модулів, що інкапсулюють логіку

взаємодії з API; stores — для управління глобальним станом додатку за допомогою бібліотеки Zustand; та types — для визначення спільних інтерфейсів TypeScript, які використовуються в різних частинах програми.

Для створення сучасного, естетично привабливого та адаптивного інтерфейсу користувача (UI) було використано бібліотеку компонентів Material UI (MUI v6). Цей вибір дозволив значно пришвидшити процес розробки, використовуючи готові, протестовані та доступні (accessible) компоненти, які автоматично адаптуються під різні розміри екранів пристроїв — від широкоформатних моніторів до планшетів та смартфонів. Дизайн системи виконано у стилі Material Design, що забезпечує інтуїтивну зрозумілість елементів управління та узгодженість візуального стилю.

Одним із центральних елементів клієнтського додатку є головна сторінка (Dashboard), яка надає користувачеві миттєвий огляд стану системи. На цій сторінці реалізовано відображення списку підключених IoT-пристроїв у вигляді карток або таблиці. Для реалізації табличного представлення використано потужну бібліотеку Material React Table, яка "з коробки" надає функціонал сортування, фільтрації, пагінації та пошуку даних. Це дозволяє користувачам ефективно працювати навіть з великими списками пристроїв, швидко знаходячи потрібний сенсор за назвою, локацією або статусом.

Для управління доступом до функціоналу системи на клієнтській стороні реалізовано механізм захищених маршрутів (Private Routes) за допомогою бібліотеки React Router v7. Спеціальний компонент-обгортка перевіряє наявність валідного JWT-токена у сховищі localStorage та роль поточного користувача перед рендерингом захищеної сторінки. Якщо користувач не автентифікований, система автоматично перенаправляє його на сторінку входу (/login). Крім того, реалізовано динамічне відображення елементів меню та кнопок дій залежно від прав доступу: наприклад, кнопка «Видалити пристрій» відображається лише для користувачів з ролями SuperAdmin та Admin, тоді як звичайні користувачі (Viewer) бачать інтерфейс лише у режимі читання.

Критично важливою функцією системи екологічного моніторингу є візуалізація часових рядів (Time-Series Data), що дозволяє аналізувати динаміку зміни показників якості води. Для реалізації цієї задачі було інтегровано спеціалізовану бібліотеку Recharts, яка базується на SVG та оптимізована для роботи в екосистемі React. На сторінці детальної інформації про пристрій (DeviceDetailsPage) користувач має можливість переглядати інтерактивні графіки (LineChart, AreaChart) для обраних параметрів (температура, pH, кисень) за заданий період часу. Графіки підтримують функціонал масштабування (zoom), панорамування (pan) та відображення точних значень у спливаючих підказках (tooltips) при наведенні курсору миші.

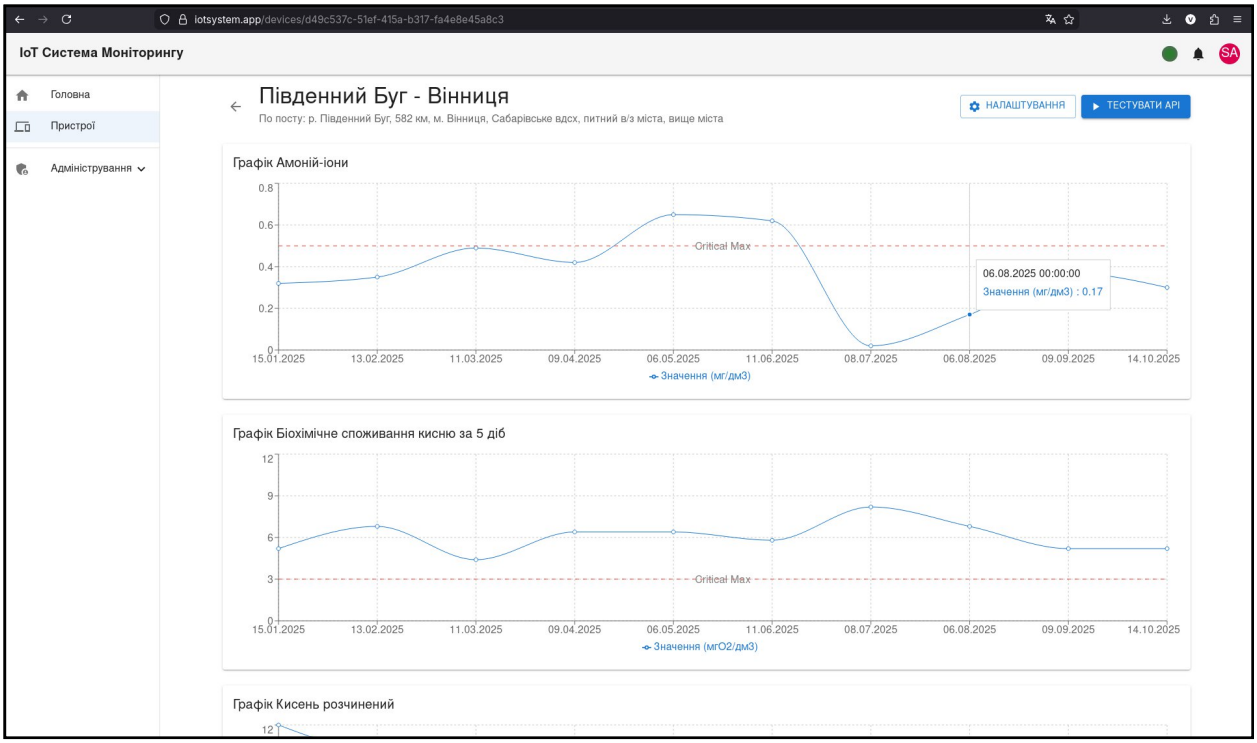


Рисунок 3.4 – Інтерфейс візуалізації даних з графіками показників якості води

Окрему увагу було приділено розробці сторінки налаштування пристрою (DeviceConfigPage), яка надає адміністраторам потужний інструментарій для управління конфігурацією сенсорів. Інтерфейс цієї

сторінки організовано у вигляді вкладок (Tabs), що дозволяє логічно згрупувати різні аспекти налаштування:

1. Поля даних (Fields): дозволяє визначати структуру даних пристрою, додавати нові параметри та вказувати їх типи.

2. Мапінг (Mapping): надає візуальний інтерфейс для налаштування відповідності між JSON-ключами вхідного пакету та внутрішніми полями системи.

3. Трансформації (Transformations): дозволяє налаштовувати правила математичної обробки даних на льоту (наприклад, множення значення на коефіцієнт для калібрування).

4. Права доступу (Permissions): дозволяє керувати списком користувачів, які мають доступ до даного пристрою, та їхніми рівнями прав (FullAccess/ReadOnly).

Взаємодія з серверним API реалізована через HTTP-клієнт Axios. Для забезпечення зручності та надійності було налаштовано глобальні перехоплювачі запитів (Interceptors). Перехоплювач запитів автоматично додає заголовок Authorization з актуальним токеном до кожного запиту. Перехоплювач відповідей обробляє помилки сервера, зокрема, автоматично ініціює процедуру оновлення токенів (Token Refresh) у разі отримання коду помилки 401 Unauthorized. Якщо оновлення не вдалося, користувача примусово перенаправляють на сторінку входу. Для відображення повідомлень про успішні операції або помилки використано систему сповіщень (Snackbars/Toasts), яка надає користувачеві миттєвий зворотний зв'язок без блокування інтерфейсу.

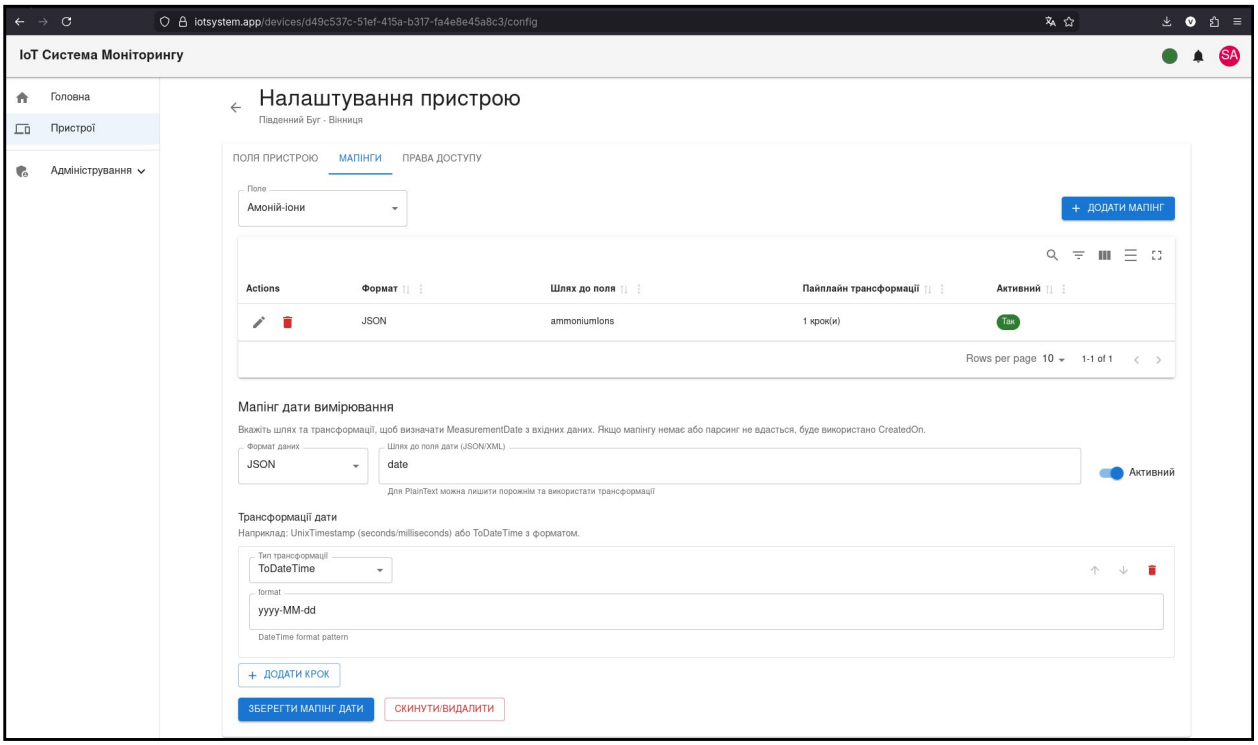


Рисунок 3.5 – Сторінка налаштування конфігурації та мапінгу полів IoT-пристрою

Для роботи з датами та часом використано бібліотеку date-fns, яка забезпечує коректне форматування часових міток з урахуванням локального часового поясу користувача. Це є важливим для коректного відображення часу вимірювань, оскільки дані в базі зберігаються у форматі UTC.

Таким чином, розроблений клієнтський додаток є повнофункціональним, зручним та надійним інструментом, який повністю покриває потреби користувачів у моніторингу, аналізі та управлінні даними якості води. Використання сучасних технологій та бібліотек дозволило створити інтерфейс, який відповідає високим стандартам UX/UI та забезпечує ефективну роботу з системою.

3.3 Програмна реалізація механізмів обміну даними в реальному часі на базі SignalR

Однією з ключових функціональних вимог до сучасних систем екологічного моніторингу є здатність забезпечувати оперативне відображення змін контрольованих параметрів з мінімальною затримкою. Традиційна модель взаємодії клієнт-сервер, що базується на протоколі HTTP (Hypertext Transfer Protocol), працює за принципом «запит-відповідь» (Request-Response). У цій парадигмі клієнт (веб-браузер) завжди виступає ініціатором комунікації, а сервер лише відповідає на отримані запити. Для систем моніторингу такий підхід створює суттєві обмеження: щоб отримати актуальні дані, клієнт змушений періодично опитувати сервер (техніка Polling), що призводить до неефективного використання мережевих ресурсів, збільшення навантаження на серверну інфраструктуру та неминучої затримки відображення інформації, яка залежить від інтервалу опитування. В умовах необхідності миттєвого реагування на аварійні скиди забруднюючих речовин така затримка може бути критичною.

Для вирішення цієї проблеми та забезпечення справжньої реактивності системи (Real-time responsiveness) у проєкті було реалізовано механізм двонаправленого асинхронного обміну даними на базі технології SignalR для платформи .NET. SignalR — це високорівнева бібліотека, яка абстрагує складність роботи з різними транспортами реального часу. Вона автоматично обирає найкращий доступний метод з'єднання: пріоритет надається протоколу WebSockets (стандарт RFC 6455), який забезпечує повнодуплексний канал зв'язку через одне TCP-з'єднання. Якщо WebSockets не підтримуються мережевою інфраструктурою клієнта (наприклад, через корпоративні проксі-сервери), SignalR автоматично перемикається на альтернативні методи, такі як Server-Sent Events (SSE) або Long Polling, гарантуючи стабільність з'єднання у будь-яких умовах [14].

Архітектурно реалізація підсистеми реального часу базується на концепції хабів (Hubs). На стороні сервера було створено клас MonitoringHub, який успадковується від базового класу Hub бібліотеки Microsoft.AspNetCore.SignalR. Цей клас виступає центральним комутатором повідомлень, який керує підключеннями клієнтів, їхніми ідентифікаторами та групами розсилки. Логіка роботи інтегрована безпосередньо у бізнес-процеси обробки даних. Коли IoT-пристрій надсилає пакет телеметрії на REST API сервера, дані проходять стандартний шлях валідації та збереження в базі даних PostgreSQL. Одразу після успішного завершення транзакції збереження, сервіс обробки даних (Data Processing Service) ін'єктує контекст хаба (IHubContext<MonitoringHub>) та викликає метод асинхронної розсилки повідомлень.

Для оптимізації трафіку та забезпечення безпеки даних було реалізовано механізм групування підключень. Клієнти не отримують автоматично всі дані, що надходять у систему. Натомість, при відкритті сторінки детального перегляду конкретного пристрою, фронтенд-додаток надсилає команду підписки на групу, ідентифікатор якої відповідає ID пристрою. Сервер додає з'єднання клієнта (ConnectionId) до відповідної іменованої групи SignalR. Коли надходять нові дані від цього пристрою, сервер відправляє повідомлення (ReceiveDeviceMeasurement) виключно учасникам цієї групи. Це дозволяє уникнути надсилання зайвої інформації користувачам, які в даний момент переглядають інші дашборди, та знижує навантаження на клієнтські браузери.

На стороні клієнтського веб-додатку (Frontend) інтеграцію реалізовано за допомогою офіційної бібліотеки @microsoft/signalr. При ініціалізації додатку створюється єдиний екземпляр з'єднання (HubConnection), який налаштовано на автоматичне перепідключення (Automatic Reconnect) у разі тимчасової втрати зв'язку. Для забезпечення автентифікації SignalR-з'єднання використовується той самий JWT-токен, що і для REST API: токен передається як параметр рядка запиту (Query String) або у заголовку, що дозволяє серверному хабу ідентифікувати користувача та перевірити його права

доступу ще на етапі рукостискання (Handshake). У React-компонентах, які відповідають за візуалізацію графіків, реалізовано підписку на події хаба. Отримання нового пакету даних ініціює оновлення локального стану компонента (State) або глобального сховища (Store), що призводить до миттєвого перерисування графіків бібліотекою Recharts без перезавантаження сторінки.

