Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Кіберфізична система моніторингу стану води. Програмна частина
Назва теми

КвРКІ 210233.21.02.07 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-21-2 _____ Дмитро ГОРБАТЮК
Підпис — Ініціали, прізвище

Керівник _____ Петро ВІЖЕВСЬКИЙ
Підпис, дата — Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата — Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис — Ініціали, прізвище

«9» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Дмитру ГОРБАТЮКУ
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Кіберфізична система моніторингу стану води. Програмна частина

Керівник проекту (роботи) Петро ВІЖЕВСЬКИЙ, асистент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз предметної області та постановка задачі розробки програмної частини кіберфізичної системи моніторингу стану води

Проєктування системи обробки інформації у кіберфізичній системі адаптивного застосування моніторингових елементів розвідувального БПЛА

Практична реалізація програмних модулів мобільного додатку системи моніторингу стану води та оцінка його функціональності

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 - аналіз предметної області та постановка задачі розробки програмної частини кіберфізичної системи моніторингу стану води	01.03.2025	виконано
4	Робота над розділом 2 - розробка архітектурно-функціональної моделі програмної частини кіберфізичної системи моніторингу води	01.04.2025	виконано
5	Робота над розділом 3 - практична реалізація програмних модулів мобільного додатку	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Дмитро ГОРБАТЮК
Підпис Ініціали, прізвище

Керівник роботи _____ Петро ВІЖЕВСЬКИЙ
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 210233.21.02.07 ПЗ	Пояснювальна записка	68		
			Графічні матеріали			
2		КвРКІ 210233.21.02.07 Е8	Архітектура програмної	1		
			частини			
3		КвРКІ 210233.21.02.07 Е8	Клієнтський модуль	1		
			керування станом			
4		КвРКІ 210233.21.02.07 Е8	Схема результатів роботи	1		
			програми			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 210233.21.02.07 ВП		
Розробив		Горбатюк			Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Віжевський				У	1	1
Н. контр.		Кисіль				ХНУ, КІ2-21-2		
Затв.		Павлова						

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Кіберфізична система моніторингу стану води. Програмна частина».

Автор роботи: Дмитро ГОРБАТЮК.

Керівник роботи: Петро ВІЖЕВСЬКИЙ.

Пояснювальна записка: 68 с., 24 рис., 2 табл., 3 дод., 46 джерел.

Графічна частина: 3 креслення.

КІБЕРФІЗИЧНА СИСТЕМА, МОНІТОРИНГ СТАНУ ВОДИ, ПРОГРАМНА ЧАСТИНА, МОБІЛЬНИЙ ДОДАТОК, REACT NATIVE, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВІЗУАЛІЗАЦІЯ ДАНИХ.

Метою дипломної роботи є дослідження архітектурних та функціональних аспектів програмної частини кіберфізичної системи моніторингу стану води, а також розробка та тестування клієнтського мобільного додатку, призначеного для ефективного збору (через API), візуалізації та аналізу даних про якість водних ресурсів.

Об’єктом дослідження є процеси функціонування програмного забезпечення клієнтської частини в кіберфізичній системі моніторингу стану води, включаючи взаємодію з серверними компонентами, обробку та представлення даних користувачеві.

Предметом дослідження є архітектура, програмні модулі, технології реалізації та методи візуалізації даних у мобільному додатку для системи моніторингу стану води, розробленому з використанням фреймворку React Native.

Під час проведення даного дослідження були використані методи системного аналізу, об'єктно-орієнтованого проектування, принципи розробки мобільних додатків, а також проведено огляд науково-технічної літератури та існуючих програмних рішень у сфері кіберфізичних систем та моніторингу довкілля.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП

Сучасний етап розвитку технологій характеризується повсюдним впровадженням кіберфізичних систем (КФС), які інтегрують обчислювальні ресурси з фізичними процесами у різноманітних сферах, починаючи від інтелектуальних виробництв та транспортних мереж до систем управління міською інфраструктурою. Така інтеграція відкриває принципово нові можливості для автоматизації, контролю та оптимізації складних процесів. Важливим напрямком застосування КФС є моніторинг стану навколишнього середовища, зокрема водних ресурсів. Проблема забезпечення якісного та своєчасного контролю за станом водних об'єктів загострюється в умовах зростаючого антропогенного навантаження, кліматичних змін та необхідності раціонального водокористування. У цьому контексті, ефективність збору, передачі, обробки та аналізу даних про фізичні, хімічні та біологічні параметри води значною мірою детермінується якістю та функціональністю програмного забезпечення, яке є невід'ємною та ключовою компонентою будь-якої КФС. Саме програмна частина забезпечує логіку функціонування системи, обробку сигналів від сенсорів, агрегацію даних, їх зберігання, аналіз та представлення користувачеві у зручному для прийняття рішень форматі. З огляду на зростаючі вимоги до оперативності, точності та комплексності екологічної інформації, дослідження та розробка ефективних програмних компонентів для кіберфізичних систем моніторингу стану води набувають особливої актуальності. Існуючі програмні рішення часто є або надто універсальними, що не враховують специфіку моніторингу водних об'єктів, або, навпаки, вузькоспеціалізованими та негнучкими для адаптації до мінливих умов та завдань. Таким чином, існує нагальна потреба у розробці архітектурно виважених та функціонально насичених програмних рішень, здатних забезпечити високий рівень автоматизації та інформаційної підтримки процесів моніторингу водних ресурсів. У даному дослідженні розглядатиметься саме проблематика створення такої спеціалізованої програмної реалізації для КФС моніторингу стану води.

Метою дипломної роботи є дослідження архітектурних та функціональних аспектів програмної частини кіберфізичної системи моніторингу стану води, а також розробка її ключових компонентів, спрямованих на забезпечення ефективного збору, багатоетапної обробки, надійного зберігання та інформативної візуалізації даних. Досягнення поставленої мети передбачає розробку програмних модулів, здатних інтегруватися з апаратними засобами збору даних, реалізацію алгоритмів попередньої обробки та аналізу інформації, а також створення інтуїтивно зрозумілого інтерфейсу користувача для взаємодії з системою. Результатом роботи має стати обґрунтована концепція та прототип програмного комплексу, що демонструє життєздатність запропонованих рішень.

Об’єктом дослідження є процеси функціонування програмного забезпечення у складі кіберфізичних систем, призначених для здійснення моніторингу параметрів водного середовища. Це охоплює процеси збору даних від різнорідних сенсорних пристроїв, їх передачі по комунікаційних каналах, довготривалого зберігання у структурованих базах даних, аналітичної обробки з метою виявлення закономірностей та аномалій, а також візуального представлення результатів моніторингу кінцевим користувачам. До об'єкта також належать питання взаємодії програмних компонентів між собою та з апаратними елементами системи, забезпечення надійності та безпеки функціонування програмного комплексу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ПРОГРАМНОЇ ЧАСТИНИ КІБЕРФІЗИЧНОЇ СИСТЕМИ МОНІТОРИНГУ СТАНУ ВОДИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Водні ресурси є основою життя та функціонування суспільства, але їхня якість і доступність перебувають під постійною загрозою. Забруднення промисловими стоками, сільськогосподарськими хімікатами, комунальними відходами, а також наслідки зміни клімату вимагають безперервного та точного контролю стану водних об'єктів - від джерел питного водопостачання до річок та озер [16]. Однак традиційні підходи до моніторингу стикаються з низкою фундаментальних обмежень [20].

Класичний метод моніторингу покладається на періодичний відбір проб води у визначених місцях з подальшим лабораторним аналізом. Цей процес є невід'ємною частиною екологічного контролю, проте він має суттєві недоліки, які представлено в таблиці 1.1.

Таблиця 1.1 − Недоліки класичного методу моніторингу

1.	Часова дискретність	- відбір проб відбувається епізодично (раз на тиждень, місяць тощо). Це означає, що короткочасні, але потенційно небезпечні події забруднення [11] (наприклад, аварійні скиди) можуть залишитися непоміченими, якщо вони відбулися між відборами
2.	Просторова обмеженість	- обмеженість точок відбору проб, зумовлена логістичними та фінансовими чинниками, забезпечує лише фрагментарне уявлення

Кінець таблиці 1.1

3.	Операційна затримка	- час між відбором проби, її транспортуванням до лабораторії, проведенням аналізів та отриманням результатів може становити від кількох годин до кількох днів або навіть тижнів. Така затримка унеможливлює оперативне реагування на критичні ситуації
4.	Високі витрати	- регулярний відбір проб, транспортування та складні лабораторні аналізи потребують значних фінансових та людських ресурсів.

Ці обмеження стають особливо критичними в умовах зростаючої динаміки змін у навколишньому середовищі [7, 23] та потреби в превентивних заходах для захисту водних ресурсів. Необхідність у більш оперативних, всеосяжних та ефективних методах моніторингу стимулювала розвиток нових технологічних рішень [20].

Саме тут на перший план виходить концепція кіберфізичних систем (КФС). КФС для моніторингу води інтегрує фізичний світ (водне середовище та процеси в ньому) з кіберпростором (обчислення, аналіз даних, комунікації) за допомогою мережі сенсорів, виконавчих механізмів (за потреби) та інтелектуального програмного забезпечення [1515]. Цей підхід дозволяє перейти від дискретного та запізнілого моніторингу до безперервного контролю в реальному часі.

Рисунок 1.1 − Стаціонарний пристрій моніторингу якості води

Розміщення стаціонарних (рисунок 1.1) або мобільних сенсорних вузлів безпосередньо у водному середовищі дає змогу автоматично збирати дані про ключові параметри (температуру, pH, каламутність, рівень кисню, наявність специфічних забруднювачів тощо) з високою частотою [13].

Однак самі по собі сенсори генерують лише потоки необроблених даних [14]. Справжня цінність КФС полягає в її здатності перетворювати ці дані на значущу інформацію та actionable insights [15]. І ключову роль у цьому процесі відіграє саме програмна частина системи [13]. Вона вирішує комплекс завдань [40, 43], що виходять далеко за межі простого збору даних:

Агрегація та передача даних: Збір показників від численних, потенційно географічно розподілених сенсорів та їхня надійна передача до центрального сервера чи хмарної платформи.

Валідація та очищення даних: Виявлення та обробка помилкових показників, шумів, пропусків, спричинених несправностями сенсорів чи проблемами зв'язку.

Обробка та аналіз: Застосування алгоритмів для розрахунку похідних показників, виявлення трендів, кореляцій, аномалій та відхилень від нормативних значень чи базових рівнів.

Візуалізація та звітність: Представлення обробленої інформації у зрозумілому вигляді (графіки, карти, дашборди) для операторів та осіб, що приймають рішення [34, 45].

Генерація сповіщень: Автоматичне інформування відповідальних осіб про виявлення критичних ситуацій чи перевищення порогових значень.

Підтримка прийняття рішень: Надання аналітичних інструментів для оцінки ризиків, прогнозування розвитку ситуації та планування заходів реагування.

Для кращого розуміння функціональної структури подібних систем, на рисунку 1.2 наведено приклад типової архітектури роботи IoT системи моніторингу стану води, що ілюструє основні етапи проходження інформації від сенсора до кінцевого користувача.

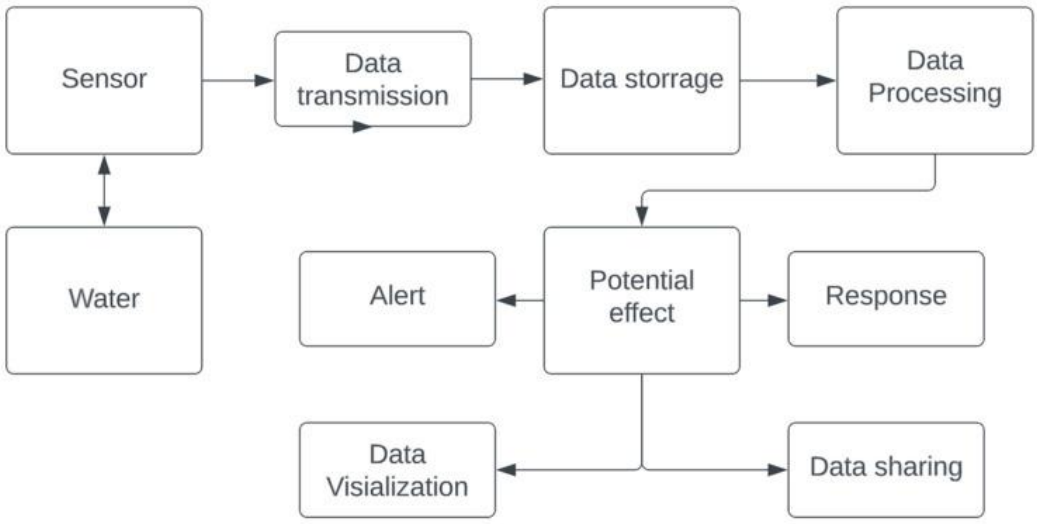


Рисунок 1.2 − Приклад роботи IoT системи моніторингу стану води

Таким чином, основна проблема, яку вирішує розробка програмної частини КФС моніторингу води, полягає у створенні інтелектуального ядра системи, здатного ефективно керувати потоками даних, видобувати з них цінну інформацію та забезпечувати своєчасне реагування на зміни стану водного

середовища. Без надійного та функціонального програмного забезпечення потенціал апаратної частини КФС (сенсорів та мереж) не може бути реалізований повною мірою.

1.2 Порівняльний аналіз: Можливості, підходи та виклики програмного забезпечення КФС моніторингу води

Перехід від традиційних, епізодичних методів моніторингу води до використання кіберфізичних систем знаменує собою значний крок уперед в розумінні та управлінні водними ресурсами. Ця трансформація зумовлена не лише впровадженням нових сенсорних технологій, але й, що має вирішальне значення, розробкою та застосуванням складних програмних рішень. Саме програмне забезпечення виступає інтелектуальним ядром КФС, перетворюючи потоки необроблених даних від датчиків на цінну, дієву інформацію та забезпечуючи основні функції сучасних систем моніторингу.

Програмна частина КФС насамперед забезпечує управління даними в реальному часі [5, 19], що кардинально відрізняє її від лабораторних методів із притаманними їм затримками. Вона відповідає за конфігурування розподілених сенсорних вузлів, синхронізацію вимірювань, застосування ефективних протоколів передачі та гарантовану доставку зібраних показників до централізованого сховища чи аналітичної платформи. Окрім простого збору, невід'ємною функцією є обробка та валідація цих потоків даних. Сирі дані неминуче містять шуми, викиди та пропуски, тому програмні алгоритми виконують їхнє очищення, фільтрацію, інтерполяцію та перевірку на достовірність, часто шляхом порівняння показників або аналізу часових рядів, забезпечуючи надійність подальшого аналізу.

Найбільша перевага програмно-орієнтованих КФС полягає в можливостях інтелектуального аналізу та виявлення подій. Тут застосовуються як класичні статистичні методи, так і передові алгоритми машинного навчання та штучного інтелекту. Це дозволяє автоматично виявляти аномалії - різкі зміни параметрів, що

сигналізують про можливі забруднення чи несправності; здійснювати прогнозування розвитку ситуації на основі накопичених даних; розраховувати комплексні індекси якості води та розпізнавати характерні патерни природних чи антропогенних процесів. Не менш важливою є функція візуалізації та звітності, яка надає користувачам інтуїтивно зрозумілі інструменти - динамічні графіки, інтерактивні геопросторові карти, інформативні дашборди та автоматизовані звіти - для ефективного аналізу стану водного об'єкта. Нарешті, програмне забезпечення реалізує систему сповіщень та підтримки прийняття рішень, автоматично інформуючи відповідальних осіб про критичні ситуації та, в деяких випадках, пропонуючи рекомендації щодо необхідних дій.

Сучасний ринок та наукові дослідження пропонують різноманітні програмні рішення для КФС моніторингу води [1, 7, 11], що відрізняються своєю архітектурою та функціональним фокусом. Широко розповсюдженим підходом є використання хмарних платформ, куди дані з сенсорів передаються для централізованої обробки та зберігання. Такі платформи забезпечують високу масштабованість, глобальну доступність даних та потужні обчислювальні ресурси для складного аналізу, проте вимагають стабільного інтернет-з'єднання та ретельної уваги до питань кібербезпеки. Альтернативна або доповнююча стратегія передбачає використання Edge Computing, коли частина попередньої обробки даних виконується безпосередньо на пристроях поблизу сенсорів. Це зменшує навантаження на мережу передачі даних, знижує затримки для критичних сповіщень і дозволяє системі частково функціонувати автономно, хоча й накладає обмеження на складність локальної обробки та ускладнює управління розподіленою інфраструктурою. Багато рішень також тісно інтегруються з геоінформаційними системами, що дозволяє не тільки візуалізувати дані на карті, але й проводити глибокий просторовий аналіз, моделюючи поширення забруднень чи оцінюючи вплив оточуючих факторів. Окрім комплексних платформ, існують і спеціалізовані аналітичні інструменти, які зосереджені на вирішенні конкретних

завдань, таких як прогнозування якості води за допомогою моделей машинного навчання або детальний аналіз часових рядів.

Застосування таких програмно-орієнтованих КФС надає значні переваги. Насамперед, це оперативність отримання інформації, що є критично важливим для своєчасного реагування на забруднення. Також досягається просторово-часова повнота даних, що дозволяє відстежувати динаміку процесів у всьому водному об'єкті. Застосування складних алгоритмів забезпечує глибину аналізу, виявляючи приховані закономірності та дозволяючи робити обґрунтовані прогнози. Все це разом сприяє підвищенню ефективності управління водними ресурсами та створює потенціал для значної автоматизації рутинних процесів моніторингу та звітності.

Однак, розробка та впровадження таких систем пов'язані і з певними викликами та недоліками. Складність розробки та інтеграції різнорідних апаратних і програмних компонентів [28, 38] вимагає високої кваліфікації та значних зусиль. Безперервний моніторинг генерує великі обсяги даних, що створює проблеми для їх зберігання, передачі та швидкої обробки. Ефективність усієї системи критично залежить від якості та надійності даних, що надходять від сенсорів, адже невраховані помилки датчиків можуть призвести до хибних висновків аналітичних алгоритмів. Хоча машинне навчання відкриває нові можливості, інтерпретація його результатів не завжди є тривіальною, а для навчання моделей потрібні великі та якісні набори даних. Не можна ігнорувати й питання кібербезпеки [9, 27], оскільки підключені до мережі системи є потенційними цілями для атак. Нарешті, вартість розробки або придбання програмного забезпечення, розгортання інфраструктури та залучення кваліфікованих фахівців може бути суттєвим бар'єром.