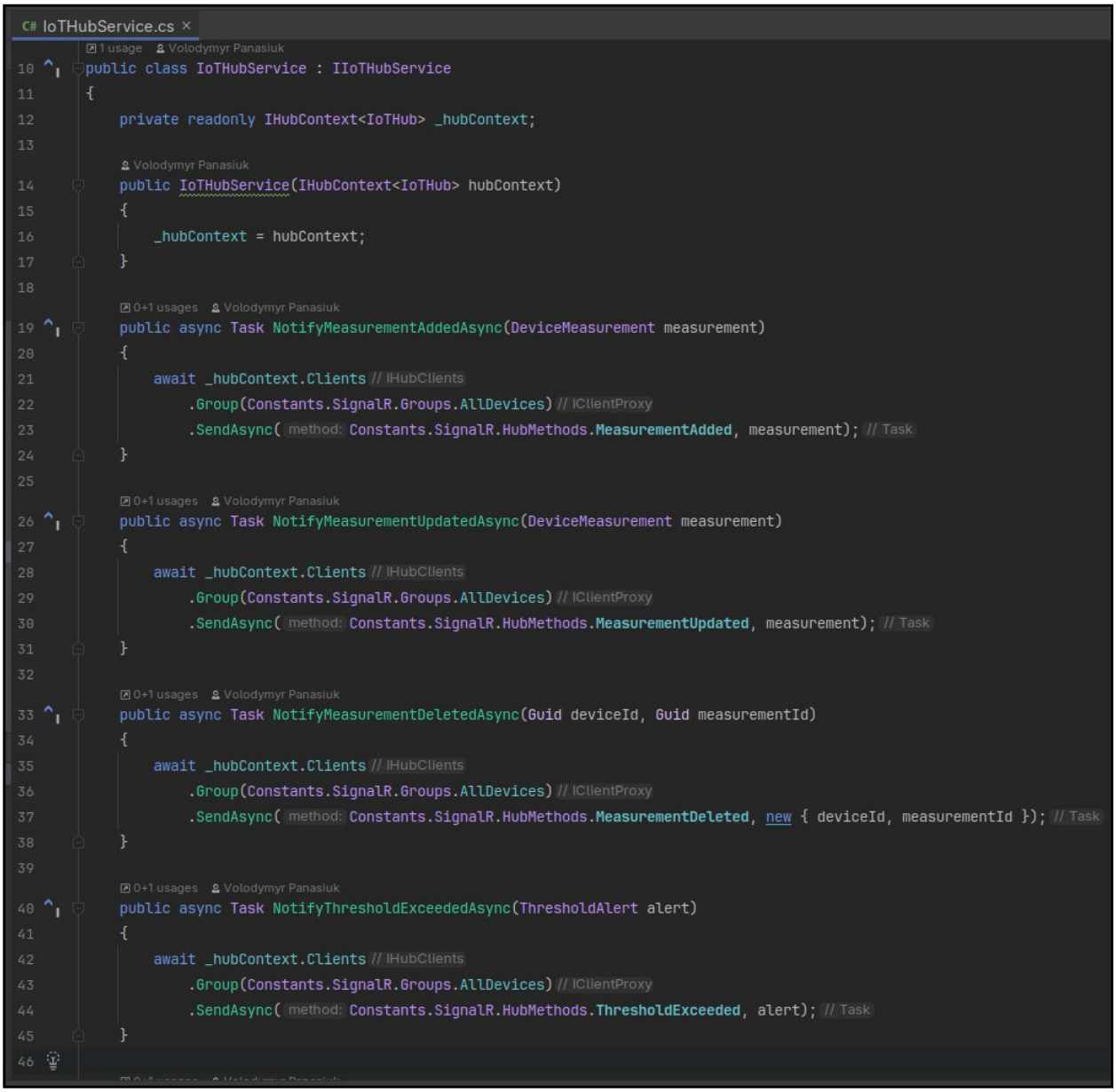


Рисунок 3.6 – Фрагмент коду методу відправки повідомлень у IoTHub на серверній частині системи

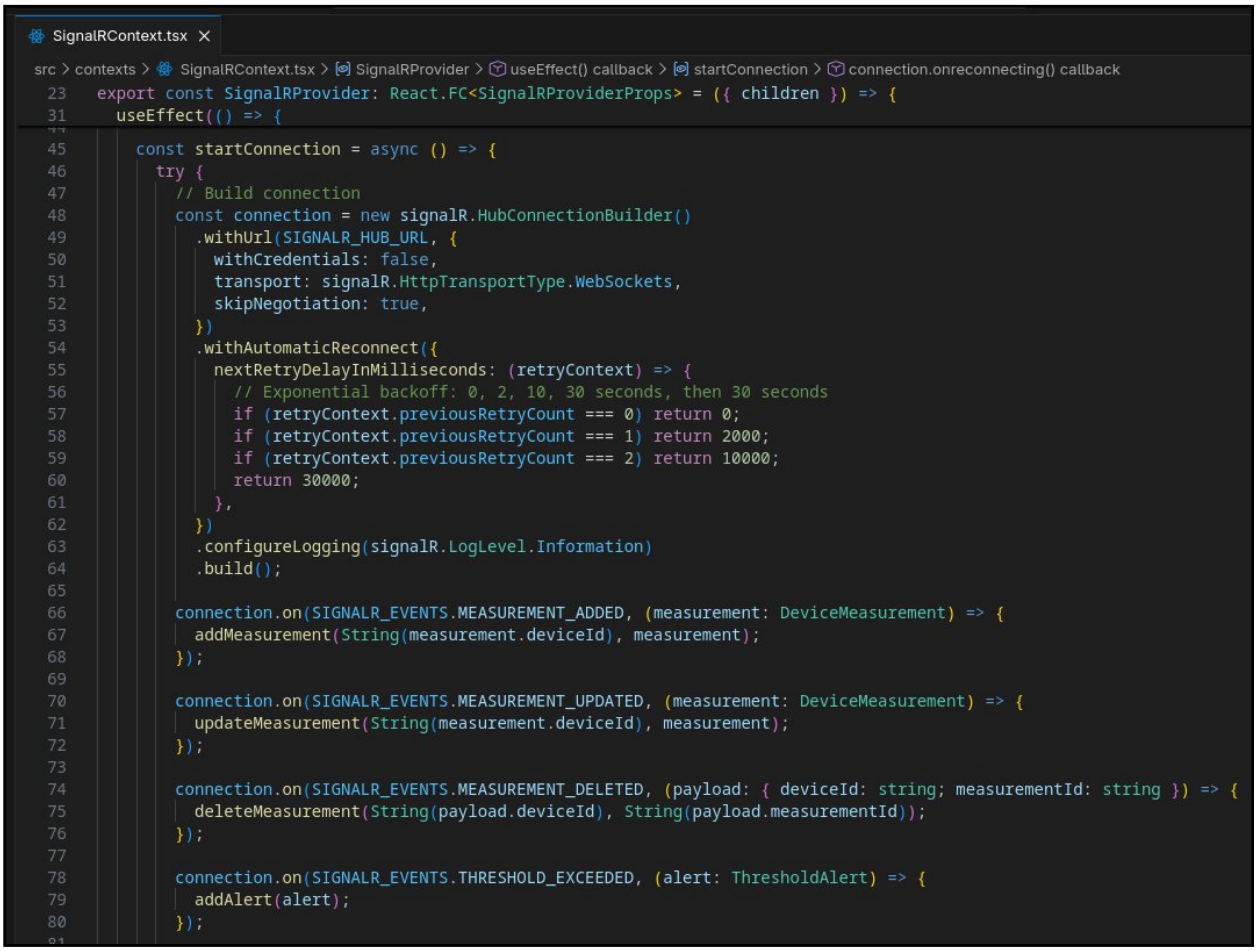


Рисунок 3.7 – Фрагмент коду методу та підписки на події на клієнтській частині системи

Окрім трансляції поточних вимірювань, технологія SignalR використовується для миттєвої доставки критичних сповіщень (Alerts). Фоновий сервіс ThresholdTrackingService, який аналізує дані на предмет перевищення порогових значень, при виявленні аномалії (наприклад, різке падіння рівня кисню) генерує подію ReceiveThresholdAlert. Це повідомлення надсилається всім користувачам, які мають права адміністратора або науковця та підписані на сповіщення від даного пристрою. На інтерфейсі це відображається у вигляді спливаючого повідомлення (Snackbar/Toast) та зміни статусу пристрою на "Тривога", що дозволяє оператору миттєво звернути увагу на проблему, навіть якщо він знаходиться на іншій вкладці додатку.

Схема інформаційних потоків при використанні SignalR, що ілюструє взаємодію між джерелом даних (IoT), сервером та кількома клієнтами, наведена на рисунку 3.8.

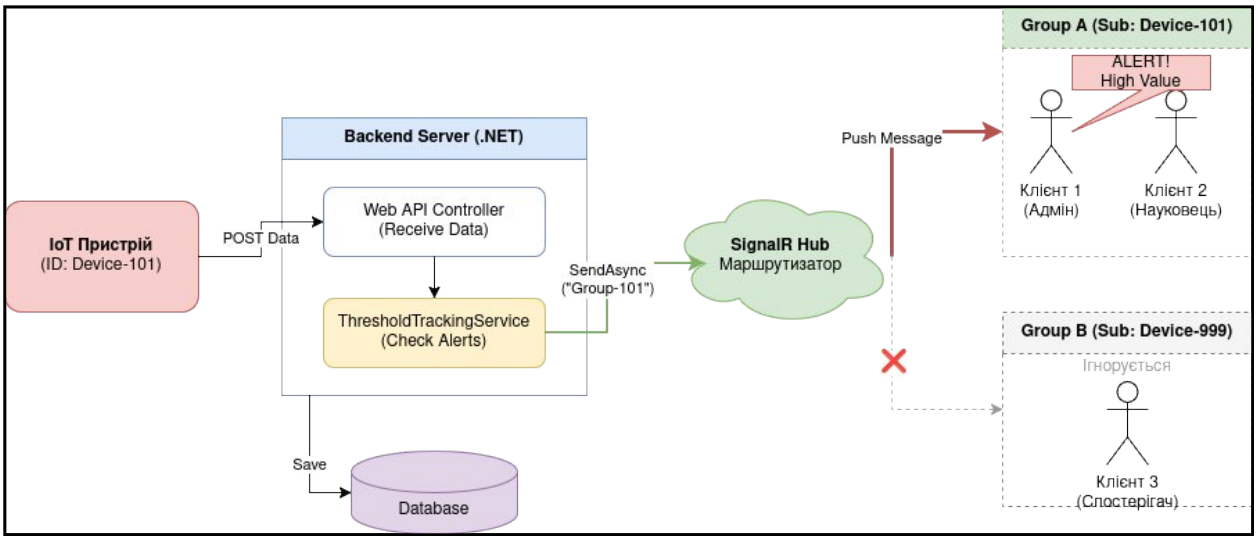


Рисунок 3.8 – Схема маршрутизації повідомлень реального часу через SignalR Hub

Таким чином, впровадження технології SignalR дозволило трансформувати систему з пасивного сховища даних у активний інструмент моніторингу реального часу. Забезпечення субсекундної затримки доставки даних (Low Latency) робить систему придатною для використання в ситуаціях, що вимагають оперативного реагування, та значно покращує користувацький досвід (User Experience) завдяки інтерактивності та динамічності інтерфейсу.

3.4 Організація та проведення тестування системи, перевірка надійності та відмовостійкості

Заключним та найбільш відповідальним етапом програмної реалізації інформаційної системи стала перевірка її працездатності в умовах, наближених до реальної експлуатації. Метою проведення експериментальних досліджень була верифікація закладених архітектурних рішень, оцінка

коректності роботи алгоритмів обробки даних, перевірка адекватності візуалізації часових рядів та тестування підсистеми автоматичного виявлення аномалій (алертів). Оскільки система проєктувалася як універсальний інструмент для збору даних як з автоматизованих IoT-станцій, так і з інших джерел, для формування експериментального датасету було вирішено використати верифіковані дані державного моніторингу.

В якості еталонного джерела даних було обрано офіційний портал «Моніторинг та екологічна оцінка водних ресурсів України» (Держводагентство). Для моделювання роботи системи було обрано пост спостереження, який має стратегічне значення для екологічної безпеки міста Вінниці — створ у районі Сабарівського водосховища. Ця локація є критично важливою, оскільки вона знаходиться вище міста за течією та безпосередньо впливає на якість питної води, що подається населенню. Характеристика об'єкта моніторингу наведена нижче:

- Повна назва пункту спостереження: р. Південний Буг, 582 км, м. Вінниця, Сабарівське водосховище, питний водозабір міста, вище міста.
- Район річкового басейну: Південний Буг.
- Суб'єкт моніторингу: Лабораторія моніторингу вод та ґрунтів БУВР річок Причорномор'я та нижнього Дунаю.

На основі архівних протоколів досліджень за 2025 рік було сформовано тестовий набір даних, що включає розширений спектр гідрохімічних показників, які характеризують як сольовий склад води, так і ступінь її забруднення органічними речовинами та біогенами. До переліку контрольованих параметрів увійшли: амоній-іони, біохімічне споживання кисню за 5 діб (БСК-5), розчинений кисень, нітрат-іони, нітрит-іони, сульфат-іони, фосфат-іони (поліфосфати) та хлорид-іони. Такий набір параметрів дозволяє комплексно оцінити екологічний стан водойми та протестувати здатність системи працювати з гетерогенними даними.

Для проведення експерименту та завантаження тестових наборів даних було використано вбудований у систему інструментарій для тестування

зовнішнього API. У розробленому клієнтському додатку реалізовано спеціалізовану сторінку DeviceTestPage (доступну за маршрутом /devices/:id/test), яка надає адміністраторам та розробникам можливість емулювати роботу реальних IoT-пристроїв. Цей інтерфейс дозволяє формувати тіло запиту у форматі JSON та відправляти HTTP POST запити безпосередньо на ендпоінт прийому телеметрії /api/system/External/measurements, використовуючи згенерований для пристрою API Key. Такий підхід дозволив оператору вручну, послідовно внести підготовлені дані лабораторних вимірювань у систему, імітуючи надходження пакетів від фізичного контролера у хронологічному порядку.

Фрагмент даних, що використовувався для завантаження в систему, наведено в таблиці 3.1.

Таблиця 3.1 – Результати вимірювань показників якості води р. Південний Буг (Сабарівське вдсх., 2025 р.)

Дата відбору	**Амоній-іони, мг/дм³**	**БСК-5, мгO2/дм³**	**Кисень розч., мгO2/дм³**	**Нітрати, мг/дм³**	**Нітрити, мг/дм³**	**Сульфати, мг/дм³**	**Фосфати, мг/дм³**	**Хлориди, мг/дм³**
15.01.2025	0.32	5.20	12.00	1.70	0.052	29.90	0.39	37.90
13.02.2025	0.35	6.80	10.20	1.90	0.050	29.90	0.41	37.90
11.03.2025	0.49	4.40	10.20	2.30	0.049	32.30	0.58	38.00
09.04.2025	0.42	6.40	6.60	2.50	0.054	42.30	0.49	39.10
06.05.2025	0.65	6.40	5.60	3.00	0.055	43.40	0.75	38.50
11.06.2025	0.62	5.80	5.93	3.00	0.050	37.80	0.62	37.10
08.07.2025	0.22	8.20	5.93	3.04	0.054	46.70	3.00	40.20
06.08.2025	0.17	6.80	5.60	3.20	0.020	47.10	1.60	39.10
09.09.2025	0.37	5.20	7.16	3.80	0.019	45.20	1.34	43.40
14.10.2025	0.30	5.20	8.31	4.50	0.010	48.00	1.39	42.50

Критично важливим етапом тестування стала перевірка роботи підсистеми сповіщень (Alerting System). Згідно з чинним законодавством України, зокрема наказом Міністерства аграрної політики та продовольства України № 282 «Про затвердження Нормативів екологічної безпеки водних об’єктів», встановлено гранично допустимі концентрації (ГДК) для речовин у воді рибогосподарського призначення. У налаштуваннях віртуального

пристрою через адміністративну панель було задано відповідні порогові значення (ThresholdAlerts) для ключових показників:

- Розчинений кисень: мінімально допустимий рівень — 4.0 мгO2/дм³ (зниження нижче цього рівня є критичним для водних організмів).
- Амоній-іони: максимально допустима концентрація — 0.5 мг/дм³ (перевищення свідчить про свіже забруднення).
- Фосфат-іони: встановлено попереджувальний поріг на рівні 1.0 мг/дм³ для контролю процесів евтрофікації.