Таким чином, програмна частина є невід'ємним та визначальним компонентом сучасних КФС моніторингу води, що надає потужні інструменти для аналізу, прогнозування та управління. Вибір оптимального програмного рішення та архітектури залежить від конкретних завдань моніторингу, масштабу системи, вимог до оперативності та наявних ресурсів, але саме ефективне програмне

забезпечення дозволяє повною мірою реалізувати потенціал технологій безперервного контролю стану водних ресурсів.

1.3 Підходи до вирішення завдання за темою дослідження

Для практичної реалізації завдань моніторингу стану води за допомогою кіберфізичних систем, як випливає з аналізу предметної області та існуючих рішень, застосовується комбінація апаратних компонентів, спеціалізованого програмного забезпечення та визначених методологічних підходів. Хоча апаратна частина (сенсори pH, температури, каламутності, розчиненого кисню тощо; автономні вимірювальні вузли/буї; модулі зв'язку LoRaWAN [18, 19], NB-IoT, GSM; шлюзи) є необхідною основою для збору даних, саме програмне забезпечення визначає функціональність, інтелект та ефективність всієї системи моніторингу [38].

Ключові програмні компоненти та інструменти, що використовуються для побудови КФС моніторингу води, охоплюють весь життєвий цикл даних - від їхнього збору до представлення кінцевому користувачеві:

1) Програмне забезпечення сенсорних вузлів та шлюзів. Зазвичай це вбудоване ПЗ для мікроконтролерів, яке керує опитуванням підключених сенсорів, виконує базову обробку (фільтрацію, усереднення), управляє енергоспоживанням (особливо критично для автономних вузлів) та реалізує комунікаційні протоколи (наприклад, MQTT [2, 4], CoAP) для передачі даних через вибрану мережу (LoRaWAN [18, 32], NB-IoT тощо). Важливою функцією сучасного ПЗ цього рівня є підтримка віддаленого оновлення (Over-the-Air, OTA), що дозволяє оновлювати логіку роботи вузлів без фізичного доступу.

2) Необхідне програмне забезпечення для збору, передачі та управління даними на серверному рівні. Це можуть бути серверні програми або хмарні сервіси, що діють як точки прийому даних від шлюзів (наприклад, MQTT брокери). Вони відповідають за аутентифікацію джерел, розбір пакетів даних, їхню трансформацію

та надійне збереження. Для ефективного зберігання часових рядів показників якості води часто використовуються спеціалізовані бази даних, такі як InfluxDB, TimescaleDB або Prometheus.

3) Ядром системи є програмне забезпечення для обробки та аналізу даних. На цьому рівні реалізуються алгоритми для валідації та очищення даних (виявлення та обробка викидів, пропусків), розрахунку похідних показників (наприклад, індексів якості води - WQI), застосування статистичних методів для виявлення трендів та сезонності, а також все частіше - алгоритмів машинного навчання та штучного інтелекту. Моделі ML/AI використовуються для складніших завдань: автоматичного виявлення аномалій (що можуть сигналізувати про забруднення), прогнозування змін параметрів якості води, класифікації стану водного об'єкта чи розпізнавання певних подій (наприклад, ризику цвітіння водоростей). Для розробки цих аналітичних модулів широко застосовуються мови програмування, такі як Python, з потужними бібліотеками для аналізу даних (Pandas, NumPy), машинного навчання (Scikit-learn, TensorFlow, PyTorch) та роботи з часовими рядами.

4) Важливим є програмне забезпечення для візуалізації та взаємодії з користувачем. Це можуть бути веб-додатки [10], настільні програми або мобільні додатки, що надають користувачам (екологам, операторам, дослідникам) зручні інструменти для перегляду даних. Зазвичай це інтерактивні графіки, таблиці, геопросторові карти (з використанням ГІС-технологій, наприклад, бібліотек Leaflet чи Mapbox, для відображення даних сенсорів на карті місцевості) та інформаційні панелі (дашборди), які агрегують ключові показники. Популярними інструментами для створення дашбордів є Grafana, Tableau або Power BI, а також кастомні рішення на базі веб-фреймворків (React, Angular, Vue). Ці інтерфейси також часто дозволяють генерувати звіти.

Система повинна включати програмне забезпечення для сповіщень та реагування. Це модулі, які постійно моніторять вхідні дані або результати аналізу на предмет перевищення заданих порогових значень (встановлених нормами або

на основі історичних даних) чи виявлення значущих аномалій. При виявленні таких подій система автоматично генерує сповіщення та надсилає їх відповідальним особам через різні канали (SMS, email, месенджери, push-сповіщення).

Окрім конкретних програмних інструментів, ефективність КФС значною мірою залежить від дотримання відпрацьованих методологій та найкращих практик вказаних у таблиці 1.2.

Таблиця 1.2 − Найкращі практики та методологічні аспекти забезпечення ефективності КФС

1.	Стратегія відбору проб та конфігурації	- програмне забезпечення дозволяє гнучко налаштовувати частоту вимірювань для кожного сенсора, адаптуючи її до динаміки вимірюваного параметра та завдань моніторингу
2.	Управління якістю даних	- регулярне калібрування сенсорів є необхідним, а його результати (калібрувальні коефіцієнти [13]) мають бути враховані програмним забезпеченням при обробці даних. У самому ПЗ повинні бути реалізовані надійні алгоритми валідації [20] для ідентифікації та маркування потенційно помилкових даних
3.	Розробка та валідація аналітичних моделей	- якщо використовуються моделі ML/AI, їх потрібно ретельно навчати на репрезентативних історичних даних та валідувати їхню точність і надійність перед впровадженням в експлуатацію.
4.	Встановлення порогів для сповіщень	- визначення адекватних порогових значень для генерації сповіщень є критично важливим для уникнення як пропуску реальних проблем, так і надмірної кількості хибних тривог

Кінець таблиці 1.2

5.	Стандартизація та інтероперабельність	- використання стандартизованих форматів даних та протоколів обміну (наприклад, OGC SensorThings API [29]) полегшує інтеграцію КФС з іншими інформаційними системами (ГІС, системи управління водними ресурсами) та забезпечує сумісність компонентів від різних виробників

Таким чином, побудова ефективної КФС моніторингу води вимагає комплексного підходу, де вибір та розробка відповідного програмного забезпечення, що охоплює всі етапи роботи з даними, є ключовим фактором успіху, доповнюючи належним чином підібрану апаратну базу та продумані методології експлуатації.

1.4 Висновки до першого розділу

У межах першого розділу дипломної роботи проведено комплексний аналіз предметної області, пов'язаної з моніторингом стану водних ресурсів. Розглянуто фундаментальну важливість контролю якості води та виявлено ключові недоліки традиційних, епізодичних методів моніторингу, зокрема їхню часову дискретність, просторову обмеженість, операційні затримки та значні ресурсні витрати. На противагу цьому, обґрунтовано переваги та актуальність застосування кіберфізичних систем для вирішення завдань безперервного та автоматизованого моніторингу стану води.

Детально проаналізовано роль та функції програмної частини в архітектурі таких КФС, яка виступає інтелектуальним ядром, відповідальним за збір, передачу, валідацію, обробку, аналіз, візуалізацію даних та генерацію сповіщень. Розглянуто основні можливості, сучасні підходи та технології, що використовуються при розробці програмного забезпечення для КФС моніторингу води, включаючи хмарні

та граничні обчислення, спеціалізовані бази даних для часових рядів, алгоритми машинного навчання для інтелектуального аналізу, а також інструменти для візуалізації та підтримки прийняття рішень. Також ідентифіковано основні виклики, пов'язані з розробкою та впровадженням таких програмних систем, зокрема складність інтеграції, обробка великих обсягів даних, забезпечення якості даних, питання кібербезпеки та вартісні аспекти.

Окреслено ключові програмні компоненти, що охоплюють весь життєвий цикл даних у КФС, від вбудованого програмного забезпечення сенсорних вузлів до серверних застосунків, аналітичних модулів та інтерфейсів користувача. Наголошено на важливості дотримання найкращих методологічних практик для забезпечення ефективності та надійності функціонування системи.

Проведений аналіз дозволив чітко сформулювати основну проблему, яка полягає у необхідності створення архітектурно виваженої та функціонально насиченої програмної частини для КФС моніторингу стану води, та визначити ключові завдання подальшого дослідження і розробки в рамках даної дипломної роботи. Таким чином, результати першого розділу слугують теоретичним підґрунтям для проектування та реалізації програмних компонентів системи.

2 РОЗРОБКА АРХІТЕКТУРНО-ФУНКЦІОНАЛЬНОЇ МОДЕЛІ ПРОГРАМНОЇ ЧАСТИНИ КІБЕРФІЗИЧНОЇ СИСТЕМИ МОНІТОРИНГУ ВОДИ

2.1 Вибір та опис архітектури програмної системи моніторингу

Проектування архітектури програмного забезпечення є основою у створенні будь-якої складної інформаційної системи, і кіберфізичні системи моніторингу стану води не є винятком. Архітектура ПЗ визначає не лише поточну структуру та функціональність, але й закладає основи для майбутньої масштабованості, супроводжуваності, надійності та безпеки системи. У контексті даної дипломної роботи, яка зосереджена на програмній частині КФС, вибір оптимальної архітектури набуває особливого значення, оскільки саме програмні компоненти відповідають за збір даних від сенсорних елементів, їхню обробку, аналіз, зберігання та представлення користувачеві. Даний підрозділ присвячений обґрунтуванню вибору та детальному опису архітектури програмної частини розроблюваної системи моніторингу стану води, з урахуванням специфічних вимог предметної області та сучасних тенденцій у розробці ПЗ для КФС та систем Інтернету Речей.

2.1.1 Теоретико-методологічні засади архітектурного проектування програмних систем для КФС

Архітектурне проектування програмного забезпечення як науково-практична дисципліна ґрунтується на сукупності фундаментальних принципів, концепцій та випробуваних методологій. Визначення архітектури ПЗ як фундаментальної організації системи, що втілена в її компонентах, їх взаємозв'язках між собою та з оточенням, а також у принципах, що керують її проектуванням та еволюцією, підкреслює динамічний та багатогранний характер цього поняття [38]. Для кіберфізичних систем, що характеризуються тісною інтеграцією обчислювальних ресурсів та фізичних процесів, проектування архітектури програмної частини несе

додаткові виклики. Програмне забезпечення таких систем повинно забезпечувати ефективну взаємодію з різнорідним апаратним комплексом, обробляти значні, часто неструктуровані потоки даних, що надходять у реальному часі, та гарантувати високий рівень відмовостійкості й інформаційної безпеки. Сучасні дослідження в галузі архітектур для систем Інтернету Речей та КФС значну увагу приділяють багатошаровим моделям, сервіс-орієнтованим підходам, концепціям граничних та туманних обчислень [32], а також розробці механізмів забезпечення семантичної та технологічної інтероперабельності між компонентами [24, 29].

Процес вибору та обґрунтування архітектурних рішень є ітеративним і зазвичай включає кілька ключових етапів. Початковим кроком є ретельний аналіз та формалізація вимог до системи, що охоплюють як функціональні аспекти (опис того, що система повинна виконувати), так і нефункціональні атрибути якості та наявні обмеження. Важливим інструментом на цьому етапі є застосування сценаріїв використання для виявлення ключових функціональних можливостей та сценаріїв якості для конкретизації та операціоналізації нефункціональних вимог. Наступним кроком є дослідження та порівняльний аналіз відомих архітектурних стилів та патернів проектування. Архітектурний стиль визначає набір дозволених типів компонентів та з'єднувачів, а також сукупність обмежень на їх конфігурування. Кожен стиль має свої іманентні переваги та недоліки щодо забезпечення тих чи інших атрибутів якості. Для оцінки та порівняння альтернативних архітектурних рішень можуть застосовуватися як формалізовані методи, наприклад, метод аналізу компромісів на основі атрибутів якості, так і менш формальні експертні оцінки та аналіз відповідності поставленим завданням. Вибір оптимальної архітектури найчастіше є результатом пошуку компромісу між різними, іноді суперечливими, вимогами та цільовими атрибутами якості.

При розробці архітектури програмної частини для КФС моніторингу стану води особлива увага приділятиметься дотриманню фундаментальних принципів програмної інженерії, таких як принцип модульності, що забезпечує розбиття системи на керовані частини; принцип слабкої зв'язності між компонентами, що

підвищує їх незалежність та полегшує модифікацію; та принцип сильної згуртованості компонентів, що означає, що кожен компонент повинен виконувати чітко визначену та логічно пов'язану множину функцій. У випадку застосування об'єктно-орієнтованого підходу до реалізації окремих модулів, важливим буде також дотримання принципів SOLID, що сприяють створенню більш гнучких, зрозумілих, легких у тестуванні та супроводі програмних систем.

2.1.2 Специфікація архітектурно значущих вимог до програмної частини системи

Архітектурно значущі вимоги становлять ту підмножину загальних вимог до системи, яка має найбільш суттєвий та визначальний вплив на вибір та формування її архітектури. Для програмної частини розроблюваної кіберфізичної системи моніторингу стану води, спираючись на результати аналізу предметної області, викладені у першому розділі, та враховуючи загальні принципи функціонування КФС, можна ідентифікувати низку ключових ASRs, які слугуватимуть основними драйверами при прийнятті архітектурних рішень.

Серед функціональних вимог, що безпосередньо впливають на архітектуру, слід виділити, по-перше, забезпечення збору та агрегації даних від множинних, потенційно географічно розподілених сенсорних вузлів [14, 41]. Програмна частина повинна підтримувати гнучкі інтерфейси для прийому даних, що надходять за різними протоколами, та забезпечувати можливість їх первинної нормалізації та агрегації. Це детермінує дизайн вхідних інтерфейсів системи, архітектуру компонентів, відповідальних за обробку потоків даних, та структуру використовуваних сховищ.

По-друге, критично важливою є обробка та аналіз даних у режимі, близькому до реального часу [5, 42]. Система повинна виконувати операції фільтрації, валідації, інтерполяції пропущених значень, а також базовий аналіз (наприклад, розрахунок середніх значень, виявлення виходів за попередньо визначені порогові значення) з мінімально можливою затримкою. Для реалізації більш складних

аналітичних функцій, таких як прогнозування динаміки параметрів або виявлення складних патернів забруднень на основі алгоритмів машинного навчання, програмна частина має надавати відповідні модулі або гнучкі інтерфейси для їх інтеграції. Це, в свою чергу, вимагає архітектури, що підтримує ефективну обробку потокових даних та можливість інтеграції зі спеціалізованими аналітичними бібліотеками та фреймворками.

По-третє, значні обсяги даних моніторингу, що характеризуються часовою прив'язкою, потребують ефективного та надійного зберігання [7] для забезпечення можливостей історичного аналізу, виявлення довгострокових трендів та формування звітності. Архітектурні рішення повинні передбачати використання сховищ даних, спеціально оптимізованих для роботи з часовими рядами.

Також, функціональні вимоги включають, що кінцеві користувачі системи (екологи, оператори, аналітики) повинні мати доступ до інтуїтивно зрозумілих засобів візуалізації даних (графіків, інтерактивних дашбордів, картографічних представлень результатів моніторингу) та отримувати своєчасні сповіщення про виникнення критичних подій або ситуацій, що потребують негайного реагування. Ця вимога безпосередньо впливає на дизайн користувацьких інтерфейсів та архітектуру системи управління подіями та сповіщеннями.

Не менш важливу роль у формуванні архітектури відіграють нефункціональні вимоги, або атрибути якості. Для їх конкретизації та подальшого аналізу доцільно використовувати підхід на основі сценаріїв якості.

Ключовим атрибутом є продуктивність системи [38]. Наприклад, сценарій затримки обробки може бути сформульований так: при надходженні стандартного пакета даних від сенсорного шлюзу в умовах нормального навантаження, час від моменту отримання цього пакета системою до моменту, коли оброблені дані стають доступними для відображення користувачеві, не повинен перевищувати 3 секунд для 95% таких пакетів. Інший аспект продуктивності - пропускна здатність - може бути оцінений через сценарій, що визначає максимальну кількість пакетів даних на секунду, яку система здатна успішно обробити без втрат в умовах

пікового навантаження. Виконання цих вимог впливає на вибір механізмів обробки потоків, доцільність застосування асинхронних операцій, вибір оптимізованих сховищ даних та можливе використання технологій кешування.

Наступним важливим атрибутом є масштабованість. Сценарій збільшення кількості сенсорів може виглядати так: при додаванні до системи 500 нових сенсорних вузлів, система повинна адаптуватися до збільшеного навантаження, зберігаючи попередньо визначений рівень продуктивності (згідно зі сценарієм затримки обробки) без необхідності внесення змін до коду основних програмних компонентів. Метрикою тут може слугувати відношення приросту використання обчислювальних ресурсів до кількості нових сенсорів та можливість горизонтального масштабування компонентів, відповідальних за обробку даних. Це вимагає розгляду таких архітектурних підходів, як мікросервісна архітектура, застосування балансувальників навантаження та проектування компонентів, що не зберігають стану між запитами.

Надійність та доступність системи моніторингу є критично важливими. Сценарій відмови компонента обробки може передбачати, що при збої одного екземпляра сервісу, відповідального за аналіз даних, система повинна продовжувати збирати та зберігати дані, а функція аналізу має бути або тимчасово недоступною, або автоматично перенаправлена на резервний екземпляр. Середній час до відновлення функціональності аналізу та загальна доступність системи є ключовими метриками. Для забезпечення таких характеристик архітектура повинна включати механізми резервування ключових компонентів, моніторингу їх стану, автоматичного перемикання на резервні потужності та використання черг повідомлень для буферизації даних при тимчасовій недоступності обробників.

Супроводжуваність та модифікованість системи визначають легкість її подальшого розвитку та адаптації до нових вимог. Сценарій додавання нового типу аналізу може оцінювати час, необхідний розробнику для інтеграції нового алгоритму розрахунку комплексного індексу якості води, та кількість інших модулів системи, які будуть зачеплені цією зміною. Висока модульність, слабка

зв'язність компонентів, чітко визначені та стабільні програмні інтерфейси (API), а також використання відомих патернів проектування, що сприяють розширюваності (наприклад, патерни Strategy або Plugin), є ключовими архітектурними рішеннями для задоволення цих вимог.

Також атрибут безпеки є невід'ємною складовою будь-якої сучасної інформаційної системи. Сценарій несанкціонованого доступу до даних передбачає, що при спробі отримати доступ до API системи без належної автентифікації, система повинна відхилити такий запит, зафіксувати спробу в логах безпеки та не допустити витоку даних. Також важливим є сценарій захисту даних при передачі, який вимагає, щоб усі дані, що передаються між компонентами системи та до кінцевого користувача, були зашифровані з використанням стійких криптографічних протоколів. Це вимагає впровадження в архітектуру надійних механізмів автентифікації та авторизації, шифрування даних, захисту API від поширених векторів атак та забезпечення регулярного оновлення безпеки компонентів системи.

2.1.3 Огляд та критичний аналіз архітектурних стилів для програмної частини КФС моніторингу

Вибір адекватного архітектурного стилю є фундаментальним рішенням, яке визначає загальну структуру, поведінку та ключові характеристики програмної системи. Для програмної частини КФС моніторингу стану води, з урахуванням визначених архітектурно значущих вимог, проведемо критичний аналіз декількох поширених архітектурних стилів.

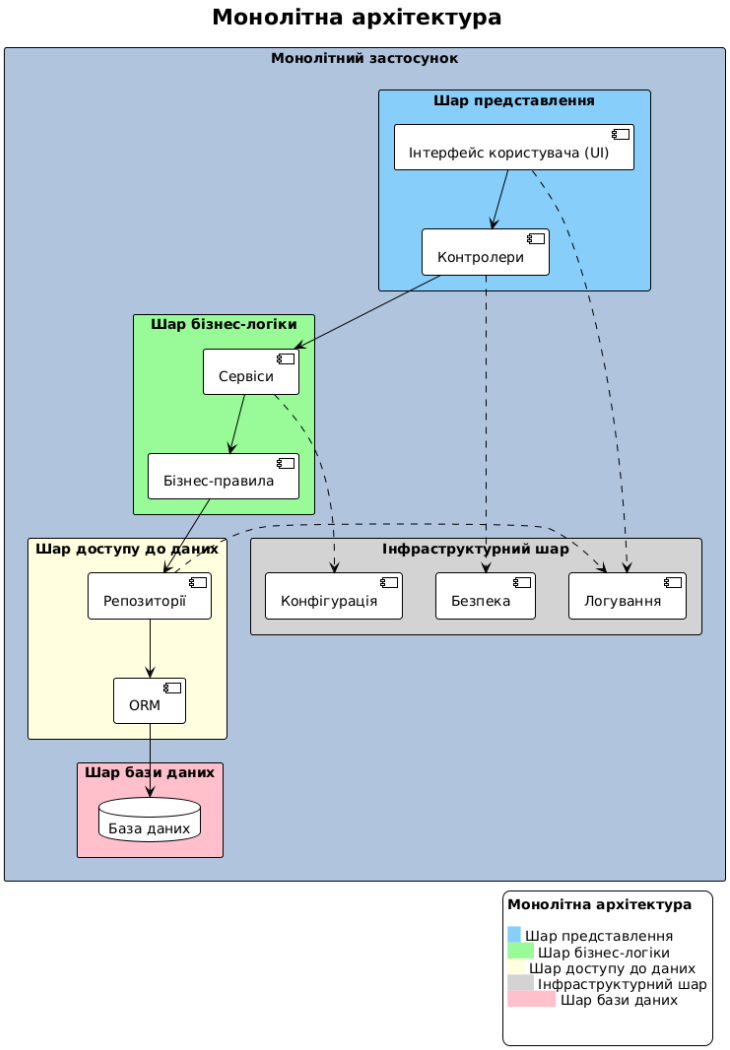


Рисунок 2.1 – Приклад монолітної архітектури

Монолітна архітектура (рисунок 2.1), що представляє собою традиційний підхід, де всі функціональні компоненти програмного забезпечення розробляються та розгортаються як єдиний, щільно інтегрований блок, характеризується простотою розробки та розгортання на початкових етапах. Вона може забезпечити відносно низькі накладні витрати на внутрішньопроцесну комунікацію між компонентами, що позитивно впливає на продуктивність певних операцій, та полегшує комплексне тестування єдиного програмного додатку. Однак, для систем моніторингу, подібних до розроблюваної, монолітна архітектура має суттєві недоліки. Її масштабованість вкрай обмежена, оскільки масштабування вимагає розгортання додаткових екземплярів всього моноліту, навіть якщо вузьким місцем

є лише одна його функціональна частина, що неефективно з точки зору використання обчислювальних ресурсів. Надійність такої системи також є низькою: відмова одного компонента або помилка в коді може призвести до повної відмови всієї системи. З ростом функціональності кодова база монолітної системи стає складною та заплутаною, що значно ускладнює внесення змін, додавання нових можливостей та виправлення помилок; час компіляції та розгортання також суттєво зростає. Крім того, моноліт зазвичай прив'язаний до одного технологічного стеку, що обмежує гнучкість у виборі інструментів та ускладнює впровадження нових технологій для окремих частин системи. Таким чином, для кінцевого рішення в рамках даної дипломної роботи, що передбачає створення розширюваної та надійної системи, монолітна архітектура розглядається як неприйнятна, хоча окремі її модулі можуть бути прототиповані за цим принципом на ранніх стадіях [38].

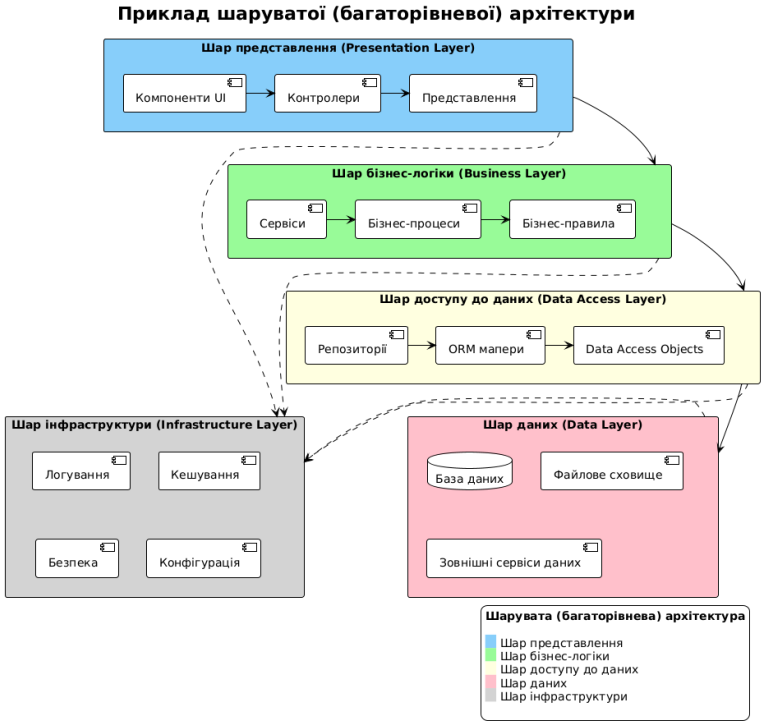


Рисунок 2.2 − Приклад шаруватої (багаторівневої) архітектури

Шарувата (або багаторівнева) архітектура (рисунок 2.2) організує систему у вигляді ієрархії шарів, де кожен шар надає специфічні сервіси вищому шару та

використовує сервіси нижчого шару. Класичним прикладом є розбиття на рівні представлення, бізнес-логіки та доступу до даних. Такий підхід забезпечує чітке розділення відповідальності, що покращує модульність та зрозумілість системи, а також дозволяє незалежну розробку та тестування окремих шарів і підвищує безпеку за рахунок ізоляції рівня даних. Проте, сама по собі шарувата архітектура не вирішує всіх проблем масштабованості та надійності, особливо якщо кожен шар реалізований як великий монолітний компонент. Вона також може призводити до "ефекту каскаду" при внесенні змін, якщо інтерфейси між шарами не є достатньо стабільними та добре визначеними, а для систем з високими вимогами до продуктивності проходження запитів через декілька шарів може створювати небажані додаткові затримки. Принципи шаруватості, безперечно, є корисними для структурування окремих компонентів або сервісів майбутньої системи, зокрема для користувацького інтерфейсу та серверних компонентів, що реалізують бізнес-логіку. Однак, як загальна архітектура для всієї КФС, вона, ймовірно, потребуватиме доповнення іншими архітектурними підходами для задоволення всіх нефункціональних вимог [38].

Рисунок 2.3 – Приклад мікросервісної архітектури

Мікросервісна архітектура (рисунок 2.3) є підходом до розробки програмного забезпечення, при якому великий та складний додаток будується як набір невеликих, автономних, незалежно розгортуваних сервісів [38]. Кожен такий сервіс реалізує певну, чітко визначену бізнес-функцію та зазвичай взаємодіє з іншими сервісами через легко-вагові, стандартизовані протоколи, такі як HTTP/REST API або асинхронні черги повідомлень. Важливою характеристикою є те, що кожен сервіс може мати власну, оптимізовану для його потреб, базу даних (принцип "database per service"). У контексті КФС моніторингу цей підхід пропонує значні переваги.

По-перше, він дозволяє незалежно масштабувати окремі сервіси залежно від поточного навантаження, що є критично важливим для функцій з різним профілем використання.

По-друге, мікросервісна архітектура сприяє підвищенню надійності та відмовостійкості системи, оскільки збій одного сервісу не обов'язково призводить до відмови всієї системи; інші сервіси можуть продовжувати функціонувати, а для ізоляції відмов можуть бути реалізовані відповідні патерни, такі як Circuit Breaker.

По-третє, кожен мікросервіс може бути розроблений з використанням найбільш підходящого для його конкретних завдань технологічного стеку, що забезпечує високу технологічну гнучкість. Невеликі, чітко сфокусовані сервіси легше розуміти, модифікувати, тестувати та супроводжувати, а незалежне розгортання сервісів дозволяє швидше впроваджувати оновлення та нові функціональні можливості. Разом з тим, мікросервісна архітектура не позбавлена недоліків та викликів. Управління великою кількістю розподілених сервісів, налагодження їх взаємодії, забезпечення узгодженості даних, моніторинг стану всієї системи є значно складнішими завданнями, ніж у випадку монолітної архітектури. Комунікація між сервісами через мережу вносить додаткові затримки та може стати "вузьким місцем", а операційна складність розгортання, моніторингу та оркестрації мікросервісів вимагає використання спеціалізованих інструментів та відповідної експертизи. Для програмної частини системи моніторингу води

мікросервісний підхід є дуже привабливим варіантом для реалізації багатьох компонентів, особливо тих, що відповідають за інтенсивну обробку даних, аналітику та функції, які потребують незалежного масштабування.

Рисунок 2.4 – Приклад подієво-орієнтованої архітектури

Подієво-орієнтована архітектура (EDA) (рисунок 2.4) представляє собою архітектурний стиль, в якому компоненти системи взаємодіють переважно шляхом генерації, передачі, виявлення та обробки подій. Події тут розглядаються як сповіщення про значущі зміни стану системи або виникнення певних умов. Цей підхід часто використовується у поєднанні з брокерами повідомлень, які забезпечують надійну та масштабовану доставку подій. Для систем моніторингу,

подібних до розроблюваної, EDA пропонує такі переваги, як асинхронність та слабка зв'язність компонентів, що дозволяє системі бути більш гнучкою та стійкою до змін. Вона також добре підходить для систем, які повинні швидко реагувати на зміни в навколишньому середовищі, такі як отримання нових даних від сенсорів або виявлення аномальних ситуацій. Крім того, EDA сприяє масштабованості обробників подій, оскільки можна легко додавати нових споживачів для паралельної обробки або реалізації нових функціональних можливостей. Серед недоліків та викликів слід зазначити потенційну складність відстеження потоків виконання у системах з великою кількістю типів подій та їх обробників, а також нетривіальність завдань управління станом у розподіленій, подієво-орієнтованій системі. Забезпечення гарантій доставки та суворого порядку обробки подій також потребує ретельного налаштування брокера повідомлень та логіки самих обробників. Для програмної частини КФС моніторингу води подієво-орієнтований підхід є дуже доречним для організації асинхронної обробки даних, що надходять від сенсорів, для реалізації ефективної системи сповіщень, а також для забезпечення гнучкої взаємодії між мікросервісами, якщо такий підхід буде обрано для декомпозиції системи.

2.1.4 Обґрунтування вибору цільової архітектури для програмної частини

Після всебічного аналізу архітектурно значущих вимог та детального критичного огляду основних архітектурних стилів [38], для розробки програмної частини кіберфізичної системи моніторингу стану води було прийнято рішення про використання гібридної архітектури. Ця архітектура синергетично поєднує в собі переваги та нівелює недоліки кількох підходів, а саме: використовує принципи трирівневої клієнт-серверної моделі для загальної структуризації системи та організації взаємодії з користувачем, інтегрує мікросервісний підхід для реалізації ключових модулів, відповідальних за збір, обробку та аналіз даних, а також впроваджує елементи подієво-орієнтованої архітектури для забезпечення

ефективної асинхронної комунікації та реактивності системи. Такий комбінований, або гібридний, підхід дозволяє досягти необхідного балансу між структурованістю, керованістю, масштабованістю, надійністю та гнучкістю системи, що є критично важливим в рамках реалізації дипломного проекту.

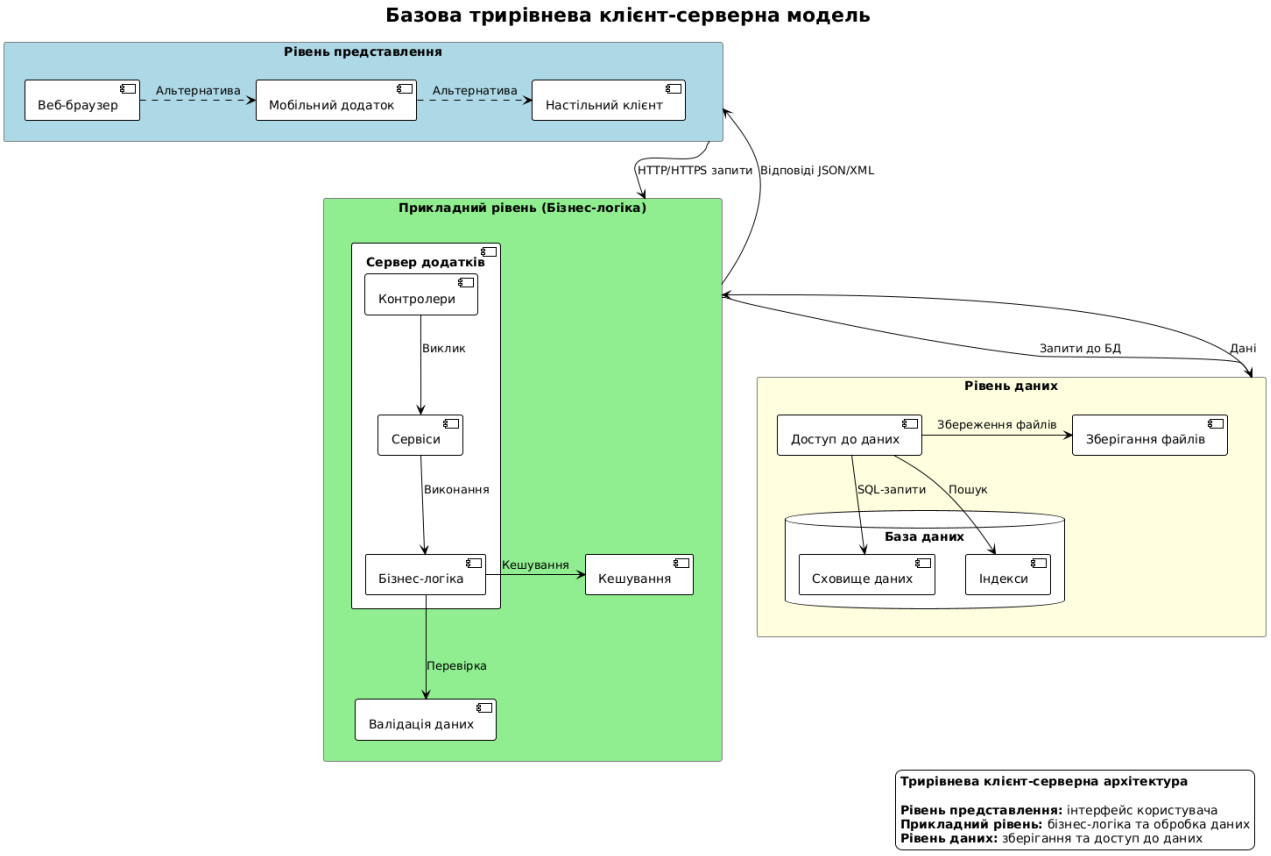


Рисунок 2.5 − Базова трирівнева клієнт-серверна модель

Обрання трирівневої клієнт-серверної моделі (рисунок 2.5) як фундаментального каркасу системи зумовлено її здатністю забезпечити чітке логічне розмежування відповідальності [38] між рівнем представлення (що відповідає за взаємодію з користувачем через веб-інтерфейс), рівнем бізнес-логіки (де зосереджені основні алгоритми обробки даних, управління системою та координації взаємодії), та рівнем даних (що забезпечує надійне зберігання всієї необхідної інформації). Такий підхід сприяє модульній розробці, полегшує процеси тестування та подальшого супроводу системи, а також дозволяє потенційно використовувати різні, найбільш оптимальні технологічні стеки для реалізації

кожного з рівнів. Централізація бізнес-логіки та доступу до даних на серверній стороні також сприяє ефективнішому впровадженню механізмів забезпечення інформаційної безпеки.