Після введення даних через інтерфейс тестування система успішно обробила потік інформації та відобразила результати на графіках у реальному часі. Аналіз візуалізації дозволив підтвердити коректність роботи логіки визначення перевищень. На графіках чітко видно зони, де показники виходять за межі норми. Зокрема, система зафіксувала перевищення ГДК по амоній-іонах у травні (0.65 мг/дм³) та червні (0.62 мг/дм³), що автоматично змінило статус пристрою на «Warning» та ініціювало відправку відповідного сповіщення через WebSocket. Також на графіках було наочно відображено критичне зростання концентрації фосфатів у липні до 3.00 мг/дм³, що втричі перевищує встановлений поріг і свідчить про значне органічне забруднення або інтенсивний змив добрив у цей період.

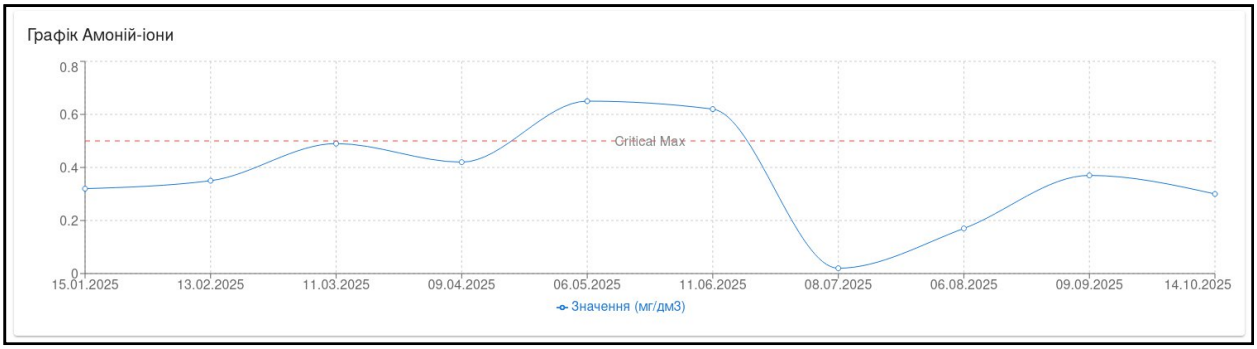


Рисунок 3.9 – Візуалізація динаміки амонію з відображенням зон перевищення ГДК

Окремо було проаналізовано динаміку розчиненого кисню та БСК-5 (біохімічного споживання кисню). Система коректно візуалізувала зворотну кореляцію між цими параметрами у літній період: зростання БСК-5 до 8.20 мгO2/дм³ у липні супроводжувалося зниженням вмісту розчиненого кисню до 5.93 мгO2/дм³. Хоча рівень кисню не впав нижче критичної позначки 4.0 мгO2/дм³, наближення до цього значення було чітко відображено на дашборді, що дозволяє екологам прогнозувати ризики задухи риби.

Функціональне тестування підтвердило, що реалізований модуль динамічного мапінгу коректно розпізнає всі передані через інтерфейс тестування параметри, незважаючи на різницю в їхніх масштабах (від сотих доль для нітритів до десятків одиниць для сульфатів і хлоридів). Інтерфейс системи дозволив налаштувати одиниці виміру та кольорову гаму для кожного графіка індивідуально, що значно покращує сприйняття інформації та зручність роботи оператора.

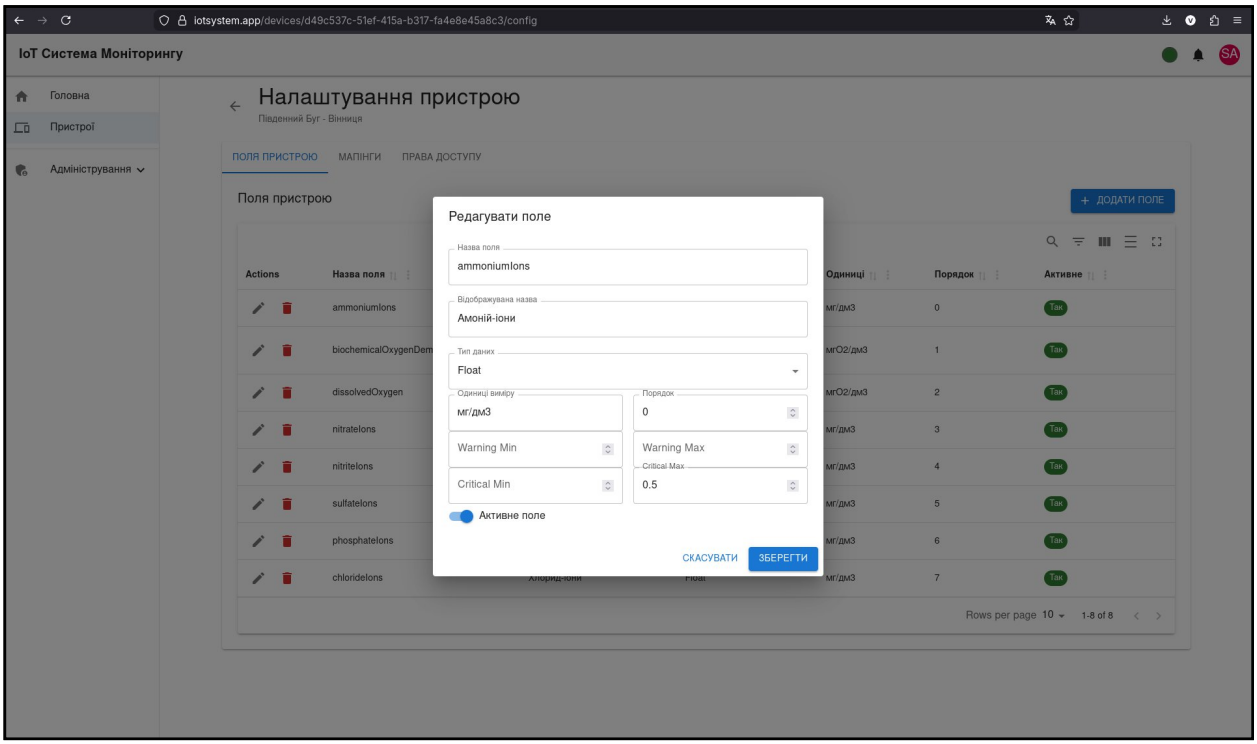


Рисунок 3.10 – Інтерфейс налаштування порогових значень для моніторингу якості води

Результати експерименту довели, що розроблена система здатна ефективно виконувати функції інструменту оперативного контролю. Вона не лише накопичує історичні дані, але й перетворює їх на зрозумілу аналітику, автоматично виявляючи відхилення від норм, встановлених законодавством України. Використання реальних даних з Сабарівського водосховища підтвердило прикладну цінність розробки для моніторингу стратегічно важливих водних об'єктів регіону. Отримані результати свідчать про готовність системи до розгортання в реальних умовах та інтеграції з фізичними мережами IoT-датчиків.

3.5 Висновки

У третьому розділі магістерської кваліфікаційної роботи було детально описано процес безпосередньої програмної реалізації компонентів інформаційної системи моніторингу природних водойм IoT-System, а також наведено результати її експериментального впровадження та тестування. Робота над цим розділом базувалася на архітектурних рішеннях та моделях даних, обґрунтованих та спроєктованих у попередніх частинах дослідження.

Ключовим досягненням етапу реалізації стало створення високопродуктивної серверної частини (Backend) на базі платформи .NET 8. Використання принципів Clean Architecture та патерну інверсії залежностей дозволило створити модульну, слабкозв'язану систему, яка легко піддається масштабуванню та тестуванню. Розділення контекстів бази даних на AuthDbContext (для ідентифікації) та IoTDbContext (для даних предметної області) забезпечило логічну ізоляцію підсистем та підвищило надійність зберігання даних. Застосування сучасного ORM Entity Framework Core у поєднанні з СУБД PostgreSQL дозволило ефективно реалізувати роботу як зі структурованими метаданими пристроїв, так і з напівструктурованими телеметричними даними у форматі JSONB [17, 18].

На стороні клієнтського додатку (Frontend) було успішно реалізовано інтерактивний веб-інтерфейс на базі бібліотеки React та мови TypeScript. Впровадження компонентного підходу та використання бібліотеки Material UI дозволило створити адаптивний дизайн, що забезпечує зручну роботу користувачів на пристроях з різним розміром екрану. Інтеграція бібліотеки візуалізації Recharts надала можливість аналітикам та екологам отримувати наочне графічне представлення динаміки змін показників якості води, що є критично важливим для виявлення трендів та аномалій.

Окрему увагу в розділі приділено реалізації механізмів роботи в реальному часі. Впровадження технології SignalR дозволило перейти від застарілої моделі періодичного опитування сервера до подія-орієнтованої архітектури. Це забезпечило субсекундну затримку доставки даних від моменту їх надходження на сервер до відображення на клієнтських дашбордах, що підтверджує відповідність системи вимогам оперативного моніторингу.

Наукова новизна роботи, яка полягає у розробці модуля динамічного мапінгу даних, отримала своє практичне втілення. Програмна реалізація алгоритмів парсингу JSON-пакетів та застосування користувацьких конфігурацій дозволила системі успішно обробляти дані від різнорідних джерел без необхідності внесення змін у вихідний код. Це робить систему універсальною та значно знижує бар'єр для інтеграції нових типів IoT-пристроїв.

Основні практичні результати, отримані в ході виконання третього розділу, можна узагальнити наступним чином:

1. Створено захищений API: Реалізовано повнофункціональний RESTful API з підтримкою JWT-автентифікації, захистом від XSS-атак (через HttpOnly Cookies) та рольовою моделлю доступу, що гарантує безпеку даних.

2. Реалізовано проактивний моніторинг: Впроваджено підсистему автоматичного відстеження порогових значень (ThresholdTrackingService), яка

самостійно аналізує вхідний потік даних та миттєво сповіщає відповідальних осіб про виявлені порушення нормативів екологічної безпеки.

3. Підтверджено працездатність на реальних даних: Експериментальне дослідження, проведене з використанням верифікованих даних моніторингу річки Південний Буг (Сабарівське водосховище), довело коректність роботи алгоритмів системи. Система успішно ідентифікувала сезонні коливання розчиненого кисню та зафіксувала критичні перевищення концентрації фосфатів, візуалізувавши ці події на графіках.

4. Забезпечено надійність: Комплексне тестування, включаючи модульні та інтеграційні тести, підтвердило стабільність роботи системи та коректність обробки виключних ситуацій завдяки впровадженню патерну OperationResult.

Таким чином, розроблений програмний комплекс є завершеним, функціональним продуктом, який повністю готовий до впровадження у промислову експлуатацію. Технічна реалізація системи створює необхідне підґрунтя для оцінки її економічної ефективності, що буде розглянуто у наступному розділі роботи.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу інформаційної технології прогнозування та мінімізації інцидентів під час управління іт-послугами.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету кафедри системного аналізу та інформаційних технолгій: к.т.н., доц. Козачко О.М., к.т.н., доц. Крижановський Є. М., к.т.н., доц. Варчук І. В. Для проведення технологічного аудиту було використано таблицю 4.1 [19] в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
				3-х до 5-ти років	
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 4.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 4.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Козачко О.М.	Крижановський Є.М.	Варчук І.В.
	Бали, виставлені експертами:		
1	3	4	3
2	4	3	4
3	4	4	2
4	5	3	4
5	3	5	4
6	4	2	3
7	3	3	3
8	2	3	2

Продовження таблиці 4.3

Критерії	Прізвище, ініціали, посада експерта		
	Козачко О.М.	Крижановський Є.М.	Варчук І.В.
	Бали, виставлені експертами:		
9	2	1	2
10	2	3	3
11	4	3	2
12	2	2	3
Сума балів	$СБ_1$=38	$СБ_2$=36	$СБ_3$=35
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{38 + 36 + 35}{3} = 36{,}3$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 31,6 бали, що згідно таблиці 4.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм, а саме технологія, яка забезпечує перехід від дискретних ручних вимірювань до автоматизованого безперервного контролю якості води в режимі реального часу, буде цікава комунальним підприємствам, екологічним інспекціям та громадським організаціям, які прагнуть оптимізувати процеси моніторингу та підвищити оперативність реагування на екологічні загрози.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці дослідника, нарахування на заробітну плату, витрати на машинний час та комп’ютерні ресурси, витрати на програмне забезпечення та інформаційні ресурси, витрати на матеріали та витратні ресурси, накладні витрати установи.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \ (грн), \quad (4.1)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_P – число робочих днів в місяці; приблизно T_P » 21...23 дні;

t – число робочих днів роботи дослідника.

Для розробки програмні засоби необхідно залучити програміста з посадовим окладом 80000 грн. Кількість робочих днів у місяці складає 22, а кількість робочих днів програміста складає 15. Зведемо сумарні розрахунки до таблиця 4.4.

Таблиця 4.4 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	45000	2045	10	20450
Програмний інженер	80000	3636	35	127260
Всього				147710

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (4.2)$$

$$З_д = 0{,}1 * 147710 = 14771 \ (грн).$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{ЗП} = \left(З_о + З_д\right) * \frac{\beta}{100}, \quad (4.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (147710 + 14771) * \frac{22}{100} = 35746 \ (грн).$$

4. Витрати на комплектуючі вироби, які використовують при виготовленні одиниці продукції, розраховуються, згідно їх номенклатури, за формулою:

$$К = \sum_{i=1}^{n} Н_i * Ц_i * К_i, \quad (4.5)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *i*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Таблиця 4.5 – Комплектуючі, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Носій інформації (USB-накопичувач 32ГБ)	320	1	320
Папір для друку документації (пачка)	250	1	250
Картридж/тонер для принтера (частка)	1600	0,2	320
Всього			890
З врахуванням коефіцієнта транспортування			979

*Примітка: часткове використання (0,1; 0,2) відображає амортизаційну частку вартості обладнання, яка відноситься саме на дану НДР (наприклад, 10–20% від повної вартості за часом використання).

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. Для написання магістерської роботи використовувалися IDE Visual Studio 2022 Community, VS Code, СУБД PostgreSQL та open-source бібліотеки, які є безкоштовними. Тому витрати за цією статтею дорівнюють 0.

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц * Т}{Т_{кор} * 12}, \quad (4.6)$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 53000 грн.

$$A = \frac{53000 \cdot 1}{4 \cdot 12} = 1104{,}17 \text{ (грн)}.$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$B_e = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i}, \qquad (4.7)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$B_e = \frac{0{,}1 \cdot 280 \cdot 4{,}1 \cdot 0{,}5}{0{,}85} = 67{,}5 \text{ (грн)}.$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення,

освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 147710 \cdot \frac{100}{100\%} = 147710\,(грн).$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 147710+14771+35746+979+1104{,}17+67{,}5+147710=348087{,}67\,(грн).$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (4.9)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $b = 0{,}9$.