Інтеграція мікросервісного підходу є стратегічним рішенням для реалізації ключових функціональних блоків системи [38], що безпосередньо відповідають за інтенсивну обробку даних та виконання аналітичних завдань. Це стосується таких потенційних незалежних сервісів, як "Сервіс прийому даних", "Сервіс валідації та очищення даних", "Сервіс зберігання часових рядів", "Сервіс аналітичної обробки" (який, у свою чергу, може бути декомпозований на більш дрібні спеціалізовані аналітичні сервіси), та "Сервіс управління сповіщеннями". Декомпозиція цих функціональних блоків на окремі, незалежно розгортувані мікросервіси дозволить досягти високого рівня масштабованості, оскільки кожен сервіс можна буде масштабувати окремо відповідно до поточного навантаження, та підвищити загальну відмовостійкість системи, адже збій в одному аналітичному мікросервісі не призведе до зупинки процесу збору даних чи неможливості доступу до користувацького інтерфейсу. Крім того, такий підхід забезпечує технологічну гнучкість, дозволяючи використовувати найбільш оптимальні мови програмування, фреймворки та бази даних для кожного конкретного сервісу, та спрощує процеси оновлення та розвитку окремих функціональних можливостей системи.

Застосування елементів подієво-орієнтованої архітектури, зокрема з використанням брокера повідомлень, є доцільним для організації ефективної асинхронної взаємодії [38] між розробленими мікросервісами, а також для реалізації гнучкої та надійної системи сповіщень. Наприклад, після надходження та первинної обробки нових даних "Сервіс прийому даних" може генерувати відповідну подію (повідомлення), яка буде опублікована у брокері. На цю подію можуть бути підписані інші сервіси, такі як "Сервіс валідації та очищення даних", "Сервіс зберігання часових рядів" та "Сервіс аналітичної обробки", які асинхронно отримають дані для подальшої роботи. Це дозволяє уникнути жорстких

залежностей між сервісами, зменшити час відгуку системи та підвищити її загальну пропускну здатність. Аналогічним чином, при виявленні критичної ситуації або аномальних значень параметрів води відповідний аналітичний сервіс може опублікувати подію, на яку оперативно відреагує "Сервіс управління сповіщеннями", ініціюючи надсилання повідомлень зацікавленим користувачам через налаштовані канали зв'язку.

Порівнюючи обраний гібридний підхід з альтернативними варіантами, слід зазначити, що чисто монолітна архітектура була відхилена через її невідповідність ключовим вимогам щодо масштабованості, надійності та супроводжуваності для системи моніторингу, яка має потенціал до зростання та розвитку. З іншого боку, повний перехід на мікросервісну архітектуру для всіх без винятку аспектів програмного додатку, включаючи користувацький інтерфейс та базову координаційну логіку, був визнаний надто складним для реалізації в рамках дипломного проекту з обмеженими часовими та ресурсними рамками, особливо враховуючи операційну складність та необхідність управління розподіленими транзакціями для всіх аспектів системи. Обраний гібридний підхід є усвідомленим компромісом: він зберігає переваги структурованості та керованості, притаманні трирівневій моделі, для загального каркасу системи, водночас використовуючи потужні можливості мікросервісної архітектури та подієво-орієнтованого підходу для реалізації критично важливих, обчислювально інтенсивних або асинхронних завдань. Основним компромісом такого рішення є деяке збільшення архітектурної складності порівняно з чисто монолітним підходом, що зумовлено необхідністю управління взаємодією між різними сервісами та підтримки інфраструктури брокера повідомлень. Однак, очікувані переваги в частині масштабованості, надійності, гнучкості та довгострокової супроводжуваності системи значно переважають ці потенційні недоліки.

Таким чином, загальна архітектура взаємодії програмної частини із зовнішніми апаратною та серверною частинами, а також внутрішня високорівнева структура клієнтського додатку представлено в діаграмі (рисунок 2.6).

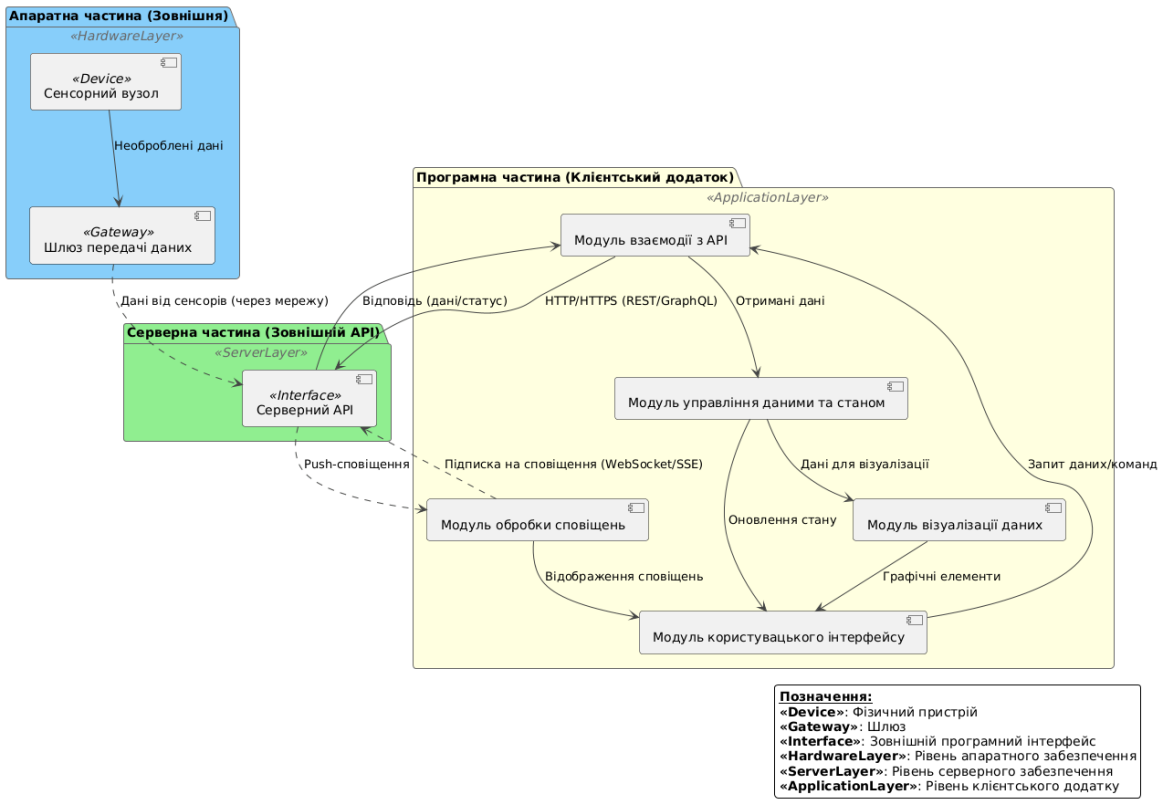


Рисунок 2.6 − Архітектура програмної частини та її взаємодія із зовнішніми системами

2.2 Проектування програмних модулів клієнтської частини системи моніторингу

На основі обраної загальної архітектури взаємодії (визначеної в розділі 2.1), програмна частина, що розробляється в рамках даної дипломної роботи, буде реалізована як клієнтський додаток. Цей додаток призначений для надання користувачам (екологам, операторам, аналітикам) зручних інструментів для доступу, візуалізації та аналізу даних моніторингу стану води, що надходять від зовнішньої серверної інфраструктури через визначений програмний інтерфейс. Даний розділ присвячений детальному проектуванню ключових програмних модулів, що утворюють цю клієнтську частину.

Структура клієнтського додатку буде модульною для забезпечення гнучкості, легкості розробки та подальшого супроводу [38]. Високорівнева

структура програмних модулів клієнтської частини представлена в діаграмі (рисунок 2.7).

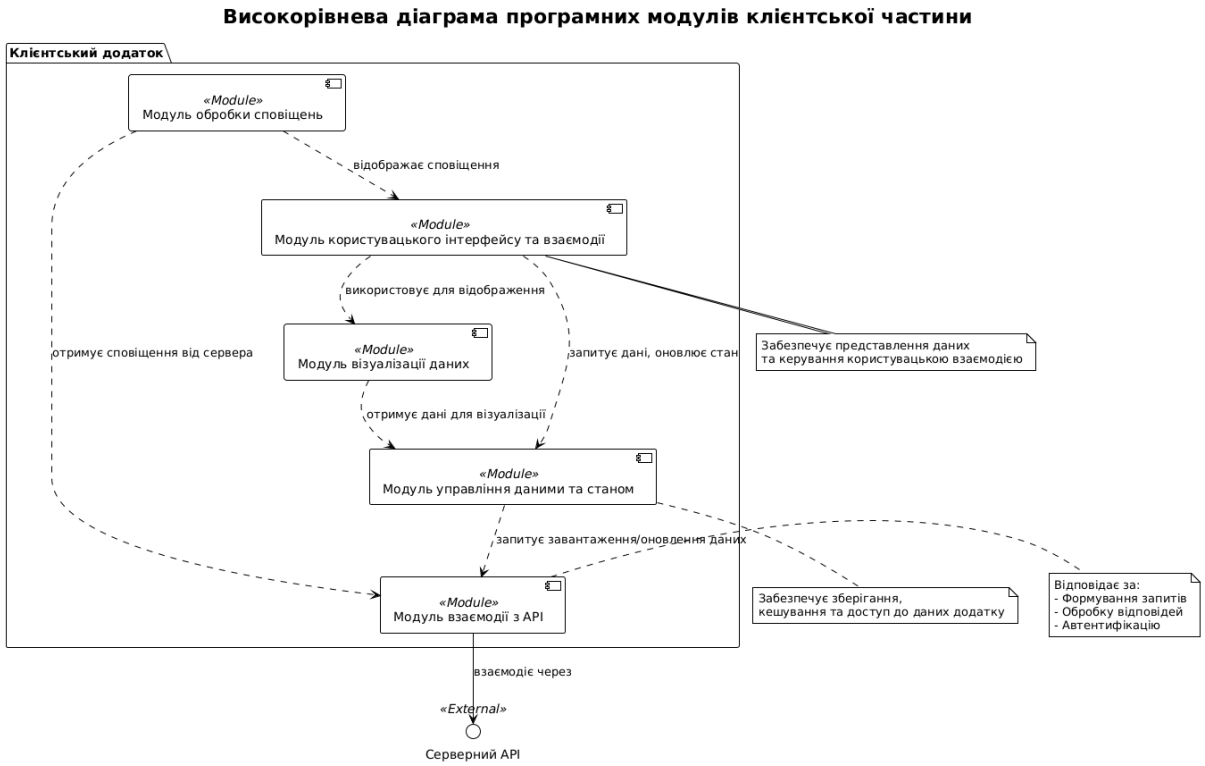


Рисунок 2.7 − Високорівнева діаграма програмних модулів клієнтської частини

Модуль "Взаємодія з API" є фундаментальним компонентом, що відповідає за всю комунікацію клієнтського додатку із зовнішньою серверною частиною системи. Основним призначенням цього модуля є інкапсуляція логіки надсилання запитів до серверного API та обробки отриманих відповідей. Його функції включають формування HTTP/HTTPS запитів згідно зі специфікацією серверного API, обробку кодів стану HTTP, парсинг відповідей від сервера (зазвичай у форматі JSON або XML), а також обробку можливих помилок, пов'язаних з мережевою взаємодією або недоступністю сервера. Важливою функцією є управління автентифікаційними даними (наприклад, токенами доступу, такими як JWT), їх безпечне зберігання на клієнті та автоматичне додавання до запитів. Для реалізації цього модуля планується використання стандартних бібліотек мови програмування, обраної для розробки клієнтської частини (наприклад, fetch API

або бібліотек типу axios для JavaScript-додатків, HttpClient для .NET, Retrofit/OkHttp для Android). Модуль надаватиме чітко визначені програмні інтерфейси для інших модулів клієнтського додатку, абстрагуючи їх від деталей мережевої взаємодії.

Модуль "Управління даними та станом" відіграє ключову роль у зберіганні, кешуванні та управлінні даними, отриманими від сервера, а також у підтримці загального стану клієнтського додатку. Його призначення - забезпечити ефективний доступ до даних для інших модулів, мінімізувати кількість запитів до сервера шляхом кешування часто використовуваної інформації, а також синхронізувати стан різних частин користувацького інтерфейсу. Функції цього модуля включають зберігання даних моніторингу (часових рядів, результатів аналізів), даних конфігурації (наприклад, налаштувань користувача, параметрів відображення), а також стану користувацького сеансу. Для реалізації механізмів кешування можуть використовуватися як прості стратегії (наприклад, зберігання даних у пам'яті з певним часом життя), так і більш складні, з можливістю персистентного зберігання на клієнтському пристрої (наприклад, IndexedDB або localStorage для веб-додатків). Якщо клієнтський додаток є складним, для управління станом можуть бути застосовані спеціалізовані бібліотеки або патерни (наприклад, Redux, Vuex, Zustand для JavaScript-додатків, або патерн MVVM/MVC [38] з відповідними механізмами спостереження за змінами даних). Цей модуль буде тісно взаємодіяти з "Модулем взаємодії з API" для отримання та оновлення даних, а також надаватиме дані "Модулю візуалізації" та іншим компонентам інтерфейсу.

Модуль "Візуалізація даних" є центральним для представлення інформації користувачеві у наочному та зрозумілому вигляді. Його основним завданням є перетворення необроблених числових даних моніторингу на інтерактивні графіки, діаграми, таблиці та, потенційно, представлення на карті. Функції модуля включають вибір типу візуалізації залежно від характеру даних та потреб користувача, налаштування параметрів відображення (масштаб, часові проміжки,

типи ліній, кольори), забезпечення інтерактивності (наприклад, можливість масштабування графіків, перегляду детальної інформації при наведенні курсору), а також експорт візуалізованих даних у різні формати (наприклад, зображення або CSV). Для реалізації цього модуля планується використання спеціалізованих бібліотек для візуалізації даних, таких як Chart.js, Apache ECharts, D3.js для веб-додатків, або відповідних бібліотек для нативних платформ (наприклад, MPAndroidChart для Android, Charts для iOS, OxyPlot для .NET). Модуль буде отримувати дані для візуалізації від "Модуля управління даними та станом" та тісно взаємодіяти з "Модулем користувацького інтерфейсу" для відображення графічних елементів.

Модуль "Користувацький інтерфейс та взаємодія" об'єднує всі візуальні елементи та механізми взаємодії, через які користувач працює з системою. Його призначення - надати інтуїтивно зрозумілий та зручний спосіб навігації по функціях додатку, введення даних (наприклад, параметрів фільтрації, налаштувань), вибору об'єктів моніторингу та перегляду інформації. Цей модуль відповідає за загальну структуру вікон, панелей, меню, кнопок та інших елементів керування. Він також обробляє дії користувача (кліки мишею, введення з клавіатури, жести на сенсорних екранах) та ініціює відповідні операції в інших модулях системи. Для реалізації цього модуля будуть використовуватися стандартні компоненти користувацького інтерфейсу, що надаються обраною платформою розробки (наприклад, HTML/CSS та компоненти фреймворку React/Vue/Angular для веб-додатків; XML-макети та компоненти Android SDK для Android-додатків; елементи керування WPF/WinForms/MAUI для десктопних додатків на .NET [21]). Дизайн інтерфейсу буде орієнтований на забезпечення хорошого користувацького досвіду та зручності використання.

Модуль "Обробка сповіщень" має відповідати за прийом та відображення сповіщень про важливі події, що сталися в системі моніторингу (наприклад, перевищення порогових значень, виявлення аномалій, системні повідомлення). Функції включають підписку на отримання сповіщень від сервера (наприклад,

через WebSockets, Server-Sent Events або механізми push-сповіщень для мобільних платформ), відображення сповіщень користувачеві у відповідному форматі (наприклад, спливаючі повідомлення, значки на іконках, записи в окремому розділі сповіщень), а також, можливо, забезпечення можливості переходу до деталей події, що спричинила сповіщення. Технології реалізації залежатимуть від обраної платформи та механізмів доставки сповіщень, що підтримуються серверною частиною.

2.3 Обґрунтування вибору стеку технологій для реалізації програмної частини

Вибір адекватного стеку технологій є критично важливим етапом у процесі розробки програмного забезпечення, оскільки він безпосередньо впливає на продуктивність розробників, якість кінцевого продукту, його супроводжуваність та можливості подальшого розвитку. Для програмної частини кіберфізичної системи моніторингу стану води, яка в рамках даного дослідження реалізується у вигляді мобільного додатку, вибір технологій ґрунтувався на низці критеріїв, що випливають із визначених функціональних та нефункціональних вимог, а також обраної архітектури програмних модулів, детально описаної раніше. Ключовими критеріями при виборі технологічного стеку виступали відповідність функціональним вимогам, підтримка нефункціональних атрибутів якості, зрілість технологій та активність їхніх спільнот, швидкість та ефективність розробки, а також сумісність та інтеграційні можливості обраних інструментів.

З огляду на необхідність створення кросплатформного мобільного додатку, здатного функціонувати як на платформі Android, так і на iOS, та з метою оптимізації процесу розробки шляхом використання єдиної кодової бази, було прийнято рішення зупинити вибір на фреймворку React Native [25]. React Native, розроблений Facebook, дозволяє створювати нативні мобільні додатки за допомогою JavaScript та бібліотеки React. Такий підхід надає значні переваги,

зокрема скорочення часу розробки порівняно з окремою розробкою для кожної платформи, можливість повторного використання значної частини коду та залучення розробників, знайомих з екосистемою React. Компонентна архітектура React, що лежить в основі React Native, добре узгоджується з модульним проектуванням клієнтської частини та сприяє створенню гнучких та легко супроводжуваних інтерфейсів користувача. Можливість доступу до нативних функцій пристрою через спеціальні модулі, забезпечує необхідний рівень продуктивності та інтеграції з операційною системою.

Основною мовою програмування для розробки мобільного додатку на React Native, відповідно, буде JavaScript. Для підвищення надійності коду, покращення досвіду розробки та забезпечення кращої масштабованості проекту розглядається також використання TypeScript, який додає до JavaScript статичну типізацію. Це дозволяє виявляти багато потенційних помилок ще на етапі розробки та значно спрощує рефакторинг та супровід кодової бази.

Для управління станом мобільного додатку, враховуючи потенційну складність потоків даних моніторингу та стану користувацького інтерфейсу, доцільно використовувати спеціалізовані бібліотеки. Серед можливих варіантів розглядалися Redux, Zustand та MobX. З огляду на потребу в передбачуваному управлінні станом, потужних інструментах для налагодження та ефективній роботі з асинхронними операціями (наприклад, запитами до серверного API), перевага надається бібліотеці Redux у поєднанні з Redux Toolkit, що значно спрощує конфігурацію та зменшує кількість шаблонного коду, пов'язаного з Redux. Це забезпечить структурований підхід до управління даними в додатку.

Взаємодія мобільного додатку з серверною частиною, яка надає дані моніторингу та, можливо, обробляє команди від клієнта, буде здійснюватися через HTTP/HTTPS API. Для виконання мережевих запитів у React Native додатку планується використання бібліотеки Axios. Axios відома своїм зручним API, підтримкою промісів, можливістю перехоплення запитів та відповідей, а також

ефективною обробкою помилок, що робить її оптимальним вибором для інтеграції з зовнішніми сервісами.

Візуалізація даних моніторингу є ключовою функцією клієнтського додатку. Для відображення часових рядів у вигляді графіків та діаграм в середовищі React Native будуть використовуватися спеціалізовані бібліотеки, такі як react-native-charts-wrapper або Victory Native. Вибір конкретної бібліотеки буде залежати від її можливостей щодо кастомізації, продуктивності при роботі з великими наборами даних та простоти інтеграції.

У контексті загальної екосистеми системи моніторингу, зокрема для швидкого прототипування потоків обробки даних на серверній стороні або для реалізації певних інтеграційних завдань, може бути використаний інструмент Node-RED [4]. Node-RED є візуальним інструментом для програмування потоків, побудованим на Node.js, який дозволяє легко з'єднувати апаратні пристрої, API [4] та онлайн-сервіси. Хоча безпосередня розробка мобільного додатку на React Native не передбачає прямого використання Node-RED у клієнтському коді, серверна частина, з якою взаємодіятиме мобільний додаток, може частково або повністю використовувати Node-RED для збору даних з сенсорів, їх попередньої обробки та надання даних через API. Це дозволяє швидко створювати та модифікувати логіку обробки даних без необхідності написання великої кількості низькорівневого коду на сервері, що може бути корисним на етапах прототипування та розгортання системи. Мобільний додаток, у свою чергу, буде споживати дані, підготовлені та надані через API, реалізований за допомогою Node-RED або інших серверних технологій.

Для забезпечення ефективного процесу розробки мобільного додатку на React Native буде використано низку стандартних інструментів. Git слугуватиме системою контролю версій. Управління залежностями JavaScript-проекту буде здійснюватися за допомогою npm або Yarn. Середовищем розробки виступатиме Visual Studio Code завдяки його відмінній підтримці JavaScript/TypeScript та React Native. Для тестування компонентів та логіки планується використання

фреймворку Jest у поєднанні з React Native Testing Library. Статичний аналіз коду та його форматування будуть забезпечені інструментами ESLint та Prettier відповідно.

Таким чином, обраний технологічний стек (рисунок 2.8), що базується на React Native для розробки кросплатформного мобільного додатку, JavaScript/TypeScript як основних мовах програмування, Redux для управління станом, Axios для мережевої взаємодії, спеціалізованих бібліотеках для візуалізації даних, та з урахуванням можливої ролі Node-RED в серверній інфраструктурі, забезпечує потужну та гнучку основу для реалізації програмної частини системи моніторингу стану води. Ці технології є сучасними, широко розповсюдженими та мають значну підтримку з боку спільноти, що сприятиме ефективності процесу розробки та якості кінцевого продукту.

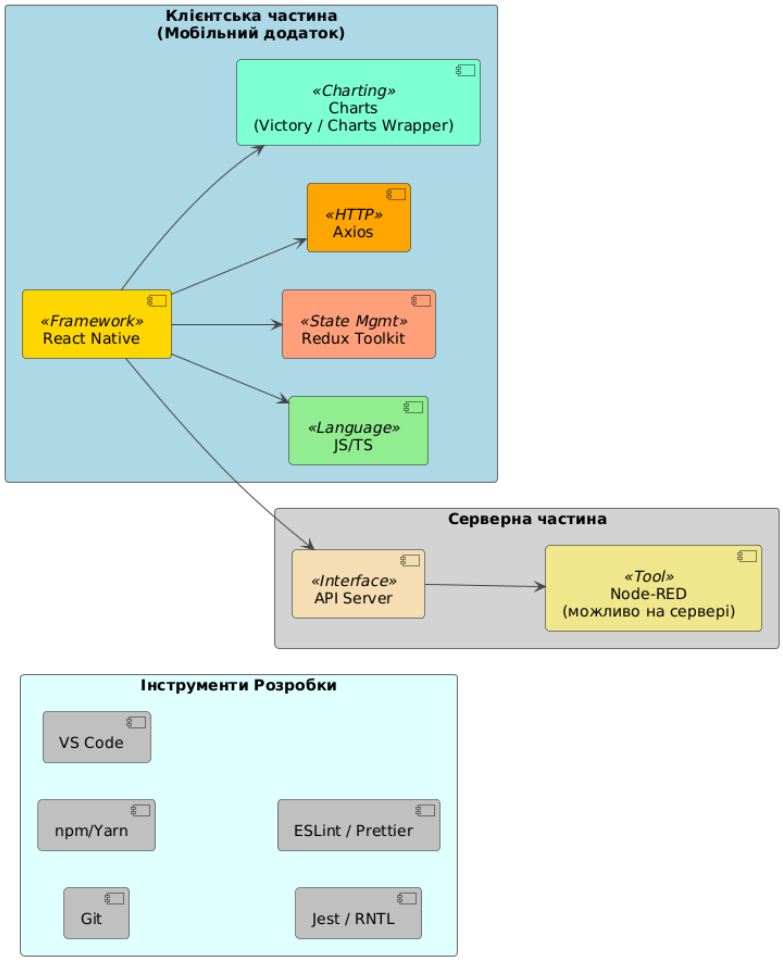


Рисунок 2.8 – Діаграма компонентів стеку технологій

2.4 Висновки до другого розділу

У межах другого розділу даної кваліфікаційної роботи було проведено комплексне проектування програмної частини кіберфізичної системи моніторингу стану води, що реалізується у вигляді мобільного додатку. Основну увагу було приділено обґрунтуванню архітектурних рішень, детальному проектуванню ключових програмних модулів клієнтської частини та вибору оптимального стеку технологій для їхньої реалізації.

На основі аналізу функціональних та нефункціональних вимог, сформульованих у першому розділі, було обрано та описано загальну архітектуру взаємодії програмної частини. Було визначено, що розроблюваний мобільний додаток функціонуватиме як клієнтська частина, що взаємодіє із зовнішньою серверною інфраструктурою через стандартизований програмний інтерфейс (API) для отримання даних моніторингу та, за потреби, надсилання команд чи конфігураційних параметрів. Такий підхід дозволяє чітко розмежувати відповідальність між клієнтською та серверною логікою, забезпечуючи гнучкість та можливість незалежного розвитку компонентів системи.

Далі було здійснено детальне проектування програмних модулів, що утворюють структуру клієнтського мобільного додатку. Визначено ключові модулі, такі як модуль взаємодії з API, відповідальний за всю мережеву комунікацію; модуль управління даними та станом, що забезпечує кешування, зберігання та синхронізацію даних на клієнті; модуль візуалізації даних, призначений для перетворення числових даних на інтерактивні графіки та діаграми; модуль користувацького інтерфейсу та взаємодії, що об'єднує візуальні елементи та механізми управління додатком; а також модуль обробки сповіщень для інформування користувача про важливі події. Для кожного модуля було визначено його основне призначення, ключові функції та принципи взаємодії з іншими модулями, що забезпечує модульність та слабку зв'язність розроблюваного програмного забезпечення.

Завершальним етапом проектної частини стало обґрунтування вибору технологічного стеку для реалізації програмної частини. З огляду на необхідність створення кросплатформного мобільного додатку, було обрано фреймворк React Native, що дозволяє розробляти нативні додатки для платформ Android та iOS з використанням JavaScript та React. Для управління станом додатку було обрано бібліотеку Redux у поєднанні з Redux Toolkit. Мережева взаємодія з серверним API буде реалізована за допомогою бібліотеки Axios. Для візуалізації даних передбачається використання спеціалізованих бібліотек, таких як react-native-charts-wrapper або Victory Native для графіків та react-native-maps для картографічного представлення. Також було згадано потенційну роль Node-RED у загальній екосистемі проекту, зокрема для прототипування або реалізації окремих компонентів серверної логіки, з якою взаємодіятиме розроблюваний мобільний додаток. Вибір інструментів для розробки, тестування та управління залежностями також був обґрунтований з точки зору ефективності та якості процесу розробки.

Таким чином, у другому розділі було закладено міцну архітектурно-технологічну основу для подальшої практичної реалізації програмної частини кіберфізичної системи моніторингу стану води у вигляді мобільного додатку. Розроблені проектні рішення спрямовані на створення функціонального, надійного, масштабованого та легкого у супроводі програмного продукту, здатного задовольнити потреби користувачів у доступі та аналізі даних про стан водних ресурсів. Результати, отримані в цьому розділі, слугуватимуть безпосереднім керівництвом на етапі розробки та тестування програмного забезпечення, який буде детально описаний у наступних розділах даної роботи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОГО ДОДАТКУ СИСТЕМИ МОНІТОРИНГУ СТАНУ ВОДИ ТА ОЦІНКА ЙОГО ФУНКЦІОНАЛЬНОСТІ

3.1 Опис середовища розробки та інструментальних засобів реалізації

Практична реалізація програмної частини кіберфізичної системи моніторингу стану води, втіленої у кросплатформеному додатку, здійснювалася з використанням комплексу сучасних технологій та інструментальних засобів. Вибір цих засобів був зумовлений необхідністю створення рішення, що функціонує на різних платформах та відповідає вимогам функціональності, продуктивності й зручності подальшої підтримки, як було обґрунтовано у попередніх розділах.

Основою для розробки слугував фреймворк React Native, який дозволяє створювати нативні застосунки для різних платформ, зокрема iOS, Android та веб, на базі єдиної кодової основи. Для спрощення процесу розробки, конфігурації, збірки та розгортання додатку активно використовувалася платформа Expo, що надає широкий набір готових інструментів та модулів. Основною мовою програмування було обрано TypeScript, що забезпечило строгу типізацію коду, сприяючи підвищенню його надійності, ранньому виявленню помилок та покращенню загальної супроводжуваності проєкту. Робочий процес розробки підтримувався сучасними інтегрованими середовищами розробки, такими як Visual Studio Code, що забезпечували ефективну роботу з кодовою базою та інтеграцію з іншими інструментами.

Управління станом компонентів реалізовувалося за допомогою вбудованих механізмів React Hooks, тоді як для глобального стану додатку та забезпечення передачі даних між компонентами використовувався React Context API. Локальне персистентне зберігання даних користувача, таких як налаштування або кешована інформація, здійснювалося за допомогою бібліотеки AsyncStorage. Навігаційна структура додатку була реалізована з використанням Expo Router, що базується на

файловій організації екранів, та доповнена можливостями бібліотеки React Navigation.

Мережева взаємодія з серверною частиною, зокрема для отримання даних від симульованих IoT-датчиків через API-ендпоінти, надані Node-RED, реалізовувалася за допомогою вбудованого Fetch API, що підтримує асинхронні HTTP-запити в рамках REST архітектури. Для побудови користувацького інтерфейсу та забезпечення візуалізації даних було залучено низку спеціалізованих бібліотек: React Native Reanimated та React Native Gesture Handler для створення плавних анімацій та обробки жестів; React Native Chart Kit для відображення історичних даних у вигляді діаграм; React Native SVG для роботи з векторною графікою; @expo/vector-icons для набору іконок та Expo Haptics для тактильного зворотного зв'язку.

Процес збірки та розробки підтримувався інструментом командного рядка Expo CLI та JavaScript-бандлером Metro. Контроль якості коду забезпечувався лінтером ESLint. Для управління версіями коду проєкту використовувалася система контролю версій Git, а управління залежностями та виконання скриптів здійснювалося за допомогою пакетного менеджера npm. Налагодження та тестування додатку проводилося з використанням вбудованих інструментів розробника Expo та React Native, спеціалізованих скриптів для перевірки алгоритму WQI, а також шляхом запуску на емуляторах мобільних платформ та у веб-браузерах, і на реальних пристроях через застосунок Expo Go. Важливою частиною середовища розробки на початкових етапах була серверна інфраструктура на базі Node-RED, яка використовувалася для симуляції даних та надання API, що дозволило розробляти клієнтську частину в умовах, наближених до реальних.

Для наочної демонстрації взаємозв'язків та робочих процесів у вибраному середовищі розробки була створена концептуальна схема (рисунок 3.1). Ця схема візуалізує ключові інструментальні засоби, такі як інтегроване середовище розробки VS Code з його компонентами, систему контролю версій Git, пакетний менеджер npm/Yarn, інструмент командного рядка Expo CLI та бандлер Metro.

Також відображено роль розробника, основні потоки даних та команд між цими інструментами, починаючи від написання та аналізу коду, управління залежностями, і завершуючи процесами збірки, розгортання та виконання застосунку в цільовому середовищі, а також його взаємодію із серверною частиною Node-RED. Така візуалізація дозволяє краще зрозуміти архітектуру та динаміку інструментального комплексу, задіяного при створенні кросплатформеного застосунку моніторингу стану води.

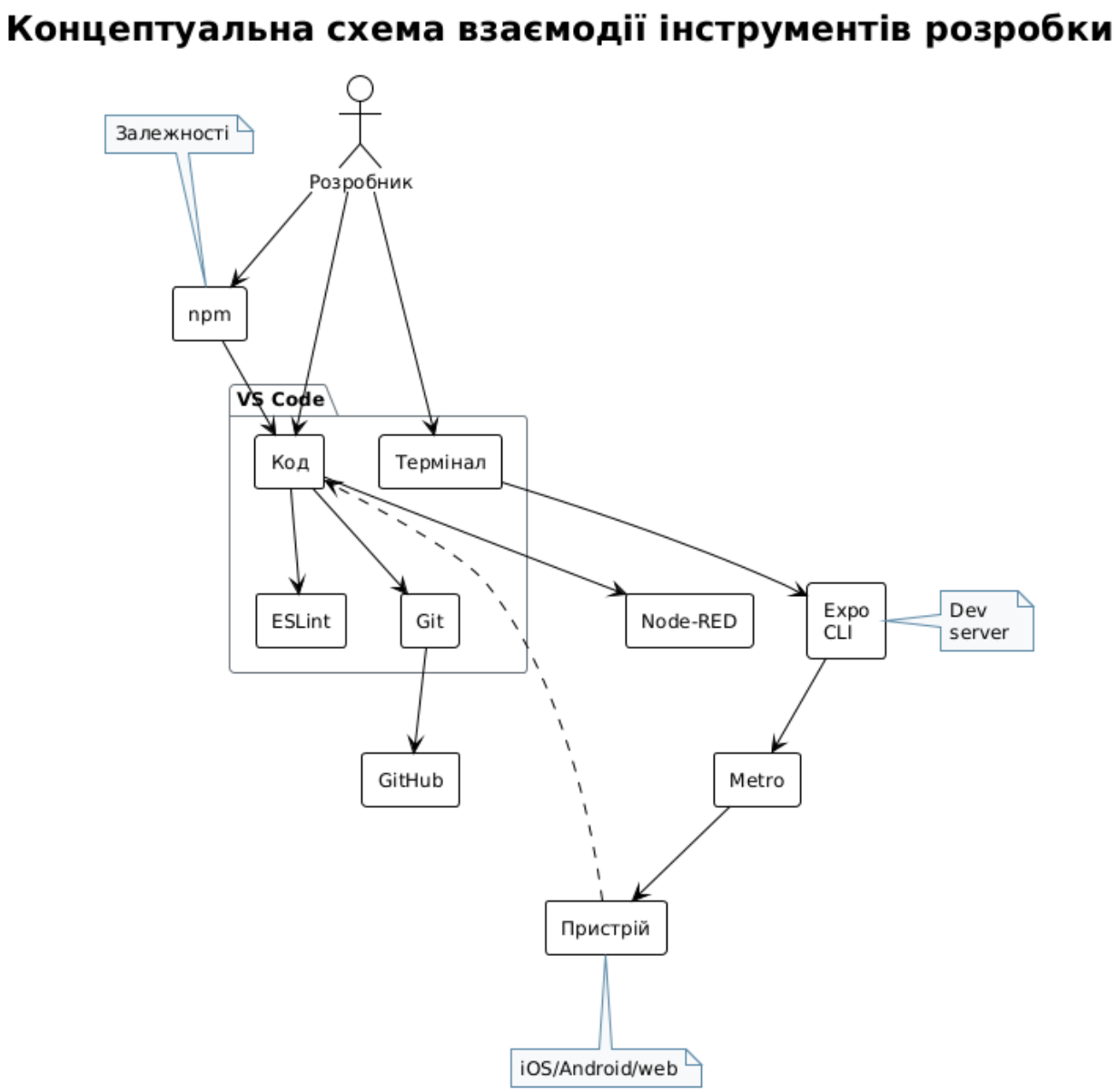


Рисунок 3.1 − Концептуальна схема взаємодії інструментів

Таким чином, обраний комплекс технологій та інструментальних засобів сформував надійне та гнучке середовище, що дозволило ефективно реалізувати всі заплановані функціональні аспекти кросплатформеного додатку.

3.2 Реалізація ключових програмних модулів мобільного додатку

Розробка кросплатформеного додатку передбачала створення низки взаємопов'язаних програмних модулів, кожен з яких відповідає за виконання специфічних завдань у загальній архітектурі системи. Детальна реалізація цих ключових модулів буде розглянута нижче.