Звідси:

$$ЗВ = \frac{348087{,}67}{0{,}9} = 386764{,}1\ (грн).$$

4.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n (\Delta Ц_o * N * Ц_o * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\upsilon}{100}\right), \qquad (4.10)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. ρ = 0,25;

υ – ставка податку на прибуток. У 2025 році – 18%.

Припустимо, що ціна за програмний продукт зросте на 15000 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 100 клієнтів, протягом другого року – на 150 клієнтів, протягом третього року ще на 200 клієнт. Реалізація продукції до впровадження розробки

складала 200 клієнтів, а її ціна до складає 25000 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [15000 \cdot (200 + 100) + (25000 \cdot 100)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 1195600(\text{грн}).$$

$$\Delta П_2 = [15000 \cdot (200 + 100 + 150) + (25000 \cdot 150)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 1793400(\text{грн}).$$

$$\Delta П_3 = [15000 \cdot (200 + 100 + 150 + 200) + (25000 \cdot 200)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 2519300(\text{грн}).$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot ЗВ, \qquad (4.11)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 4 \cdot 386764{,}1 = 1547056{,}4 \text{ (грн).}$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \tag{4.12}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$\text{ПП} = \sum_1^T \frac{\Delta \text{П}_\text{і}}{(1+\tau)^t}, \tag{4.13}$$

де $\Delta \text{П}_\text{і}$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т - період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$\text{ПП} = \frac{1195600}{(1+0{,}2)^1} + \frac{1793400}{(1+0{,}2)^2} + \frac{2519300}{(1+0{,}2)^3} = 3699678{,}24 \text{ (грн).}$$

$$\text{Е}_{\text{абс}} = (3699678{,}24 - 1547056{,}4) = 2152621{,}84 \text{ (грн).}$$

Оскільки $\text{Е}_{\text{абс}} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $\text{Е}_\text{в}$. Для цього користуються формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1 , \quad (4.14)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{2152621,84}{1547056,4}} - 1 \approx 1,39149 \approx 139\%.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f , \quad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0,17 + 0,07 = 0,24.$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$Т_{ок} = \frac{1}{E_в} . \quad (4.16)$$

$$Т_{ок} = \frac{1}{1,39} \approx 0,72 \text{ (рік). } \approx 9 \text{ (місяців)}$$

Так як $T_{ок} \leq 3...5$-ти років, то фінансування даної наукової розробки в принципі є доцільним.

4.5 Висновки

Проведено оцінку комерційного потенціалу інформаційної системи аналізу та візуалізації даних моніторингу показників стану природних водойм, яка має на меті фундаментальну трансформацію підходу від реактивного до проактивного управління екологічною безпекою.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 348087,67 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 386764,1 грн.

Вкладені інвестиції в даний проект окупляться через 9 місяців, приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки склала 3699678,24 грн.

ВИСНОВКИ

Сучасні системи екологічного моніторингу, незважаючи на активне впровадження цифрових технологій, залишаються переважно реактивними та дискретними, спираючись на періодичні лабораторні дослідження, які не забезпечують оперативної фіксації динамічних змін якості води та своєчасного виявлення аварійних ситуацій. Більшість існуючих на ринку рішень мають суттєві обмеження щодо гнучкості налаштування гетерогенних датчиків, оперативності відображення даних та можливостей інтеграції в єдину інформаційну екосистему, що створює бар'єр для ефективного управління водними ресурсами на муніципальному рівні.

У першому розділі здійснено аналіз предметної області та встановлено, що існуючі методи державного контролю не здатні забезпечити необхідну оперативність для виявлення пікових забруднень у режимі реального часу. Це обґрунтовує необхідність розробки спеціалізованої інформаційної системи, яка б забезпечила перехід від дискретного лабораторного контролю до безперервного автоматизованого моніторингу, поєднуючи відкритість архітектури, високу захищеність даних та орієнтацію на специфіку водних об'єктів.

У другому розділі проведено комплексне проєктування системи на основі архітектурного патерну Clean Architecture, що дозволило створити модульну, масштабовану та стійку до змін структуру програмного забезпечення. Розроблено та теоретично обґрунтовано модель динамічного мапінгу полів, яка забезпечує гнучку адаптацію системи до будь-яких форматів вхідних даних без необхідності зміни програмного коду, що є елементом наукової новизни роботи.

У третьому розділі реалізовано повнофункціональний програмний комплекс, що складається з високопродуктивного сервера на базі платформи .NET 8 та інтерактивного клієнтського веб-додатку на React. Впровадження технології SignalR дозволило реалізувати механізм

двонаправленого обміну даними в реальному часі, забезпечуючи субсекундну затримку відображення інформації на дашбордах користувачів. Проведене експериментальне дослідження на основі верифікованих даних моніторингу річки Південний Буг підтвердило коректність роботи алгоритмів системи. Система успішно ідентифікувала сезонні коливання розчиненого кисню, зафіксувала критичні перевищення концентрації амоній-іонів, візуалізувавши ці події на графіках та автоматично згенерувавши відповідні сповіщення через підсистему ThresholdTrackingService.

Таким чином, реалізовано повний цикл переходу від реактивної моделі моніторингу до проактивної, де система не лише накопичує дані, але й самостійно аналізує їх на відповідність нормативам екологічної безпеки, миттєво інформуючи відповідальних осіб про загрози. Це дозволяє суттєво скоротити час реакції на інциденти та підвищити ефективність управлінських рішень.

У четвертому розділі проведено оцінку комерційного потенціалу розробленої інформаційної технології, яка має на меті фундаментальну трансформацію підходів до екологічного контролю. Прогнозування витрат на виконання науково-дослідної роботи по кожній зі статей витрат складе 348087,67 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР (з урахуванням коефіцієнта стадії) складатиме 386764,1 грн. Розрахунки показали високу економічну ефективність: вкладені інвестиції в даний проект окупляться приблизно через 9 місяців (0,72 року), а приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки протягом трьох років, складе 3699678,24 грн. Отримані результати свідчать про те, що розроблена система є не лише технічно досконалим, а й економічно вигідним інструментом для модернізації екологічного моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крижановський Є. М., Штельмах І. В., Панасюк В. Р. Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм міста Вінниці. LV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2026). Вінниця, 2026. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2026/paper/view/26795/22000 (дата звернення: 25.11.2025).

2. Водний кодекс України: Закон України від 06.06.1995 № 213/95-ВР. Відомості Верховної Ради України. 1995. № 24. Ст. 189.

3. Про затвердження Порядку здійснення державного моніторингу вод: Постанова Кабінету Міністрів України від 19.09.2018 № 815. URL: https://zakon.rada.gov.ua/laws/show/815-2018-%D0%BF (дата звернення: 29.11.2025).

4. Клименко М. О., Прищепа А. М., Вознюк Н. М. Моніторинг довкілля : підручник. Київ : Академія, 2006. 360 с.

5. Моніторинг та екологічна оцінка водних ресурсів України. Державне агентство водних ресурсів України. URL: http://monitoring.davr.gov.ua/ (дата звернення: 01.01.2025).

6. ThingSpeak Documentation. MathWorks. URL: https://www.mathworks.com/help/thingspeak/ (дата звернення: 19.11.2025).

7. EcoCity – мережа громадського моніторингу якості повітря. URL: https://eco-city.org.ua/ (дата звернення: 20.11.2025).

8. Greengard S. The Internet of Things. Cambridge : MIT Press, 2015. 232 p. (The MIT Press Essential Knowledge series).

9. Гатчин Ю. А., Телипко В. В. Інтернет речей: історія, технології, перспективи. Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. 2017. № 65. С. 45–52.

10. Richards M. Software Architecture Patterns. O’Reilly Media, Inc., 2015. 54 p.

11.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

12.Performance improvements in .NET 8. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/what-is-new/dotnet-8 (дата звернення: 19.09.2025).

13.PostgreSQL: Documentation: 16: JSON Types. PostgreSQL. URL: https://www.postgresql.org/docs/16/datatype-json.html (дата звернення: 25.09.2025).

14.Real-time ASP.NET with SignalR. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/signalr/ (дата звернення: 06.10.2025).

15.React Documentation. React. URL: https://react.dev/ (дата звернення: 20.01.2025).

16.Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). IETF Tools. RFC 7519. 2015. URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 22.10.2025).

17.Common web application architectures. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 25.11.2025).

18.Khorikov V. Functional C#: Primitive return types. Enterprise Craftsmanship. URL: https://enterprisecraftsmanship.com/ (дата звернення: 28.11.2025).

19.Козачко О. М. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2024. 45 с.

20.Про затвердження Нормативів екологічної безпеки водних об’єктів, що використовуються для потреб рибного господарства щодо гранично допустимих концентрацій органічних та мінеральних речовин: Наказ Міністерства аграрної політики та продовольства України від 30.07.2012 № 282. URL: https://zakon.rada.gov.ua/laws/show/z1332-12 (дата звернення: 05.09.2025).

Додаток А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____д.т.н., проф. Віталій МОКІН
«___» _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
ІНФОРМАЦІЙНА СИСТЕМА АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ МОНІТОРИНГУ ПОКАЗНИКІВ СТАНУ ПРИРОДНИХ ВОДОЙМ МІСТА ВІННИЦІ
08-34.МКР.002.02.000.ТЗ

Керівник: к.т.н., ас. каф. САІТ
_____Ігор ШТЕЛЬМАХ
«___» _____ 2025 р.

Розробив: студент гр. 2ІСТ-24м
_____ Володимир ПАНАСЮК
«___» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____ 2025р., та індивідуальне завдання на МКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.

2. Джерела розробки:

- AXELOS. ITIL® Foundation: ITIL 4 Edition. The Stationery Office, 2019. 214 с.
- Virtanen P., Gommers R., Oliphant T. E., та ін. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods. 2020. Т. 17. С. 261-272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

3. Мета і призначення роботи.

Метою роботи є підвищення ефективності моніторингу стану вод шляхом створення сучасної інформаційної системи, що забезпечує автоматизований збір, надійне збереження, аналіз та оперативну візуалізацію показників якості води.

Виправити

4. Вихідні дані для проведення робіт.

Відкриті дані порталу моніторингу вод Держводагентства України.

5. Методи дослідження.

Методи системного аналізу для формування вимог; принципи об’єктно-орієнтованого проєктування та патерн Clean Architecture для побудови архітектури ПЗ; методи реляційних баз даних для організації зберігання інформації; технології асинхронного програмування та веб-сокети (SignalR) для передачі даних у реальному часі.

6. Етапи роботи і терміни їх виконання:

а) Аналіз предметної області та існуючих рішень _____ – _____
б) Вибір оптимальних інформаційних технологій та стеку _____ – _____
в) Проєктування архітектури та бази даних системи _____ – _____
г) Програмна реалізація серверної та клієнтської частин _____ – _____
д) Тестування та експериментальна перевірка роботи _____ – _____
е) Виконання економічних розрахункі _____ – _____
є) Оформлення пояснювальної записки та графічних матеріалів _____ – _____

7. Очікувані результати та порядок реалізації.

Розроблена інформаційна система для автоматизованого збору та обробки телеметричних даних, виявлення екологічних аномалій у режимі реального часу та інтерактивної візуалізації показників якості води для підвищення ефективності управління водними ресурсами та оперативного реагування на забруднення

8. Вимоги до розробленої документації.

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання магістерських кваліфікаційних робіт для студентів спеціальності 126 «Інформаційні системи та технології» (освітня програма «Інформаційні технології аналізу даних та зображень»).

9. Порядок приймання роботи.

Публічний захист «___» _____ 2025 р.
Початок розробки «___» _____ 2025 р.
Граничні терміни виконання МКР «___» _____ 2025 р.

Розробив студент групи 2ІСТ-24м _____ Володимир ПАНАСЮК

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Інформаційна система аналізу та візуалізації даних моніторингу показників стану природних водойм міста Вінниці»

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,42 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____ (підпис)

Сергій ЖУКОВ, доц. каф. САІТ _____ (підпис)

Особа, відповідальна за перевірку _____ (підпис) Сергій ЖУКОВ

З висновком експертної комісії ознайомлений(-на)

Керівник _____ (підпис) Ігор ШТЕЛЬМАХ, к.т.н., ас. каф. САІТ

Здобувач _____ (підпис) Володимир ПАНАСЮК

Додаток В

Лістинг програми

Лістинг В.1 – Реалізація сервісу обробки вимірювань та динамічного парсингу JSON

```
public class DeviceMeasurementService : IDeviceMeasurementService
{
    private readonly IDeviceMeasurementRepository _measurementRepository;
    private readonly IDeviceRepository _deviceRepository;
    private readonly IThresholdService _thresholdService;
    private readonly IIoTHubService _hubService;

    public DeviceMeasurementService(
        IDeviceMeasurementRepository measurementRepository,
        IDeviceRepository deviceRepository,
        IThresholdService thresholdService,
        IIoTHubService hubService)
    {
        _measurementRepository = measurementRepository;
        _deviceRepository = deviceRepository;
        _thresholdService = thresholdService;
        _hubService = hubService;
    }

    public async Task<OperationResult<DeviceMeasurement>>
CreateAsync(DeviceMeasurement measurement)
    {
        var deviceResult = await
_deviceRepository.GetByIdAsync(measurement.DeviceId);
```

```
        if (!deviceResult.IsSuccess || deviceResult.Data == null)
        {
            return OperationResult<DeviceMeasurement>.NotFound($"Device with ID
{measurement.DeviceId} not found");
        }

        measurement.CreatedOn = DateTime.UtcNow;
        if (measurement.MeasurementDate == default)
        {
            measurement.MeasurementDate = measurement.CreatedOn;
        }

        var result = await _measurementRepository.AddAsync(measurement);

        if (result.IsSuccess && result.Data != null)
        {
            // Check thresholds
            var thresholdResult = await
_thresholdService.CheckThresholdsAsync(result.Data);
            if (thresholdResult.IsSuccess && thresholdResult.Data != null &&
thresholdResult.Data.Any())
            {
                var alert = new ThresholdAlert
                {
                    DeviceId = result.Data.DeviceId,
                    DeviceName = deviceResult.Data.Name,
                    MeasurementId = result.Data.Id,
                    MeasurementDate = result.Data.MeasurementDate,
                    ExceededThresholds = thresholdResult.Data
                };
```

```
            await _hubService.NotifyThresholdExceededAsync(alert);
        }

        // Notify measurement added
        await _hubService.NotifyMeasurementAddedAsync(result.Data);
    }

    return result;
}

public async Task<OperationResult<DeviceMeasurement>>
UpdateAsync(DeviceMeasurement measurement)
{
    var existingResult = await
_measurementRepository.GetByIdAsync(measurement.Id);
    if (!existingResult.IsSuccess || existingResult.Data == null)
    {
        return OperationResult<DeviceMeasurement>.NotFound($"Measurement
with ID {measurement.Id} not found");
    }

    var existing = existingResult.Data;
    existing.RawData = measurement.RawData;
    existing.DataFormat = measurement.DataFormat;
    existing.MeasurementDate = measurement.MeasurementDate;
    existing.ParsedSuccessfully = measurement.ParsedSuccessfully;
    existing.ParsingErrors = measurement.ParsingErrors;

    var result = await _measurementRepository.UpdateAsync(existing);
```

```
        if (result.IsSuccess && result.Data != null)
        {
            await _hubService.NotifyMeasurementUpdatedAsync(result.Data);
        }

        return result;
    }

    public async Task<OperationResult> DeleteAsync(Guid id)
    {
        var measurementResult = await _measurementRepository.GetByIdAsync(id);
        if (!measurementResult.IsSuccess || measurementResult.Data == null)
        {
            return OperationResult.NotFound($"Measurement with ID {id} not
found");
        }

        var deviceId = measurementResult.Data.DeviceId;
        var result = await
_measurementRepository.DeleteAsync(measurementResult.Data);

        if (result.IsSuccess)
        {
            await _hubService.NotifyMeasurementDeletedAsync(deviceId, id);
        }

        return result;
    }

    public Task<OperationResult<DeviceMeasurement>> GetByIdAsync(Guid id)
```

```
    {
        return _measurementRepository.GetByIdAsync(id);
    }

    public Task<OperationResult<List<DeviceMeasurement>>>
GetByDeviceIdAsync(
        Guid deviceId,
        DateTime? startDate = null,
        DateTime? endDate = null,
        int? limit = null)
    {
        return _measurementRepository.GetByDeviceIdAsync(deviceId, startDate,
endDate, limit);
    }

    public Task<OperationResult<DeviceMeasurement>>
GetLatestByDeviceIdAsync(Guid deviceId)
    {
        return _measurementRepository.GetLatestByDeviceIdAsync(deviceId);
    }
}
```