3.2.1 Розробка модуля взаємодії з серверним API

Модуль взаємодії з серверним API є фундаментальним компонентом архітектури додатку, що забезпечує всю комунікацію із зовнішньою серверною частиною. Його основне призначення полягає в інкапсуляції логіки надсилання запитів до сервера та обробки отриманих відповідей, таким чином абстрагуючи деталі мережевої взаємодії від інших модулів застосунку. В контексті даного проєкту, первинна розробка та тестування клієнтської частини здійснювалися з використанням серверної інфраструктури, симульованої на платформі Node-RED. Ця платформа надавала необхідні ендпоінти для емуляції роботи IoT-датчиків, отримання даних моніторингу, управління умовними пристроями та перевірки їхнього статусу.

Процес розробки модуля передбачав ретельне проектування механізмів формування та надсилання HTTP-запитів до визначених ендпоінтів серверного API. Для цього використовувався вбудований Fetch API, що забезпечує асинхронне виконання мережевих операцій. Модуль був спроектований для роботи з такими ключовими ендпоінтами, як POST /api/getWQI для отримання поточного індексу якості води та асоційованих параметрів, GET /api/getParameterHistory для запиту часових рядів історичних даних, POST /api/addDevice для реєстрації нового

пристрою моніторингу, GET /api/getDeviceStatus для перевірки статусу підключення пристрою, а також, як видно з конфігурації серверної симуляції, ендпоінти POST /api/calibrateSensors для ініціації калібрування сенсорів та GET /api/listAvailableSensors для отримання списку доступних сенсорів. При формуванні запитів враховувався необхідний HTTP-метод (GET або POST), можливість передачі тіла запиту у форматі JSON (наприклад, при додаванні пристрою чи калібруванні) та параметрів у рядку URL.

Для забезпечення ефективної розробки та всебічного тестування клієнтської частини в умовах, що максимально наближені до реальних, та до моменту повного розгортання продуктивної серверної інфраструктури, була розроблена та налаштована комплексна симуляційна модель серверного API на базі візуального інструменту програмування потоків Node-RED. На рисунку 3.2 представлено фрагмент реалізованих потоків (flows) у середовищі Node-RED, що демонструє архітектуру симуляції серверної логіки. Як видно зі схеми, були створені спеціалізовані потоки для обробки HTTP-запитів, що надходять від клієнтського додатку до згаданих ключових ендпоінтів. Кожен з цих шляхів API був пов'язаний з відповідними вузлами Node-RED (типу http in), які приймали запити, та вузлами-функціями (function), що реалізовували логіку симуляції. Наприклад, потік для /getWQI не просто повертав статичні дані, а міг генерувати значення WQI та пов'язані параметри води (pH, TDS, турбідність, температура) з певною динамікою, імітуючи реальні коливання показників з плином часу. Це досягалося за допомогою вузлів ініціалізації та періодичного оновлення стану симуляції (зокрема, вузли "Ініціалізація/скидання симуляції" та "Інтервал оновлення (5с)"), що дозволяло клієнту отримувати дані, які змінюються, подібно до взаємодії з реальними IoT-датчиками. Аналогічно, потік для /calibrateSensors імітував процес калібрування, повертаючи відповідний результат "Calibration Response", а /listAvailableSensors надавав "Sensor List Response", що містив список умовних датчиків. Верхня частина схеми (рис. 3.2) також ілюструє логіку налаштування розширених станів та конфігурацій пристроїв, включаючи ініціалізацію симуляції для кількох умовних

пристроїв та оновлення їхнього стану, що відстежувалося через вузли "Debug: Ініціалізація" та "Debug: Оновлення стану". Окрім обробки запитів на отримання поточних даних та управління пристроями, симуляційне середовище Node-RED також включало потоки для імітації збереження історичних даних, як показано в нижній частині рисунку 3.2. Реалізовано логіку для "Збереження історії", що імітувала запис даних у сховище, з можливістю автозбереження через задані інтервали. Приклад конфігурації одного з вузлів обробки запитів, зокрема для ендпоінту /api/getWQI, наведено на рисунку 3.3, де вказано HTTP-метод GET та відносний URL, що підтверджує відповідність між клієнтською логікою та серверною симуляцією. Використання вузлів "Debug" на різних етапах усіх потоків забезпечувало можливість моніторингу та налагодження роботи симуляції в реальному часі. Таким чином, створене на Node-RED симуляційне середовище надало розробникам клієнтської частини потужний інструмент для ітеративної розробки, тестування мережевої взаємодії, логіки обробки даних та коректності роботи користувацького інтерфейсу в контрольованих та відтворюваних умовах.

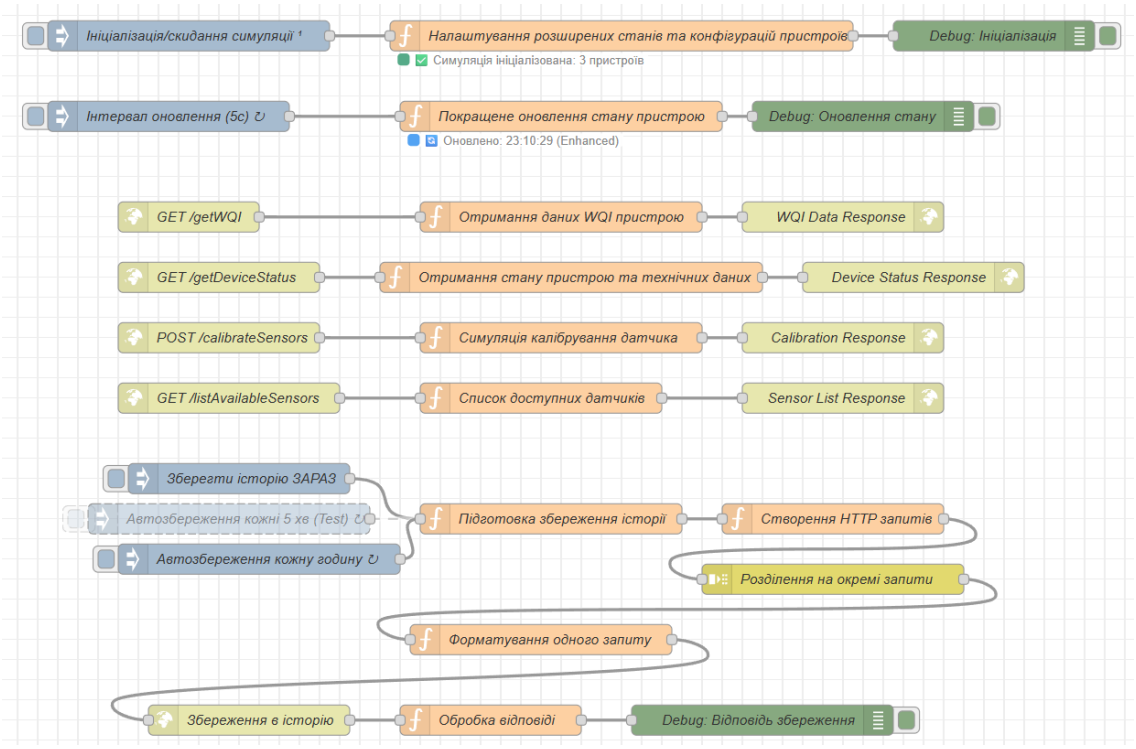


Рисунок 3.2 – Фрагмент потоків Node-RED, що демонструють архітектуру симуляції серверного API

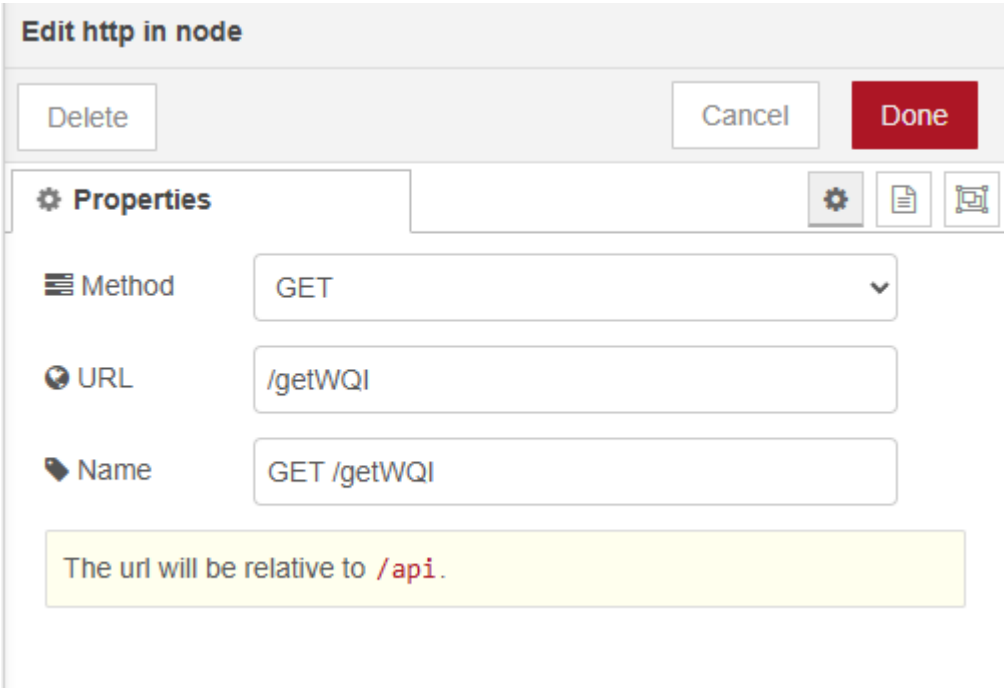


Рисунок 3.3 − Приклад конфігурації вузла для ендпоінту /api/getWQI в Node-RED

Після отримання відповіді від сервера, модуль виконує її комплексну обробку, що включає перевірку HTTP-статус коду на успішність операції (наприклад, 200 OK) та парсинг тіла відповіді, яке, як правило, передається у форматі JSON. Отримані та валідовані дані далі передаються іншим компонентам системи для їх подальшої обробки, зберігання у локальному стані або візуалізації в інтерфейсі користувача. Модуль також інкапсулює управління конфігураційними параметрами з'єднання, зокрема адресою центрального сервера. Передбачено можливість гнучкого налаштування цих параметрів, що є важливим для переходу до продуктивного середовища.

Ключовою особливістю модуля є розгалужена система обробки помилок, спроектована для забезпечення максимальної стабільності та надійності роботи додатку. Ця система здатна ідентифікувати та обробляти різні типи помилок, включаючи мережеві збої (наприклад, втрата з'єднання або недоступність сервера), перевищення часу очікування відповіді (таймаути), а також помилки, що повертаються сервером (наприклад, HTTP-коди 4xx для помилок клієнта або 5xx для серверних помилок). Для підвищення відмовостійкості реалізовано механізми

повторних спроб запиту (retry logic), часто з використанням стратегії експоненційної затримки (exponential backoff), що дозволяє додатку автоматично відновлювати роботу після короткочасних збоїв. У випадку неможливості отримання даних з сервера, модуль може ініціювати перехід до роботи з локально кешованими даними, якщо такі доступні, мінімізуючи негативний вплив на досвід користувача. Важливою частиною є також коректне інформування користувача про виникнення проблем зі зв'язком чи отриманням даних. Хоча на поточному етапі симуляції складні механізми автентифікації та авторизації не були першочерговими, модуль спроектований з урахуванням майбутньої інтеграції таких протоколів для захисту даних та забезпечення безпечного доступу до API, передбачаючи передачу даних через захищене HTTPS з'єднання у продуктивному середовищі.

Таким чином, розроблений модуль взаємодії з серверним API виступає як надійний та добре структурований посередник між клієнтською частиною додатку та зовнішньою серверною інфраструктурою. Він забезпечує ефективну інкапсуляцію всієї мережевої логіки, обробку помилок та управління конфігурацією з'єднання, що значно спрощує розробку інших компонентів системи та сприяє підвищенню загальної модульності, тестованості й супроводжуваності застосунку.

3.2.2 Імплементація модуля управління даними та станом на клієнті

Ефективне управління даними та внутрішнім станом є фундаментальною вимогою для забезпечення належної функціональності та користувацького досвіду в інтерактивному застосунку. Розроблений модуль управління даними та станом на клієнтській стороні відповідає за комплексну організацію, зберігання, кешування та синхронізацію всієї інформації, яка необхідна для коректної роботи користувацького інтерфейсу та реалізації клієнтської бізнес-логіки. Ключовим завданням даного модуля є надання консистентного, реактивного та централізованого доступу до даних для всіх компонентів застосунку, оптимізація

взаємодії з сервером шляхом мінімізації надлишкових запитів, а також підтримка актуального стану користувацького сеансу та налаштувань.

В архітектурі клієнтської частини застосунку реалізовано структурований підхід до управління станом, що поєднує локальний та глобальний рівні. Для управління локальним станом окремих UI-компонентів, таких як тимчасові прапорці, стан елементів форм чи локальні візуальні ефекти, широко застосовуються вбудовані механізми React Hooks, зокрема useState та useEffect. Такий підхід забезпечує високий рівень інкапсуляції та модульності компонентів.

Для управління глобальним станом, доступ до якого є необхідним для багатьох компонентів застосунку, незалежно від їхнього положення в ієрархії, використовується React Context API. Цей механізм дозволив створити декілька спеціалізованих контекстів, що відповідають за різні аспекти глобальних даних. Зокрема, реалізовано SettingsContext, який керує налаштуваннями користувача, такими як обрана візуальна тема (світла/темна) та параметри сповіщень. DeviceContext відповідає за управління списком налаштованих пристроїв моніторингу, інформацією про поточний активний пристрій та його статус з'єднання із сервером. Центральним для відображення даних моніторингу є DataContext, що інкапсулює поточні значення індексу якості води (WQI), деталізовані параметри води (pH, TDS, турбідність, температура) та історичні дані, необхідні для побудови графіків та аналітики. Кожен з цих контекстів надає відповідний Provider та функції для модифікації стану, які викликаються у відповідь на дії користувача або при отриманні нових даних від серверного API.

Візуальна структура та взаємозв'язки ключових елементів модуля управління станом представлені на рисунку 3.4. На діаграмі відображено основні компоненти, такі як UI-компоненти, що використовують React Hooks для локального стану та споживають дані з глобальних контекстів (SettingsContext, DeviceContext, DataContext). Також показано взаємодію модуля API, який оновлює глобальний стан новими даними, та механізм локального збереження даних (AsyncStorage), що

забезпечує їх персистентність. Ця схема наочно демонструє архітектуру управління даними на клієнтській стороні.

Рисунок 3.4 − Діаграма компонентів модуля керування станом на клієнтській стороні

Для забезпечення збереження важливих даних користувача між сеансами роботи з додатком, інтегровано механізм персистентного локального зберігання, реалізований за допомогою бібліотеки AsyncStorage. Даний інструмент використовується для асинхронного збереження на пристрої та завантаження при старті застосунку таких даних, як конфігурації підключених пристроїв моніторингу, останній обраний активний пристрій, налаштування інтерфейсу користувача (наприклад, тема), а також кешовані некритичні дані для покращення продуктивності. Така персистенція значно покращує загальний досвід користувача, оскільки застосунок зберігає індивідуальні налаштування та контекст попередньої роботи.

Важливою функцією модуля є забезпечення синхронізації та реактивності даних. При отриманні нових даних від модуля взаємодії з API (наприклад, при періодичному оновленні параметрів води кожні 5 секунд), ініціюється оновлення

відповідних сегментів глобального стану через спеціалізовані функції, надані контекстами. Завдяки реактивній природі фреймворку React, будь-які зміни в глобальному стані автоматично спричиняють перерендер лише тих UI-компонентів, які залежать від змінених даних. Для оптимізації продуктивності та уникнення непотрібних перерендерів активно використовуються відповідні патерни програмування, такі як мемоізація компонентів та значень (React.memo, useMemo, useCallback). Усі структури даних, що використовуються для зберігання стану, строго типізовані за допомогою TypeScript, що забезпечує їх консистентність, передбачуваність та полегшує процес розробки й підтримки коду, спираючись на визначені інтерфейси, такі як WaterParameters та UserDevice. Модуль також інкапсулює логіку обробки станів завантаження даних та можливих помилок при їх отриманні, надаючи відповідну інформацію для відображення в UI.

Таким чином, реалізований модуль управління даними та станом на клієнті відіграє ключову роль у підтримці функціональності та інтерактивності застосунку. Він забезпечує структуроване та централізоване управління всіма релевантними даними, підтримує їх актуальність та консистентність, реалізує механізми кешування та персистентного зберігання, а також гарантує своєчасне та ефективне оновлення користувацького інтерфейсу у відповідь на зміни даних або дії користувача.

3.2.3 Створення модуля візуалізації даних моніторингу

Ключовим аспектом забезпечення ефективного сприйняття та аналізу результатів моніторингу якості води є їх належне візуальне представлення. Модуль візуалізації даних у кросплатформеному додатку "Water Monitoring App" був спроектований та реалізований з метою трансформації комплексних числових наборів даних, що включають значення індексу якості води (WQI) та окремих фізико-хімічних параметрів, на інтуїтивно зрозумілі, наочні та, за можливості, інтерактивні графічні елементи. Такий підхід не лише полегшує користувачам швидку оцінку поточного екологічного стану водних об'єктів, але й надає

інструменти для аналізу динаміки змін у часі, виявлення довгострокових трендів, сезонних коливань та потенційних аномальних відхилень. Для реалізації цієї функціональності була обрана бібліотека React Native Chart Kit, що дозволяє інтегрувати різноманітні типи діаграм та графіків безпосередньо в середовище React Native.

Центральним елементом інтерфейсу, що забезпечує миттєву оцінку загальної якості води, є спеціально розроблений компонент ScoreCircle. Цей компонент продемонстровано на головному екрані додатку (рисунок 3.5), візуалізує поточне розраховане значення індексу WQI у вигляді анімованої кругової діаграми, що функціонує як прогрес-бар. Важливою особливістю є динамічна зміна кольору цього індикатора - від насиченого синього для відмінної якості води до відтінків зеленого, жовтого, помаранчевого та червоного/коричневого для градацій від хорошої до дуже поганої якості. Таке кольорове кодування, що відповідає загальній адаптивній колірній системі додатку, дозволяє користувачеві здійснити експрес-оцінку стану води, навіть не вдивляючись у числові значення. Естетичне та інформаційне навантаження цього елементу підсилюється фоновою анімацією хвиль, що створює тематичну атмосферу та візуально асоціюється з водою.

Рисунок 3.5 − Компонент ScoreCircle для відображення загального індексу якості води

Для поглибленого аналізу динаміки зміни як загального індексу WQI, так і окремих параметрів, було створено компонент WQIChartView. Приклад роботи цього компонента у вигляді модального вікна, що з'являється після взаємодії користувача зі зведеною панеллю WQI, наведено на рисунку 3.6. Цей екран надає користувачеві поточне значення WQI із текстовою інтерпретацією його рівня та часом останнього оновлення даних. Основним елементом є лінійний графік, що наочно демонструє коливання індексу якості води за часовий інтервал, наприклад, за останні 8 годин. Графік доповнено відображенням агрегованих статистичних даних - мінімального, максимального та середнього значень WQI за довший період, наприклад, за останні 24 години. Таке поєднання графічної та числової інформації дозволяє не тільки відстежувати короткострокові флуктуації, але й оцінювати загальну стабільність та діапазон змін якості води.

Рисунок 3.6 − Компонент WQIChartView для відображення діаграми змін значень якоості води за певний проміжок часу

Окрім агрегованого індексу WQI, модуль надає можливість детального аналізу кожного окремого параметра, що вимірюється. Екран для відображення параметрів якості води (рисунок. 3.7) представлений у вигляді списку ключових показників, таких як рівень pH, температура, загальний вміст розчинених речовин (TDS) та каламутність. Для кожного параметра в цьому списку відображається його поточне числове значення, візуальний індикатор відповідності оптимальному діапазону та мініатюрний прев'ю-графік, що показує динаміку змін за останні 24 години. Це дозволяє користувачеві швидко охопити поточний стан всіх основних параметрів та виявити ті, що потребують більш пильної уваги. При виборі конкретного параметра зі списку, наприклад, "Рівень pH", користувач переходить на екран детальної інформації (рисунок 3.8). Цей екран містить розгорнутий лінійний графік динаміки обраного параметра за останні 24 години, його точне поточне значення, розраховані статистичні показники (максимум, мінімум, середнє за період) та інформаційний блок з описом параметра та рекомендованим оптимальним діапазоном для питної води.

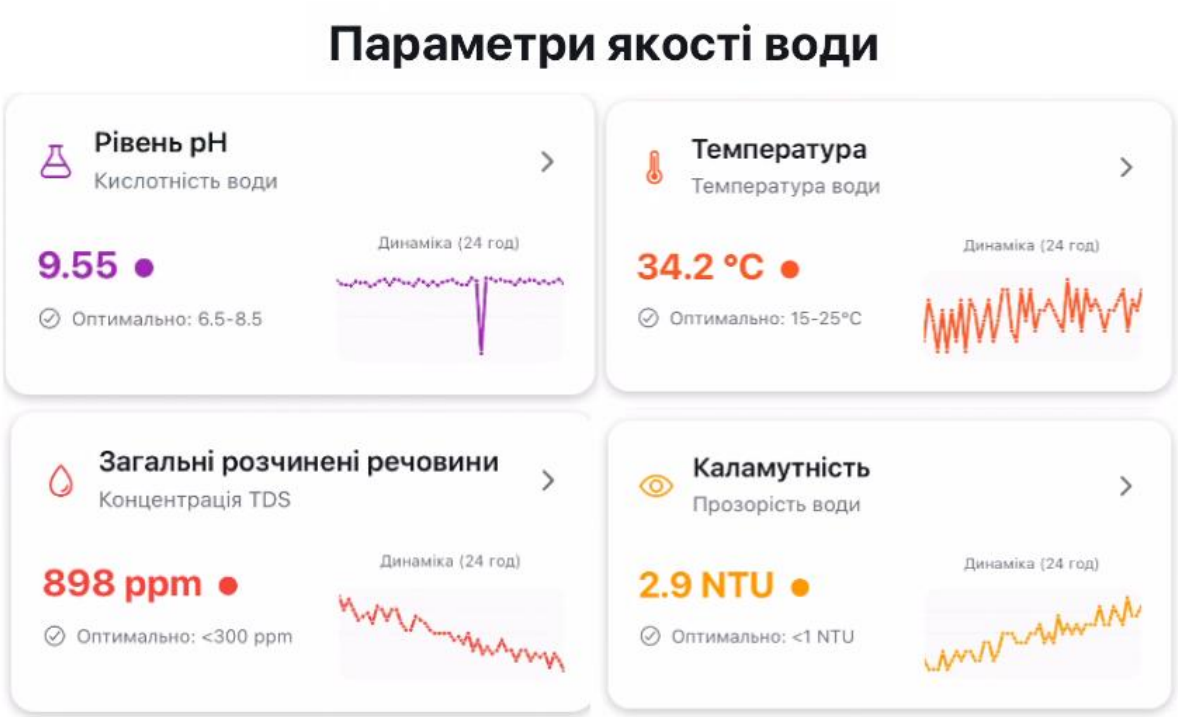


Рисунок 3.7 − Меню "Параметри якості води", що відображає поточні значення та міні-графіки динаміки для окремих показників.

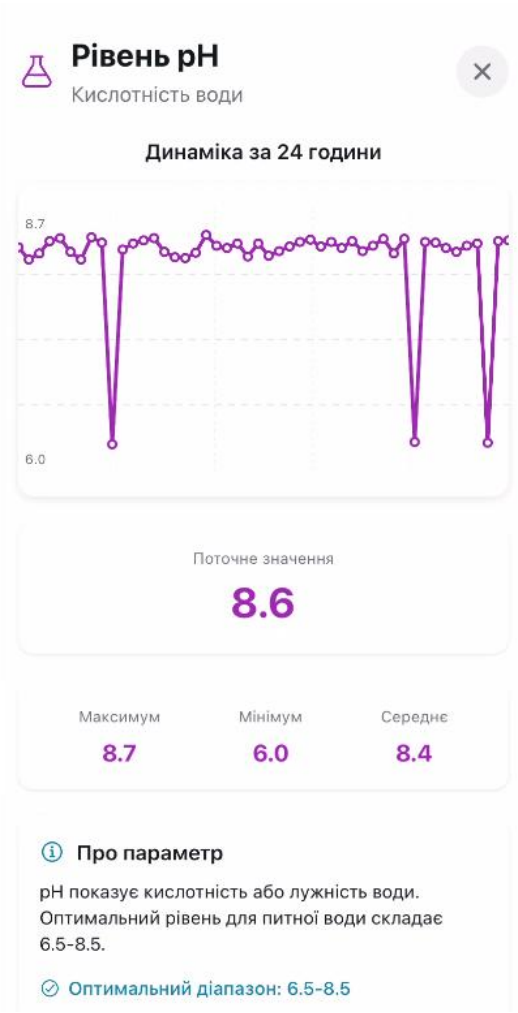


Рисунок 3.8 − Детальне відображення даних для параметру кислотності

Усі елементи візуалізації розроблені з урахуванням принципів зручності використання та інформативності. Застосування єдиної адаптивної колірної системи у всіх графічних компонентах забезпечує візуальну консистентність та полегшує інтуїтивне сприйняття даних. Модуль візуалізації отримує дані для відображення безпосередньо з централізованого модуля управління даними та станом, що гарантує їх актуальність та узгодженість з іншими частинами застосунку. Хоча основний акцент зроблено на наочному представленні, реалізовано базові елементи інтерактивності, зокрема можливість переходу до більш детальних графіків та екранів. Значна увага приділялася оптимізації продуктивності рендерингу діаграм, особливо при роботі з потенційно великими масивами історичних даних, шляхом застосування технік мемоізації та

ефективного управління оновленнями компонентів React. Передбачається можливість подальшого розвитку модуля, включаючи додавання більш складних інтерактивних елементів, таких як масштабування графіків "щипком", вибір довільних часових діапазонів для аналізу, а також функціонал експорту візуалізованих даних у поширених форматах для створення звітів або використання в інших аналітичних системах.

Таким чином, створений модуль візуалізації даних моніторингу є невід'ємною та важливою складовою реаілзованого додатку. Він перетворює сирі числові дані на зрозумілі візуальні образи, надаючи користувачам потужні інструменти для швидкої оцінки якості води, аналізу її динаміки в часі та детального вивчення окремих параметрів, що суттєво підвищує цінність та ефективність усієї системи екологічного моніторингу.

3.2.4 Реалізація модуля користувацького інтерфейсу та навігації

Розробка модуля користувацького інтерфейсу та навігації для кросплатформеного додатку мала на меті створення єдиного, інтуїтивно зрозумілого та функціонального середовища взаємодії для користувачів як на мобільних пристроях, так і в веб-браузерах на настільних комп'ютерах. Ключовим завданням була технічна реалізація UI-компонентів та навігаційної структури, що забезпечували б консистентний досвід користувача незалежно від платформи, з урахуванням специфіки та очікувань користувачів кожного середовища.

Основою для побудови інтерфейсу слугували стандартні компоненти фреймворку React Native та можливості платформи Expo, що сприяло створенню універсальних UI-елементів. Особлива увага приділялася розробці перевикористовуваних компонентів, таких як тематизовані ThemedText та ThemedView, що спираються на централізовану колірну палітру та підтримують адаптивність до світлої та темної тем оформлення завдяки кастомному хуку useColorScheme.ts. Приклади реалізації інтерфейсу для десктопної веб-версії та

мобільної версії головного екрану моніторингу представлені на рисунках 3.9 та 3.10 відповідно.

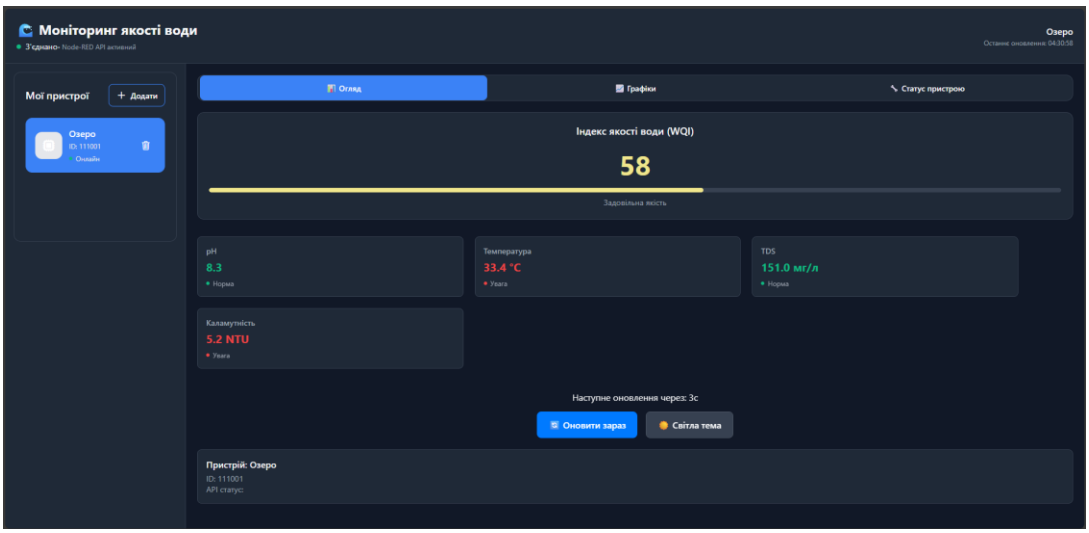


Рисунок 3.9 − Інтерфейс головного екрану моніторингу додатку (веб-версія)

Рисунок 3.10 − Інтерфейс головного екрану моніторингу додатку (мобільна версія)

Дані зображення демонструють, як ключові інформаційні блоки, такі як статус пристрою та оглядові картки окремих параметрів (pH, температура, TDS, каламутність), адаптуються до різних розмірів екрану та візуальних тем. Важливо відзначити, що підхід до візуалізації центрального показника WQI відрізняється: на мобільній версії використовується виразний круговий індикатор ScoreCircle з анімацією хвиль для наочного представлення індексу, тоді як десктопна версія представляє WQI у більш стриманому вигляді числового значення та горизонтального індикатора на головній інформаційній панелі, що краще відповідає традиційним дашбордам. На десктопній версії інформація розташована більш розлого, з використанням бічної панелі для вибору пристрою, тоді як мобільна версія оптимізована для вертикального скролінгу та компактного представлення даних з використанням плаваючих кнопок для зміни теми та додавання пристроїв.

Для специфічних візуальних потреб були створені такі компоненти, як DeviceStatusView для індикації стану підключення та ExpandedParameterChart для показників на мобільних пристроях або в деталізованих поданнях. Для покращення тактильної взаємодії на мобільних пристроях інтегровано бібліотеку Expo Haptics. При розробці компонентів застосовувалися принципи адаптивного дизайну та гнучкої верстки для забезпечення коректного відображення на різних роздільних здатностях екранів, що передбачало умовний рендеринг певних елементів або зміну їхньої компоновки залежно від платформи та розміру екрану.

Навігаційна структура додатку реалізована за допомогою інструменту Expo Router, що підтримує файлову систему маршрутизації та дозволяє гнучко визначати навігаційні шляхи для різних платформ. Наприклад, десктопна веб-версія використовує систему горизонтальних вкладок для зручного перемикання між основними розділами функціоналу в межах обраного пристрою моніторингу. Для мобільної версії було обрано підхід, орієнтований на жести: основна навігація між ключовими екранами або поданнями даних реалізована за допомогою свайпів (горизонтальних жестів), що забезпечує плавний та інтуїтивний перехід без

необхідності використання традиційних навігаційних панелей чи меню-бургерів. При цьому для переходів на екрани деталізації параметрів (як було показано на рисунках 3.7 та 3.8 у розділі 3.2.3) або для відображення модальних вікон використовується стекова навігація, керована через Expo Router та доповнена можливостями бібліотеки React Navigation. Такий підхід до навігації на мобільних пристроях мінімізує кількість видимих навігаційних елементів на екрані, роблячи акцент на безпосередньому контенті та даних моніторингу.

Технічна реалізація управління взаємодією користувача включала обробку натискань (за допомогою компонентів Pressable), жестів (з використанням бібліотеки React Native Gesture Handler для мобільних платформ) та інших подій вводу, які ініціювали відповідні дії: оновлення стану в модулі управління даними, запити до модуля API, або навігаційні переходи. Для забезпечення коректного відображення інтерфейсу на різних пристроях, особливо мобільних з урахуванням системних елементів, таких як вирізи камери або навігаційні панелі, використовувався Safe Area Context. Динамічне управління системним статус-баром (колір, видимість) здійснювалося за допомогою Expo Status Bar. Використання стандартизованого набору іконок з бібліотеки @expo/vector-icons забезпечило візуальну консистентність та чіткість навігаційних елементів, кнопок та індикаторів на всіх підтримуваних платформах.

Таким чином, технічна реалізація модуля користувацького інтерфейсу та навігації була спрямована на створення гнучкої, адаптивної та візуально консистентної системи, що забезпечує ефективну та зручну взаємодію користувача з додатком як на мобільних, так і на десктопних платформах. Це досягається шляхом застосування як спільних перевикористовуваних компонентів, так і специфічних для кожної платформи UI-рішень та навігаційних патернів, що підтверджується представленими візуальними прикладами інтерфейсу.

3.2.5 Розробка модуля обробки сповіщень

Своєчасне інформування користувача про критичні зміни у стані якості води або про важливі системні події є невід'ємною частиною ефективної системи моніторингу. Модуль обробки сповіщень у додатку проектувався з метою забезпечення механізмів прийому, обробки та відображення таких повідомлень для користувача, що дозволяє оперативно реагувати на потенційні загрози або важливу інформацію. Хоча повноцінна реалізація push-сповіщень через зовнішні сервіси є плановим покращенням, на поточному етапі розробки було закладено основи для внутрішньосистемних сповіщень та підготовлено архітектуру для майбутньої інтеграції більш розширених механізмів.

Основна функціональність модуля на даному етапі зосереджена на обробці подій, що генеруються всередині клієнтського додатку або надходять від серверної частини у відповідь на запити. Передбачається, що модуль взаємодії з API може отримувати спеціальні індикатори від сервера у випадку виявлення аномальних значень параметрів. На клієнтській стороні, модуль обробки сповіщень відповідає за прийом та інтерпретацію сигналів про такі події, що може бути реалізовано через підписку на зміни у глобальному стані, де модуль управління даними встановлює відповідні індикатори тривоги. На основі отриманого сигналу модуль генерує зрозумілий для користувача текст повідомлення та визначає його тип (інформаційне, попереджувальне, критичне).

Для відображення сповіщень користувачеві реалізовано базові механізми безпосередньо в інтерфейсі додатку, такі як невеликі спливаючі повідомлення (toasts або snackbars), які тимчасово інформують про подію, з візуальним оформленням, що залежить від критичності повідомлення. Архітектурно закладається можливість зв'язування сповіщення з певним контекстом або екраном у додатку для отримання більш детальної інформації. Логіка обробки та генерації сповіщень інкапсульована в окремих функціях або хуках, які можуть бути викликані з різних частин застосунку. Налаштування порогів та умов для генерації сповіщень, наприклад, на основі "Системи штрафів" в алгоритмі WQI, є важливим

аспектом, що визначає релевантність та своєчасність інформування. Хоча поточна реалізація сфокусована на внутрішньододаткових повідомленнях, архітектура модуля розроблялася з урахуванням легкої інтеграції з push-сповіщеннями в майбутньому, дозволяючи перевикористати логіку визначення подій та формування змісту повідомлень.

Таким чином, модуль обробки сповіщень, навіть з базовою функціональністю, є важливим елементом системи, що підвищує її інтерактивність та здатність своєчасно інформувати користувача про значущі події.