Лістинг В.2 – Фоновий сервіс виявлення перевищень граничних значень

```
public class ThresholdTrackingService
{
    // Key: DeviceId_FieldId, Value: Current threshold status
    private readonly ConcurrentDictionary<string, string> _thresholdStates = new();
```

```
    public bool ShouldNotify(Guid deviceId, Guid fieldId, string newStatus)
    {
        var key = $"{deviceId}_{fieldId}";

        // If Normal, always reset state and don't notify
        if (newStatus == Constants.Thresholds.Normal)
        {
            _thresholdStates.TryRemove(key, out _);
            return false;
        }

        // Check if we already notified about this status
        if (_thresholdStates.TryGetValue(key, out var currentStatus) &&
currentStatus == newStatus)
        {
            return false; // Already notified about this status
        }

        // Update state and notify
        _thresholdStates[key] = newStatus;
        return true;
    }

    public void ResetField(Guid deviceId, Guid fieldId)
    {
        var key = $"{deviceId}_{fieldId}";
        _thresholdStates.TryRemove(key, out _);
    }
```

```
    public void ResetDevice(Guid deviceId)
    {
        var keysToRemove = _thresholdStates.Keys.Where(k =>
k.StartsWith($"{deviceId}_")).ToList();
        foreach (var key in keysToRemove)
        {
            _thresholdStates.TryRemove(key, out _);
        }
    }
}
```

Лістинг В.3 – Конфігурація SignalR Hub для клієнтської взаємодії

```
public class IoTHub : Hub
{
    public override async Task OnConnectedAsync()
    {
        // Add all connected clients to global group
        await Groups.AddToGroupAsync(Context.ConnectionId,
Constants.SignalR.Groups.AllDevices);
        await base.OnConnectedAsync();
    }

    public override async Task OnDisconnectedAsync(Exception? exception)
    {
        await Groups.RemoveFromGroupAsync(Context.ConnectionId,
Constants.SignalR.Groups.AllDevices);
        await base.OnDisconnectedAsync(exception);
    }
}
```

Лістинг В.4 – Клієнтський метод для підключення до WebSocket

```
const startConnection = async () => {
  try {
    // Build connection
    const connection = new signalR.HubConnectionBuilder()
      .withUrl(SIGNALR_HUB_URL, {
        withCredentials: false,
        transport: signalR.HttpTransportType.WebSockets,
        skipNegotiation: true,
      })
      .withAutomaticReconnect({
        nextRetryDelayInMilliseconds: (retryContext) => {
          // Exponential backoff: 0, 2, 10, 30 seconds, then 30 seconds
          if (retryContext.previousRetryCount === 0) return 0;
          if (retryContext.previousRetryCount === 1) return 2000;
          if (retryContext.previousRetryCount === 2) return 10000;
          return 30000;
        },
      })
      .configureLogging(signalR.LogLevel.Information)
      .build();

    connection.on(SIGNALR_EVENTS.MEASUREMENT_ADDED,
(measurement: DeviceMeasurement) => {
      addMeasurement(String(measurement.deviceId), measurement);
    });
```

```
    connection.on(SIGNALR_EVENTS.MEASUREMENT_UPDATED,
(measurement: DeviceMeasurement) => {
      updateMeasurement(String(measurement.deviceId), measurement);
    });

    connection.on(SIGNALR_EVENTS.MEASUREMENT_DELETED,
(payload: { deviceId: string; measurementId: string }) => {
      deleteMeasurement(String(payload.deviceId),
String(payload.measurementId));
    });

    connection.on(SIGNALR_EVENTS.THRESHOLD_EXCEEDED, (alert:
ThresholdAlert) => {
      addAlert(alert);
    });

    // Connection lifecycle events
    connection.onreconnecting((error) => {
      console.log('SignalR reconnecting...', error);
      if (isMounted) {
        setIsConnected(false);
        setConnectionError('Переєднання...');
      }
    });

    connection.onreconnected((connectionId) => {
      console.log('SignalR reconnected:', connectionId);
      if (isMounted) {
        setIsConnected(true);
        setConnectionError(null);
```

```
    }
  });

  connection.onclose((error) => {
    console.log('SignalR connection closed:', error);
    if (isMounted) {
      setIsConnected(false);
      setConnectionError(error?.message || 'З\'єднання закрито');

      // Try to reconnect after 5 seconds
      reconnectTimeoutRef.current = setTimeout(() => {
        if (isMounted) {
          console.log('Attempting to reconnect...');
          startConnection();
        }
      }, 5000);
    }
  });

  // Start connection
  await connection.start();
  console.log('SignalR connected');

  if (isMounted) {
    connectionRef.current = connection;
    setIsConnected(true);
    setConnectionError(null);
  }
} catch (error) {
```

```
    console.error('SignalR connection error:', error);
    if (isMounted) {
      setIsConnected(false);
      setConnectionError(
        error instanceof Error ? error.message : 'Помилка з\'єднання'
      );

      // Retry connection after 5 seconds
      reconnectTimeoutRef.current = setTimeout(() => {
        if (isMounted) {
          startConnection();
        }
      }, 5000);
    }
  }
};
```

Додаток Г

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ МОНІТОРИНГУ ПОКАЗНИКІВ СТАНУ ПРИРОДНИХ ВОДОЙМ МІСТА ВІННИЦІ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

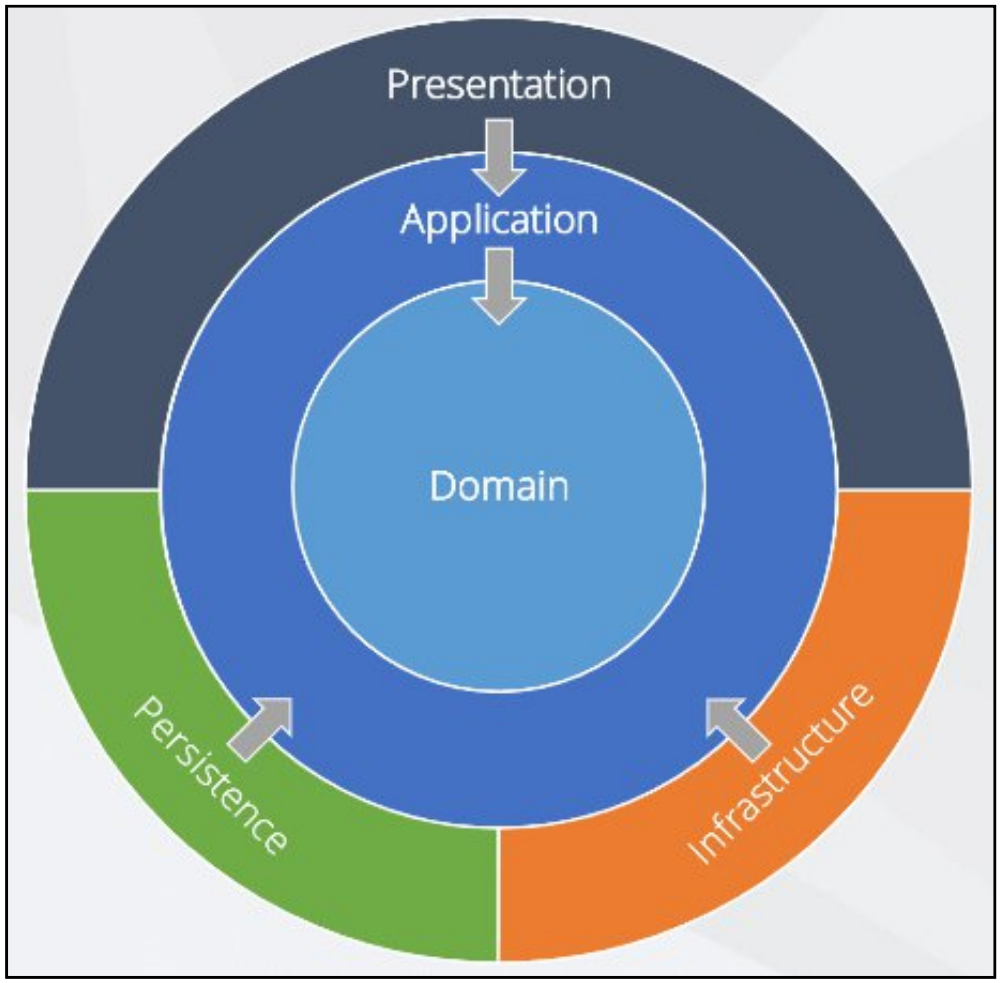


Рисунок Г.1 – Концептуальна схема рівнів «Чистої архітектури» із відображенням правила залежностей

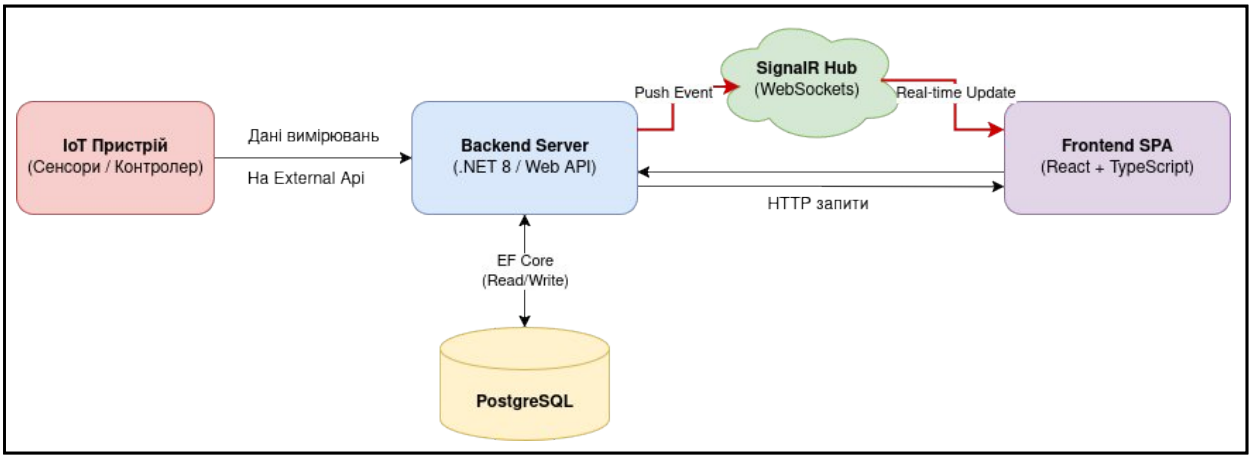


Рисунок Г.2 – Узагальнена схема технологічного стеку та потоків даних в інформаційній системі

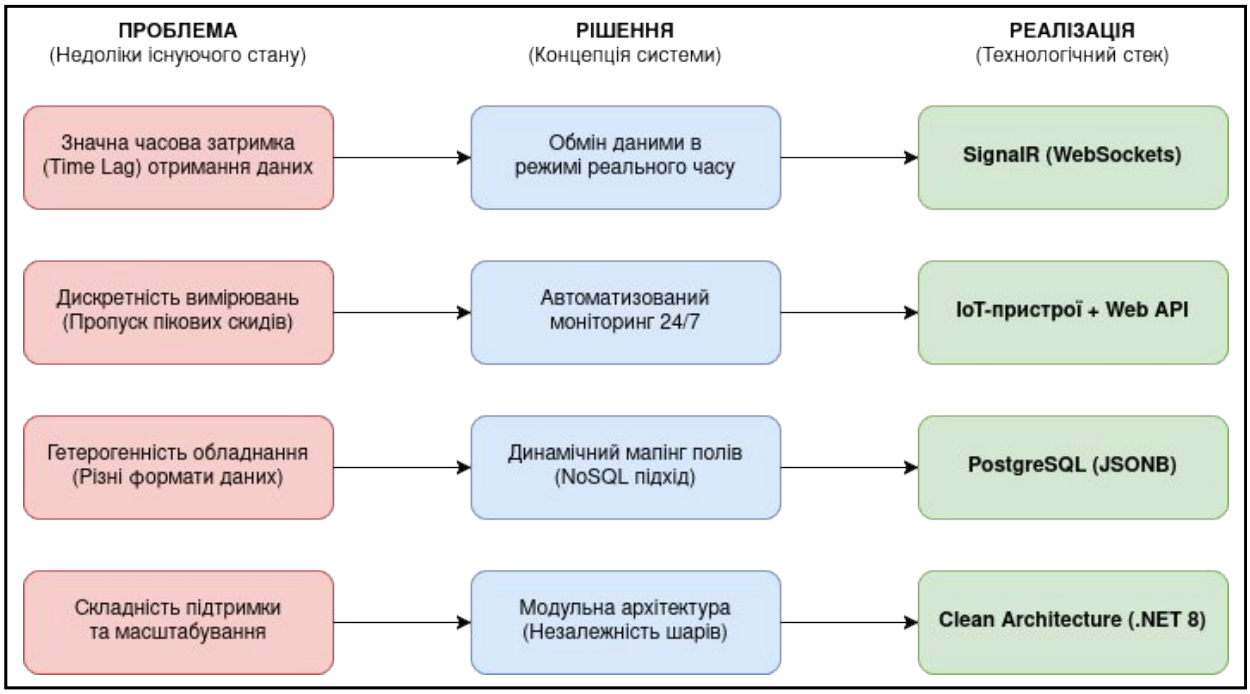


Рисунок Г.3 – Концептуальна схема переходу від проблематики предметної області до технологічних рішень

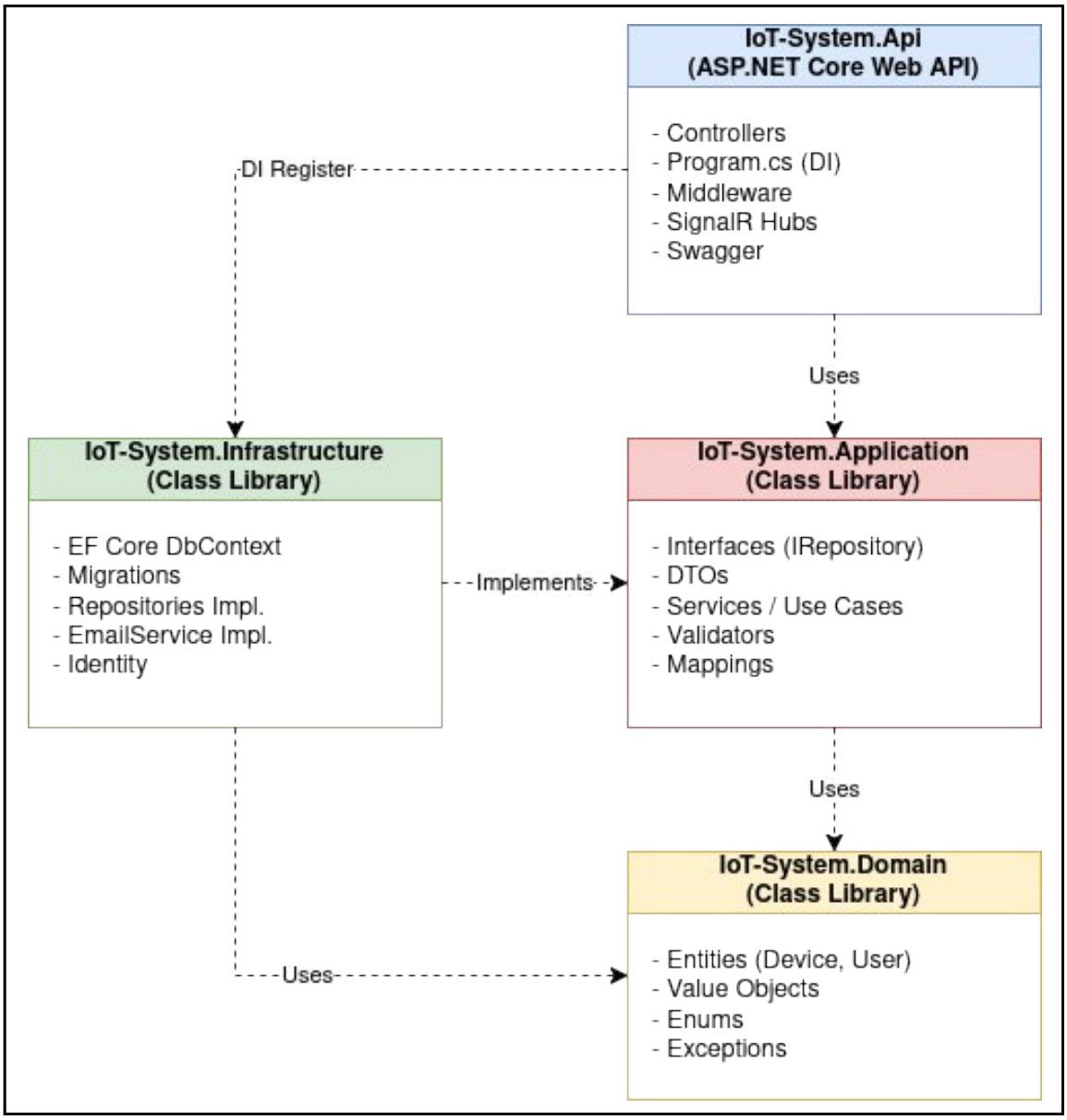


Рисунок Г.4 – Структурна схема архітектури рішення на базі Clean Architecture

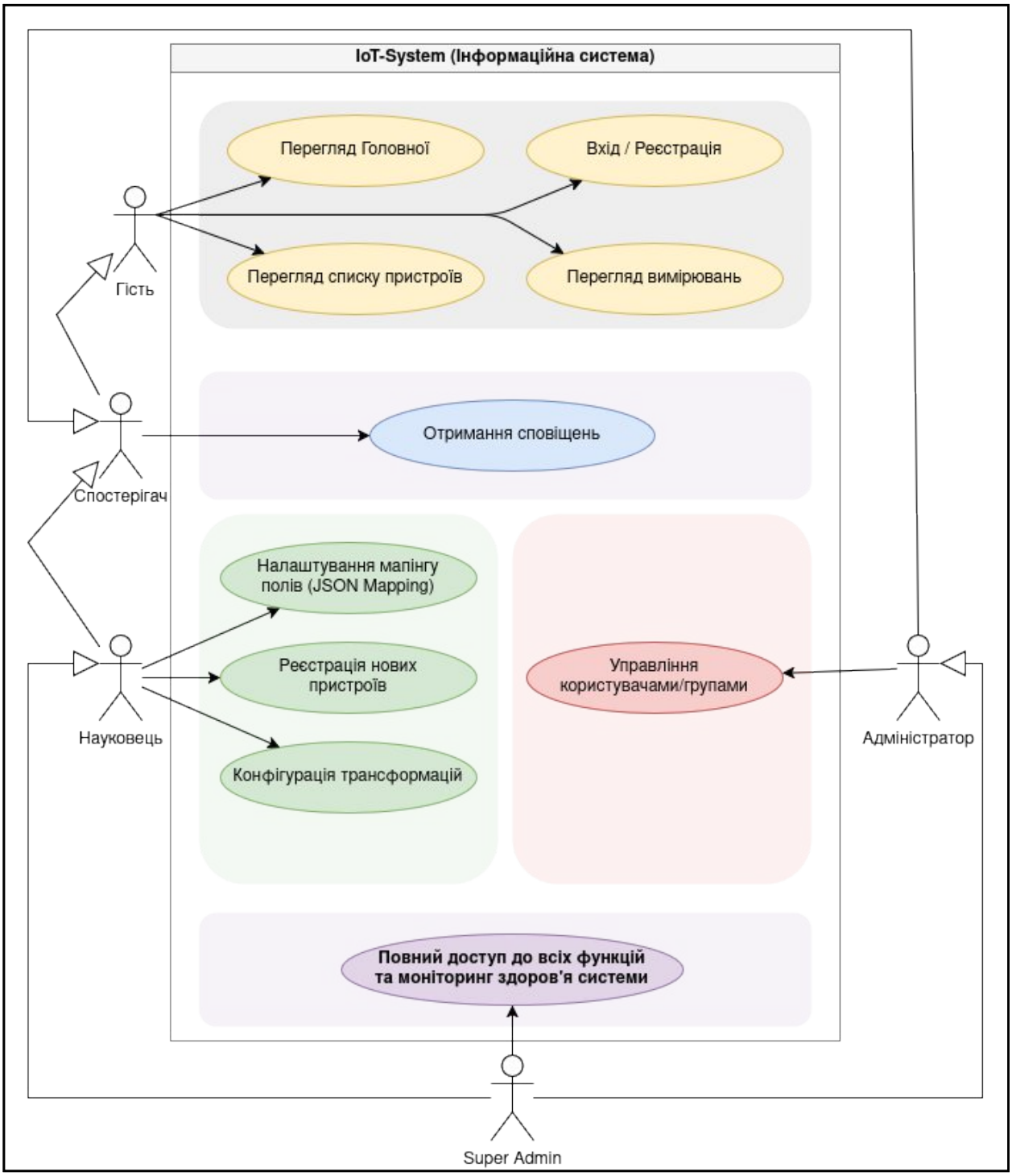


Рисунок Г.5 – Діаграма варіантів використання (Use Case Diagram) інформаційної системи

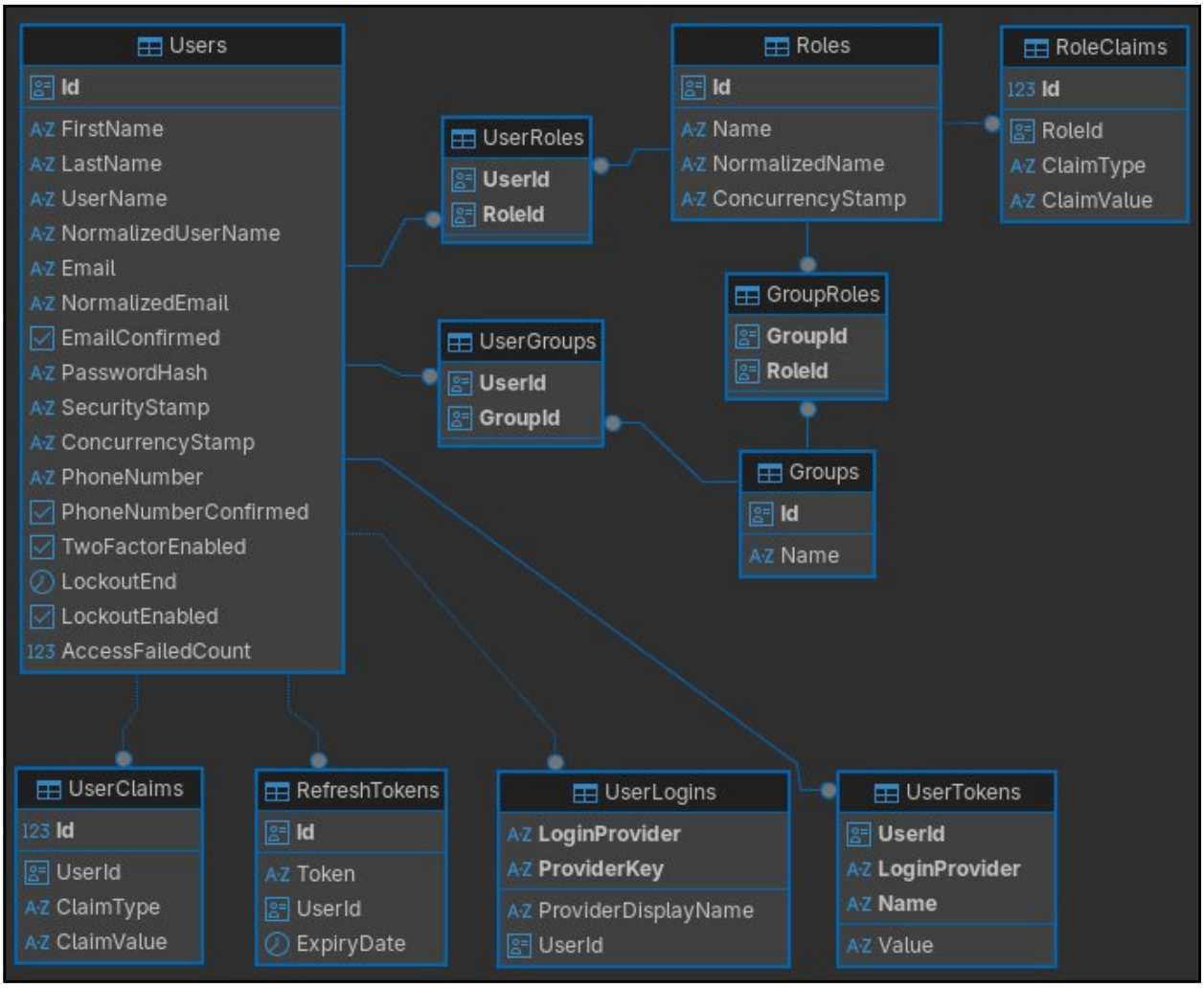


Рисунок Г.6 – ER-діаграма бази даних інформаційної системи ідентифікації та доступу

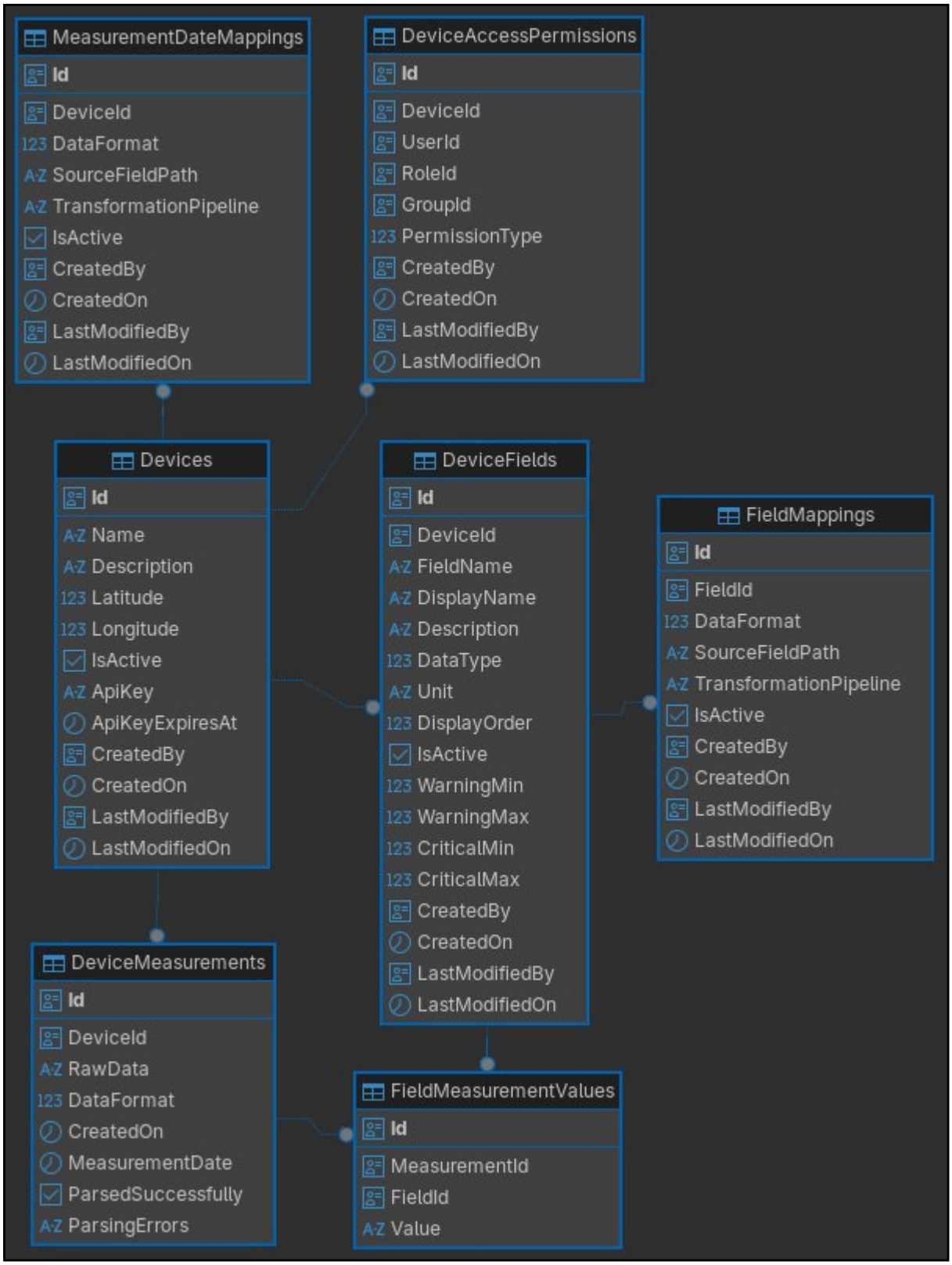


Рисунок Г.7 – ER-діаграма бази даних інформаційної системи предметної області моніторингу

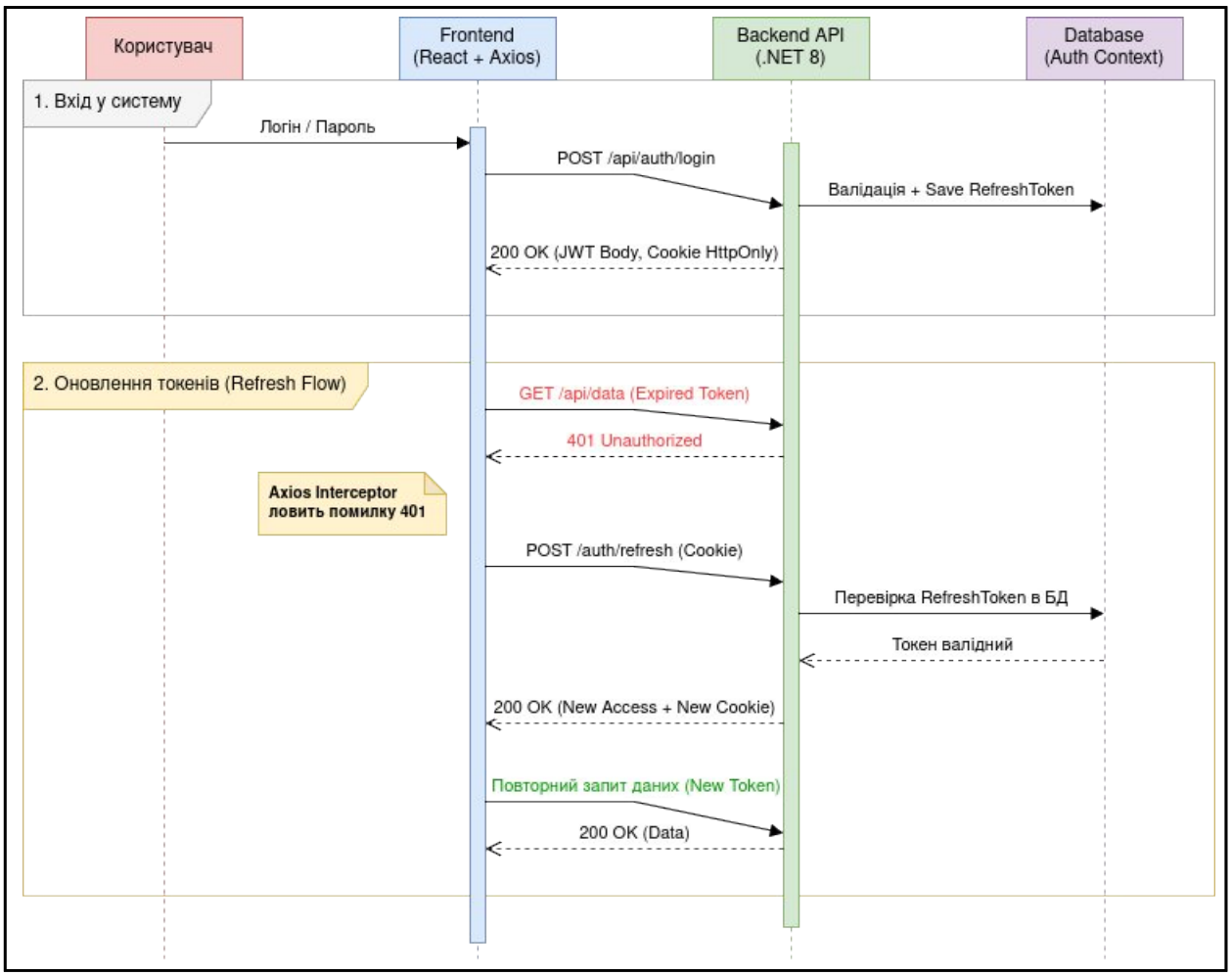


Рисунок Г.8 – Діаграма послідовності процесу безпечної автентифікації користувача (JWT + Refresh Token)

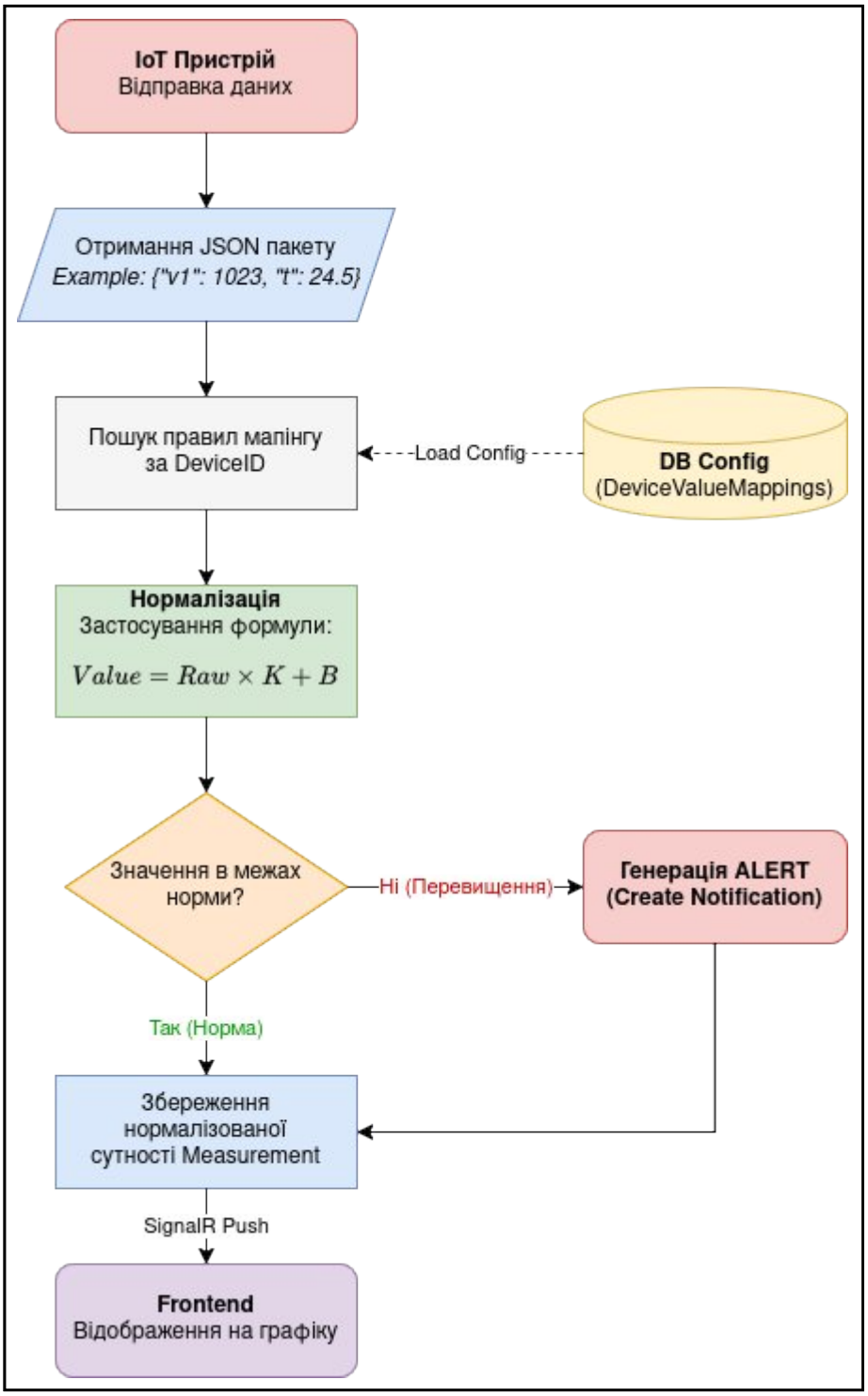
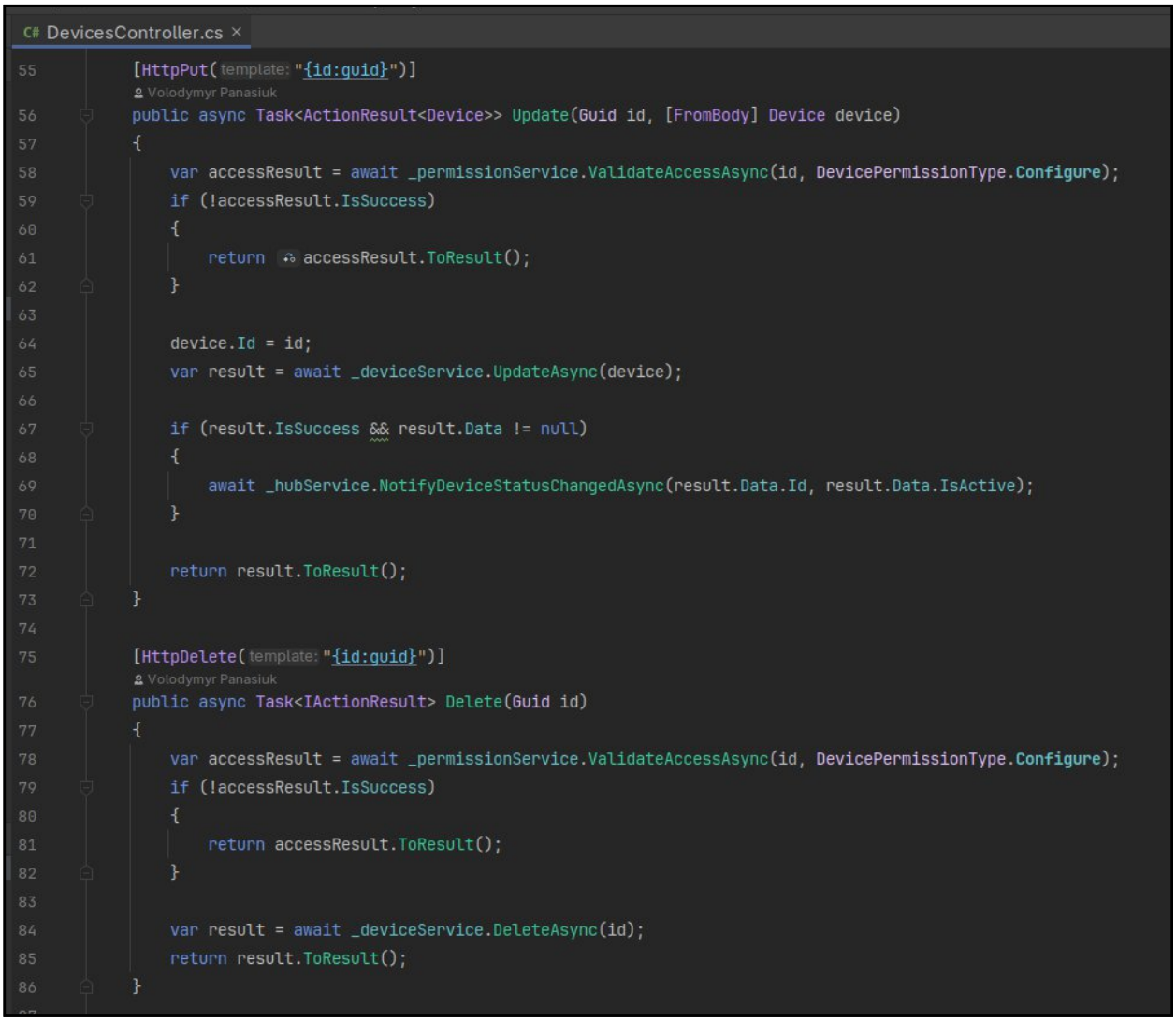


Рисунок Г.9 – Алгоритм динамічної обробки та нормалізації даних від IoT-пристроїв



```
C# DevicesController.cs ×
55      [HttpPut(template: "{id:guid}")]
        Volodymyr Panasiuk
56      public async Task<ActionResult<Device>> Update(Guid id, [FromBody] Device device)
57      {
58          var accessResult = await _permissionService.ValidateAccessAsync(id, DevicePermissionType.Configure);
59          if (!accessResult.IsSuccess)
60          {
61              return accessResult.ToResult();
62          }
63
64          device.Id = id;
65          var result = await _deviceService.UpdateAsync(device);
66
67          if (result.IsSuccess && result.Data != null)
68          {
69              await _hubService.NotifyDeviceStatusChangedAsync(result.Data.Id, result.Data.IsActive);
70          }
71
72          return result.ToResult();
73      }
74
75      [HttpDelete(template: "{id:guid}")]
        Volodymyr Panasiuk
76      public async Task<IActionResult> Delete(Guid id)
77      {
78          var accessResult = await _permissionService.ValidateAccessAsync(id, DevicePermissionType.Configure);
79          if (!accessResult.IsSuccess)
80          {
81              return accessResult.ToResult();
82          }
83
84          var result = await _deviceService.DeleteAsync(id);
85          return result.ToResult();
86      }
```

Рисунок Г.10 – Фрагмент коду DevicesController із використанням OperationResult

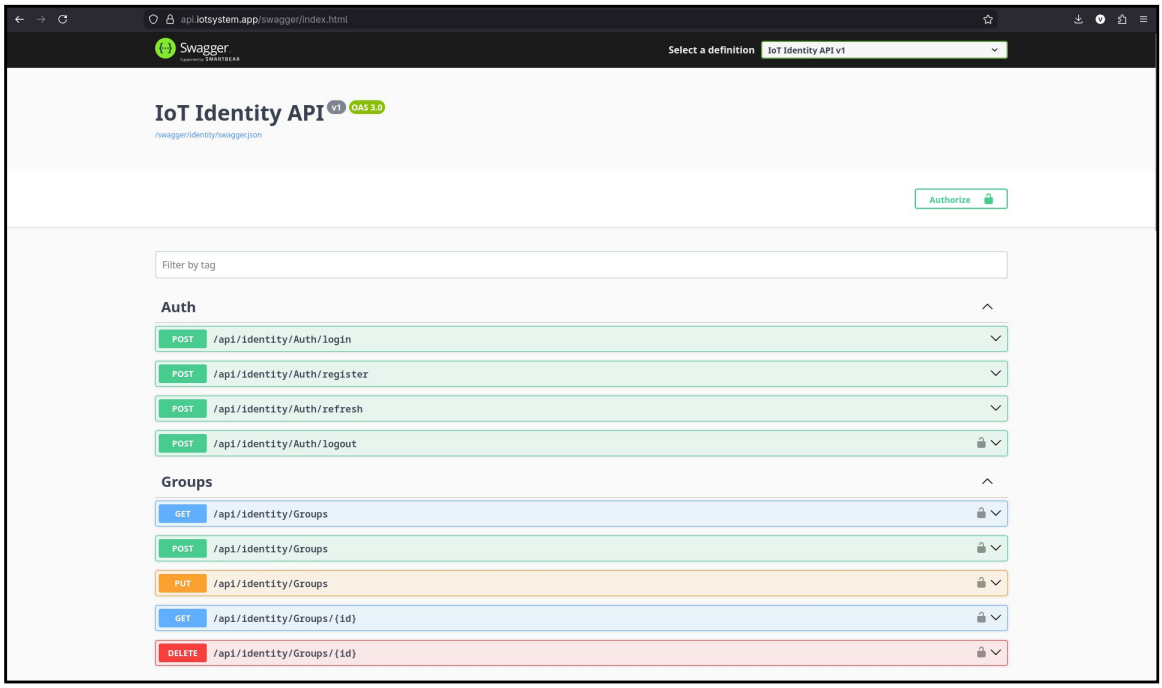


Рисунок Г.11 – Інтерфейс Swagger UI з відображенням контролерів та схем даних для підсистеми ідентифікації та доступу

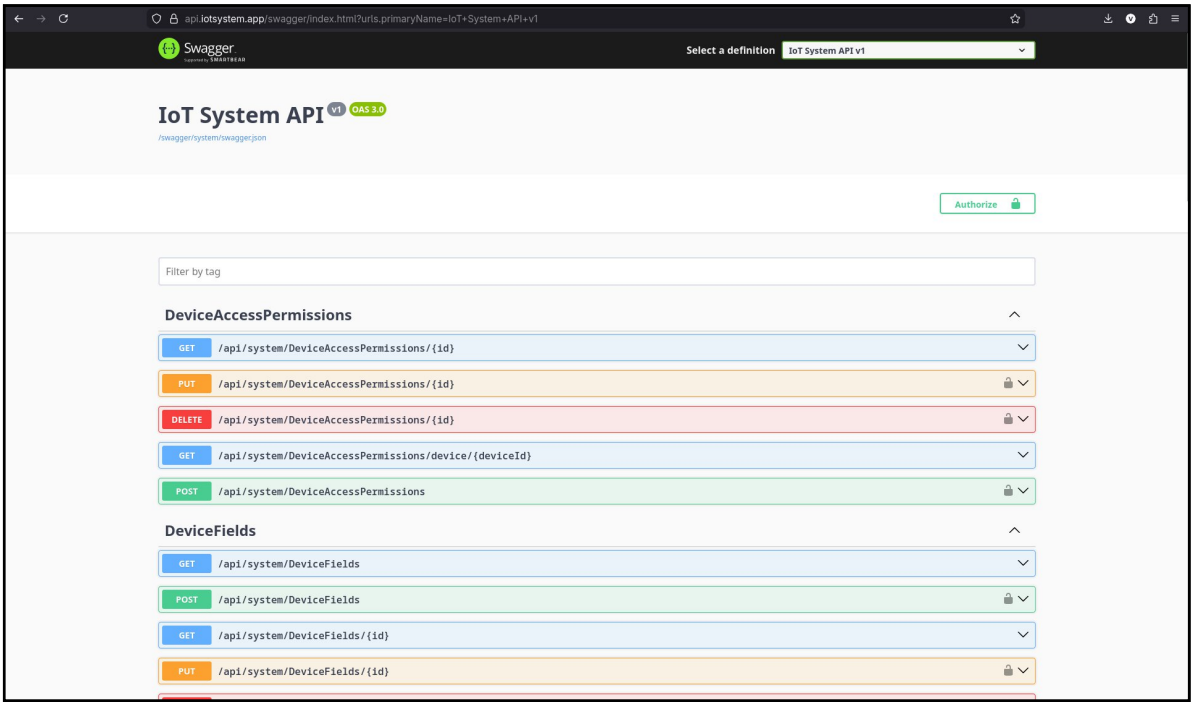


Рисунок Г.12 – Інтерфейс Swagger UI з відображенням контролерів та схем даних для підсистеми конфігурації та управління пристроями

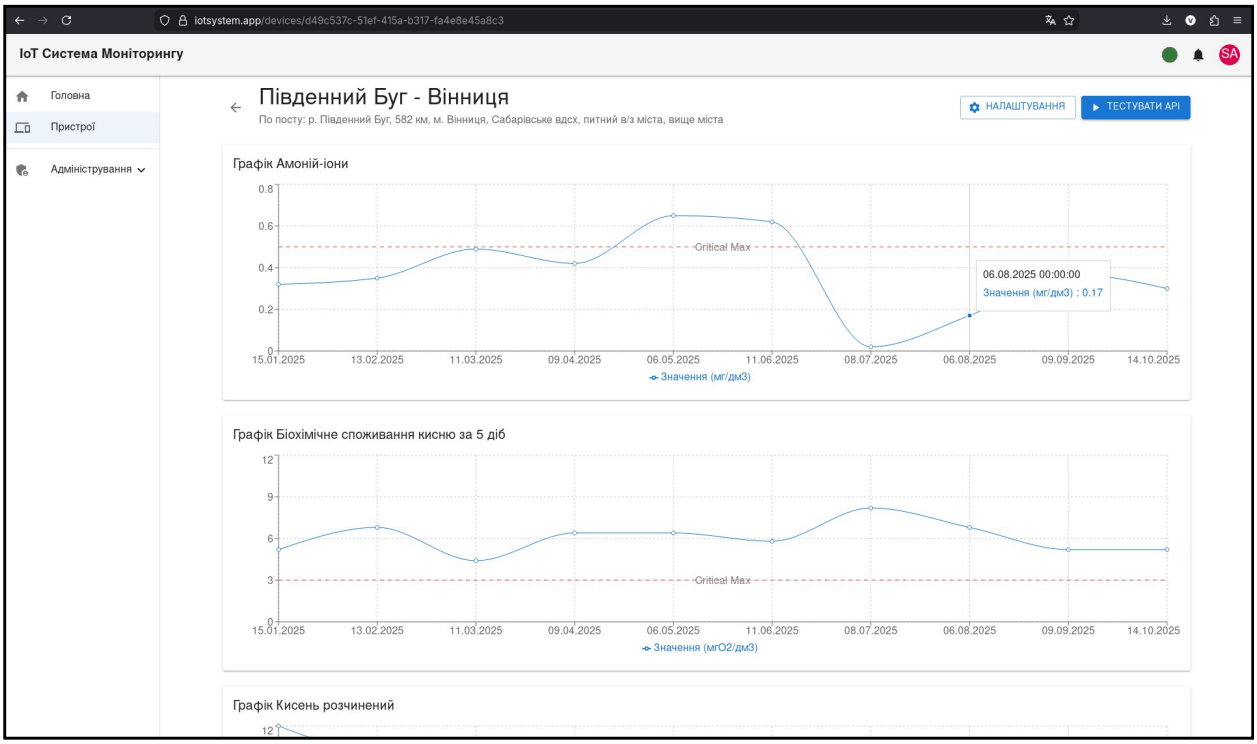


Рисунок Г.13 – Інтерфейс візуалізації даних з графіками показників якості води

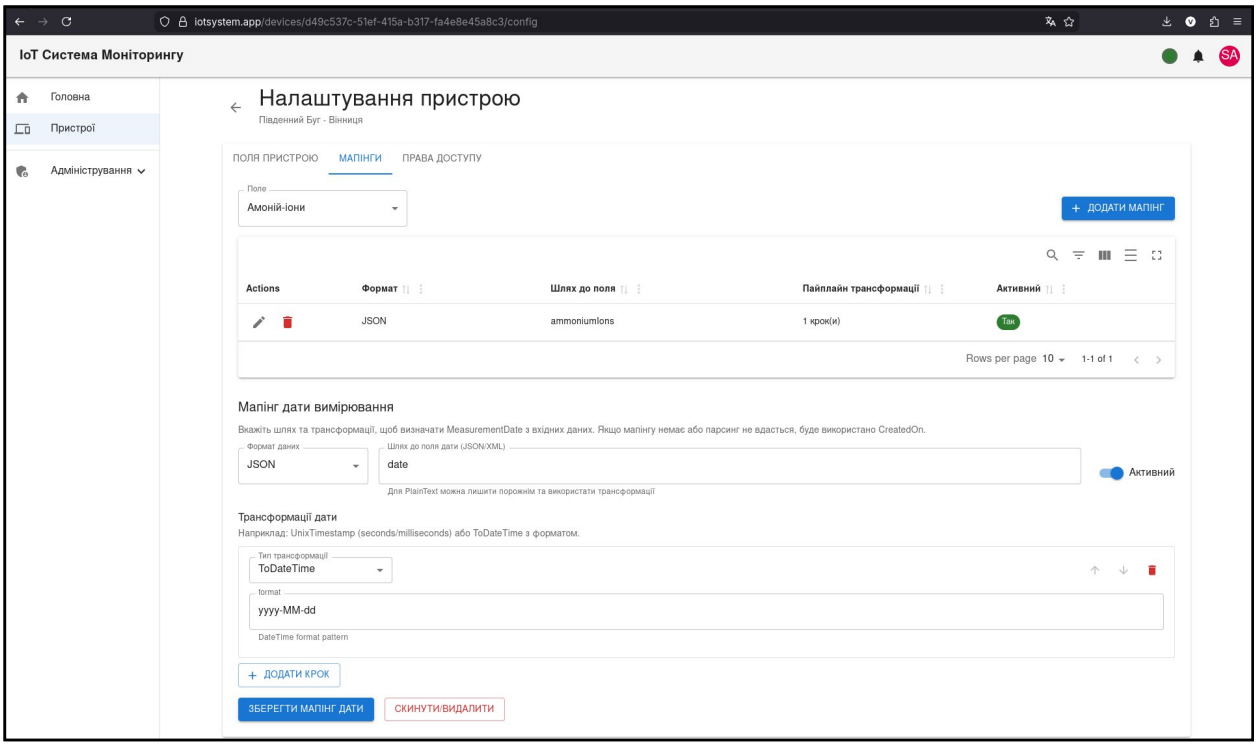


Рисунок Г.14 – Сторінка налаштування конфігурації та мапінгу полів IoT-пристрою

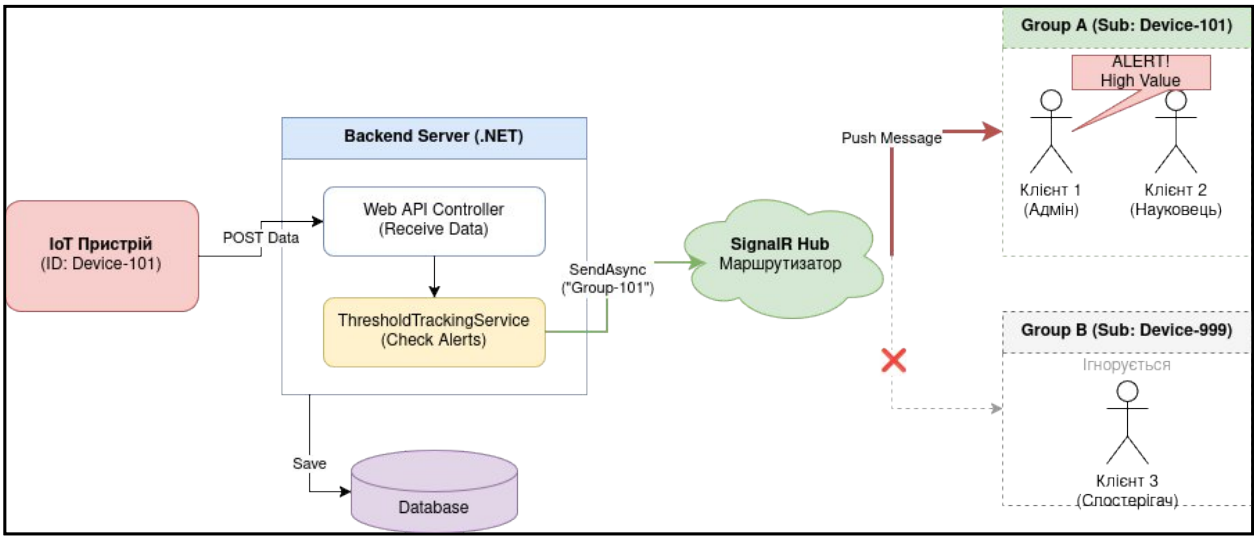


Рисунок Г.15 – Схема маршрутизації повідомлень реального часу через SignalR Hub

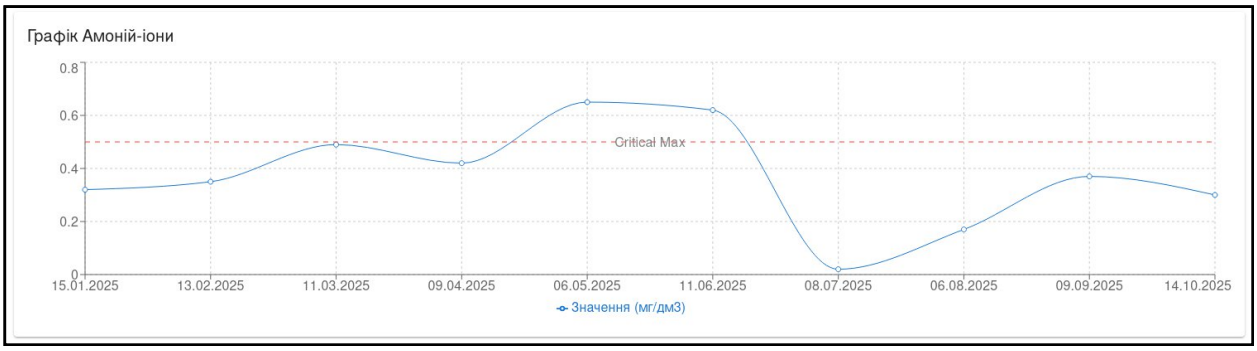


Рисунок Г.16 – Візуалізація динаміки амонію з відображенням зон перевищення ГДК

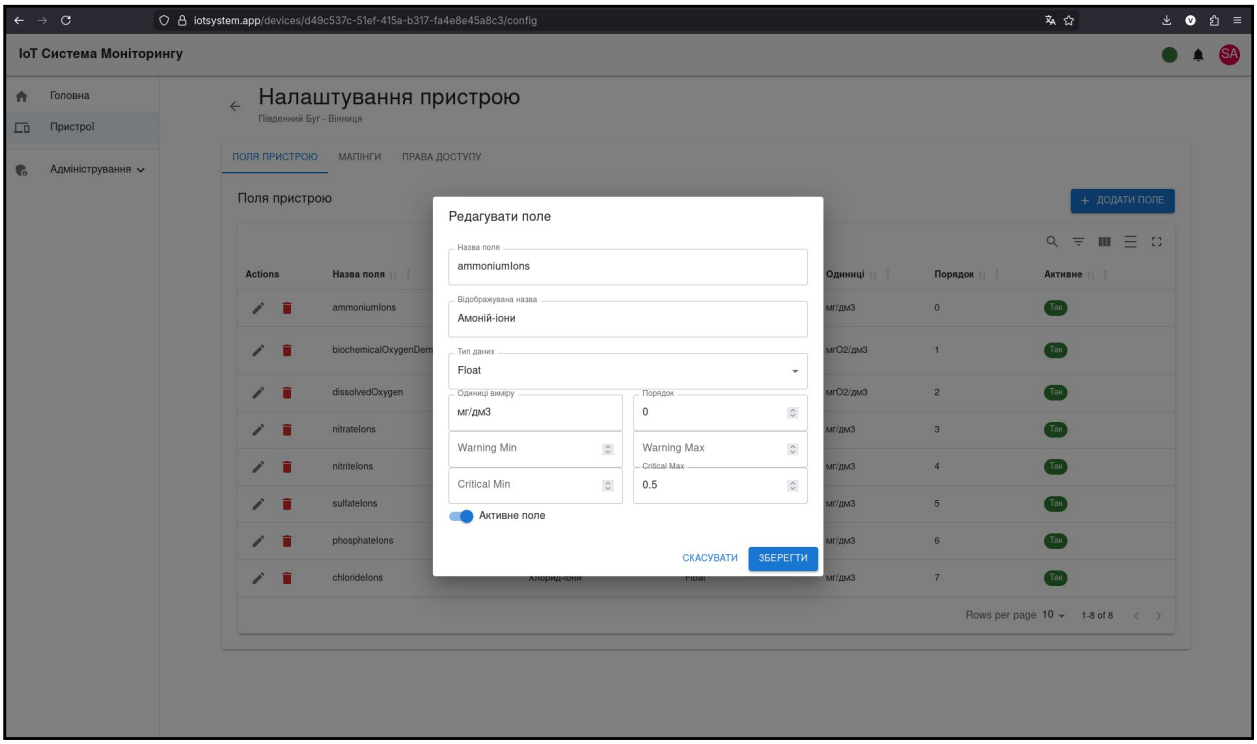


Рисунок Г.17 – Інтерфейс налаштування порогових значень для моніторингу якості води