3.3 Візуальний дизайн та колірне інформування в інтерфейсі додатку

Розробка користувацького інтерфейсу (UI) для кросплатформеного додатку "Water Monitoring App" ґрунтувалася не лише на забезпеченні функціональності, але й на створенні ефективної системи візуального інформування користувача. Ключову роль у цьому відіграли продумані дизайн-рішення, зокрема використання адаптивної колірної палітри та тематизації для швидкого та інтуїтивного сприйняття стану якості води та окремих її параметрів. Метою було створення інтерфейсу, де колір виступає не просто естетичним елементом, а й важливим носієм інформації, що сприяє оперативному прийняттю рішень користувачем.

Центральним елементом системи візуального інформування стала розроблена адаптивна колірна палітра, інтегрована у всі компоненти, що відображають дані моніторингу.

Рисунок 3.11 − Шкала якості води

Основний принцип полягав у використанні спектру кольорів для градації якості води, що базується на розрахованому індексі WQI та відхиленнях окремих параметрів від оптимальних значень. Шкала якості води (рисунок 3.11), що включає рівні "Відмінна", "Хороша", "Прийнятна", "Погана" та "Дуже погана", отримала відповідне колірне представлення. Відмінна якість візуалізується насиченими синіми або яскраво-зеленими відтінками. Хороша якість відображається світлішими відтінками зеленого. Прийнятна якість індикується жовтими або світло-помаранчевими кольорами. Погана якість позначається насиченими помаранчевими або червоно-помаранчевими відтінками, а дуже погана якість води візуалізується червоними або темно-червоними відтінками, що сигналізують про критичний стан. Це колірне кодування послідовно застосовується у ключових UI-елементах, таких як круговий індикатор WQI, горизонтальний індикатор на десктопній версії та індикатори стану окремих параметрів.

Окрім інформаційного колірного кодування даних, додаток підтримує світлу та темну теми оформлення, що дозволяє користувачеві налаштувати візуальний стиль. Як показано на рисунках 3.9 та 3.10, перемикання теми змінює фонові кольори та елементи управління, зберігаючи контрастність. Ця функціональність реалізована за допомогою кастомного хука useColorScheme та тематизованих компонентів, що автоматично адаптують свій вигляд.

Інформування кольором також активно використовується при відображенні окремих параметрів якості води. На екрані "Параметри якості води" (рисунок 3.7) біля кожного числового значення параметра присутній кольоровий індикатор-крапка. Колір цього індикатора сигналізує про відповідність значення параметра нормі або про наявність відхилення. Такі візуальні підказка дозволяють користувачеві миттєво ідентифікувати параметри, що потребують уваги.

Для покращення сприйняття даних та надання інтерфейсу більш динамічного вигляду використовуються анімаційні ефекти. Анімація хвиль, що слугує фоном для кругового індикатора WQI на мобільній версії (рисунок 3.10), може динамічно змінювати свій колір залежно від рівня WQI, додатково підсилюючи візуальне

інформування. Плавні переходи станів та анімації значень у діаграмах також сприяють кращому розумінню динаміки змін.

Система візуального дизайну та колірного інформування в додатку спроектована як цілісний механізм, що перетворює складні дані моніторингу на легкозрозумілі візуальні сигнали. Це дозволяє користувачеві швидко оцінювати ситуацію, виявляти проблемні аспекти та приймати обґрунтовані рішення, що є критично важливим для ефективного контролю якості води

3.4. Тестування розробленого програмного забезпечення

Процес тестування розробленого кросплатформеного додатку "Water Monitoring App" був невід'ємною частиною циклу розробки, спрямованою на забезпечення належної якості, стабільності функціонування та відповідності програмного забезпечення поставленим вимогам. Тестування охоплювало різні рівні та аспекти роботи додатку, від перевірки окремих логічних блоків до комплексної оцінки взаємодії всіх модулів та користувацького досвіду.

Основним методом перевірки коректності реалізації алгоритму розрахунку індексу якості води (WQI) стало модульне тестування за допомогою спеціалізованих тестових скриптів (test-current-wqi.js, test-strict-wqi.js та інші), що дозволило верифікувати ключову бізнес-логіку додатку на різних наборах вхідних даних.

Функціональне тестування проводилося вручну та охоплювало перевірку всіх основних функцій. Перевірялася коректність відображення даних від симульованого серверного API, актуальність їх оновлення, робота навігаційної системи та модуля візуалізації. Важливою частиною функціонального тестування була перевірка поведінки додатку в нештатних ситуаціях та при взаємодії зі специфічними функціями. Наприклад, на рисунку 3.12 продемонстровано реакцію інтерфейсу на помилку підключення до сервера або таймаут мережевого запиту (Network request timed out). Як видно, додаток коректно інформує користувача про

проблему як у статусі пристрою ("Помилка підключення"), так і в центральному індикаторі WQI, відображаючи відповідне повідомлення про помилку та нульове значення індексу, що запобігає відображенню застарілих або некоректних даних.

Рисунок 3.12 − Реакція інтерфейсу на помилку підключення до сервера

Тестування специфічних функціональних можливостей, таких як процес калібрування датчиків, також було частиною перевірки. Хоча сам процес калібрування симулювався на стороні Node-RED, клієнтський додаток мав коректно ініціювати цей процес та обробляти відповідні сповіщення. На рисунку 3.13 показано приклад попереджувального сповіщення про те, що виявлено дрейф датчика pH і він потребує калібрування. Це ілюструє, як система може інформувати користувача про необхідність технічного обслуговування. Рисунок 3.14 демонструє інформаційне сповіщення про успішне завершення процесу калібрування, що підтверджує коректну взаємодію клієнта з відповідним API ендпоінтом (POST /api/calibrateSensors) та обробку відповіді від сервера. Також на

цих зображеннях видно секцію "Статус датчиків", яка відображає інформацію про стан та останнє калібрування, що також було об'єктом тестування.

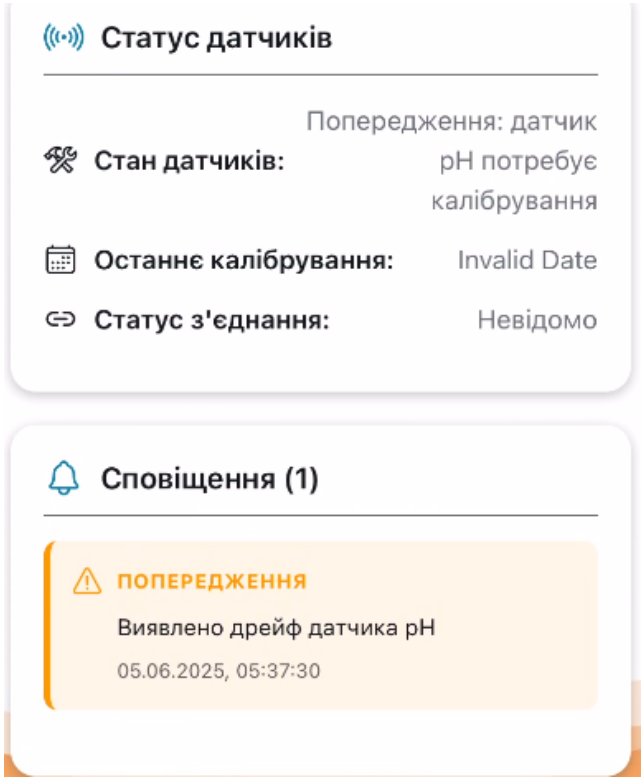


Рисунок 3.13 − Попередження датчика pH

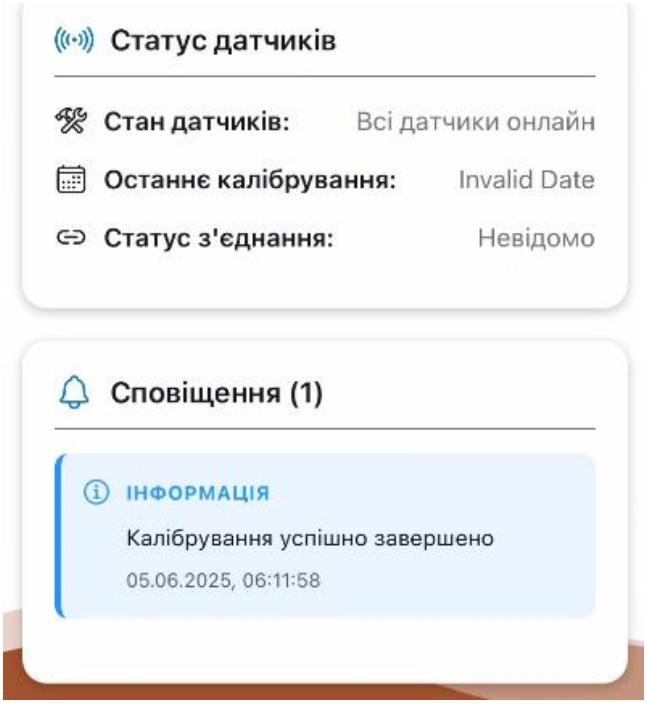


Рисунок 3.14 − Повідомлення про успішне калібрування датчиків

Тестування користувацького інтерфейсу було спрямоване на оцінку зручності та інтуїтивності взаємодії, читабельності текстів, контрастності елементів при різних темах оформлення та адаптивності до різних розмірів екранів.

Інтеграційне тестування полягало у перевірці коректної взаємодії між різними програмними модулями додатку. Особлива увага приділялася перевірці повного циклу обробки даних та реакції системи на помилки, імітовані на стороні симуляційного сервера Node-RED.

За результатами проведеного тестування були виявлені та усунені деякі незначні недоліки. В цілому, розроблене програмне забезпечення продемонструвало стабільну роботу, відповідність основним функціональним вимогам, коректність реалізації ключових алгоритмів та адекватну обробку помилкових станів, що підтверджується наведеними прикладами інтерфейсу. Подальше розширення покриття автоматизованими тестами є одним із напрямків майбутнього вдосконалення проєкту.

3.5. Висновки до третього розділу

У третьому розділі дипломної роботи було детально описано практичну реалізацію програмних модулів кросплатформеного додатку, призначеного для моніторингу стану якості води, а також проведено оцінку його функціональності через тестування. Основною метою цього етапу було втілення теоретичних та проектних рішень, розроблених у попередніх розділах, у функціонуючий програмний продукт.

Було детально розглянуто обране середовище розробки та інструментальні засоби, що включали фреймворк React Native, платформу Expo, мову програмування TypeScript, а також низку спеціалізованих бібліотек для управління станом, навігації, візуалізації даних та побудови користувацького інтерфейсу. Обґрунтовано вибір технологічного стеку, який дозволив забезпечити

кросплатформеність, ефективність розробки та сучасний рівень користувацького досвіду.

Здійснено послідовну реалізацію ключових програмних модулів додатку. Розроблено модуль взаємодії з серверним API, що інкапсулює логіку комунікації із симульованою на Node-RED серверною частиною, включаючи обробку запитів, відповідей та потенційних помилок. Імплементовано модуль управління даними та станом на клієнті, який використовує React Hooks та React Context API для управління локальним та глобальним станом застосунку, а також AsyncStorage для персистентного зберігання користувацьких налаштувань та даних. Створено модуль візуалізації даних моніторингу, що застосовує бібліотеку React Native Chart Kit для наочного представлення індексу WQI та окремих параметрів якості води у вигляді інтерактивних графіків та діаграм. Також було реалізовано модуль користувацького інтерфейсу та навігації, який забезпечує інтуїтивно зрозумілу взаємодію на мобільних та веб-платформах, з урахуванням адаптивного дизайну та колірного інформування. Окремо розглянуто підходи до розробки модуля обробки сповіщень, що закладає основи для інформування користувача про важливі події.

Важливою частиною практичної реалізації стало детальне опрацювання дизайну користувацького інтерфейсу, з особливим акцентом на візуальне інформування через колірне кодування станів якості води та окремих параметрів, а також на підтримку світлої та темної тем оформлення для покращення користувацького досвіду.

Проведене тестування розробленого програмного забезпечення охоплювало модульне тестування ключових алгоритмів, функціональне тестування основних можливостей додатку, перевірку користувацького інтерфейсу та тестування обробки помилкових ситуацій, таких як втрата зв'язку з сервером або некоректна робота симульованих датчиків. Результати тестування підтвердили працездатність додатку, коректність реалізації його основних функцій та відповідність поставленим вимогам.

Таким чином, у рамках третього розділу було успішно завершено етап практичної реалізації програмної частини системи моніторингу стану води. Створений кросплатформений додаток "Water Monitoring App" демонструє можливість ефективного поєднання сучасних технологій для розробки інформативних та зручних у використанні інструментів екологічного моніторингу. Результати, отримані в цьому розділі, є основою для подальшого розвитку та вдосконалення системи.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було розроблено та реалізовано програмну частину кіберфізичної системи моніторингу стану води у вигляді кросплатформеного додатку, що забезпечує збір, обробку, візуалізацію даних про якість води та інформування користувача в режимі, наближеному до реального часу. Досягнуто поставленої мети шляхом вирішення низки завдань, пов'язаних з аналізом предметної області, проектуванням архітектури та програмних модулів, а також їх практичною імплементацією та тестуванням.

У першому розділі проведено комплексний аналіз предметної області моніторингу якості водних ресурсів та існуючих підходів до реалізації кіберфізичних систем у цій сфері. Було виявлено недоліки традиційних методів контролю та обґрунтовано актуальність застосування сучасних IoT-технологій та програмних рішень. Проаналізовано ключові функції та вимоги до програмної частини таких систем, включаючи збір даних з сенсорів, їх обробку, розрахунок інтегральних показників, візуалізацію, зберігання історії та систему сповіщень. Розглянуто основні архітектурні підходи та технологічні стеки, що застосовуються для розробки подібних систем, що стало основою для формування вимог до власного проєкту.

У другому розділі проведено детальне проектування архітектурно-функціональної моделі програмної частини кіберфізичної системи моніторингу стану води, що реалізується у вигляді кросплатформеного додатку. Було визначено загальну архітектуру взаємодії клієнтської частини із серверною інфраструктурою (симульованою на Node-RED на етапі розробки) через API. Розроблено структуру ключових програмних модулів клієнтського додатку, включаючи модуль взаємодії з API, модуль управління даними та станом, модуль візуалізації даних, модуль користувацького інтерфейсу та навігації, а також модуль обробки сповіщень. Для кожного модуля визначено його основні функції та взаємозв'язки. Також було

обґрунтовано вибір технологічного стеку для реалізації програмної частини, що включає React Native, Expo, TypeScript та супутні бібліотеки.

У третьому розділі здійснено практичну реалізацію розроблених програмних модулів кросплатформеного додатку та проведено його комплексне тестування. Детально описано середовище розробки та інструментальні засоби, використані для імплементації. Представлено процес створення кожного з ключових програмних модулів, включаючи особливості їхньої взаємодії та реалізацію основних функцій. Особливу увагу приділено розробці користувацького інтерфейсу, зокрема системі візуального інформування через колірне кодування стану якості води та підтримці адаптивності для різних платформ (мобільні пристрої та веб). Проведене тестування підтвердило працездатність додатку, коректність реалізації алгоритму розрахунку WQI, адекватність обробки даних та взаємодії з симульованим серверним API, а також зручність користувацького інтерфейсу.

Результатом виконання дипломної роботи є функціонуючий прототип програмної частини кіберфізичної системи моніторингу стану води, що демонструє можливість ефективного застосування сучасних технологій для вирішення актуальних завдань екологічного контролю. Розроблений додаток має потенціал для подальшого розвитку та впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Adjovu G. E., Stephen H., James D., Ahmad S. Overview of the application of remote sensing in effective monitoring of water quality parameters. *Remote Sensing*. 2023. Vol. 15, No. 7. P. 1938.

2. Al Hanif A., Ilyas M. Effective feature engineering framework for securing MQTT protocol in IoT environments. *Sensors*. 2024. Vol. 24, No. 6. P. 1782.

3. Arnemann L., Galera S. L., Winter S., Schleich B. Gamification of Resource Consumption Monitoring of Products and Machines: A Cross-Platform and User-Friendly Approach. *Procedia CIRP*. 2024. Vol. 122. P. 569-574.

4. Baig M. J. A., Iqbal M. T., Jamil M., Khan J. Design and implementation of an open-Source IoT and blockchain-based peer-to-peer energy trading platform using ESP32-S2, Node-Red and, MQTT protocol. *Energy reports*. 2021. Vol. 7. P. 5733-5746.

5. Barbosa R., Ogobuchi O. D., Joy O. O., Saadi M., Rosa R. L., Al Otaibi S., Rodríguez D. Z. IoT based real-time traffic monitoring system using images sensors by sparse deep learning algorithm. *Computer Communications*. 2023. Vol. 210. P. 321-330.

6. Cao J., Lam K-Y., Lee L-H., Liu X., Hui P., Su X. Mobile augmented reality: User interfaces, frameworks, and intelligence. *ACM Computing Surveys*. 2023. Vol. 55, No. 9. P. 1-36.

7. Chen J., Chen S., Fu R., Li D., Jiang H., Wang C., Peng Y., Jia K., Hicks B. J. Remote sensing big data for water environment monitoring: Current status, challenges, and future prospects. *Earth's Future*. 2022. Vol. 10, No. 2. P. e2021EF002289.

8. Cheng C., Zhang F., Shi J., Kung H-T. What is the relationship between land use and surface water quality? A review and prospects from remote sensing perspective. *Environmental Science and Pollution Research*. 2022. Vol. 29, No. 38. P. 56887-56907.

9. Duo W., Zhou M., Abusorrah A. A survey of cyber attacks on cyber physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*. 2022. Vol. 9, No. 5. P. 784-800.

10. Fauzan R., Krisnahati I., Nurwibawa B. D., Wibowo D. A. A systematic literature review on progressive web application practice and challenges. *IPTEK The Journal for Technology and Science*. 2022. Vol. 33, No. 1. P. 43-58.

11. Frincu R. M. Artificial intelligence in water quality monitoring: A review of water quality assessment applications. *Water Quality Research Journal*. 2025. Vol. 60, No. 1. P. 164-176.

12. Garg G., Gupta S., Mishra P., Vidyarthi A., Singh A., Ali A. CROPCARE: an intelligent real-time sustainable IoT system for crop disease detection using mobile vision. *IEEE Internet of Things Journal*. 2021. Vol. 10, No. 4. P. 2840-2851.

13. Gavrilaș S., Ursachi C. Ș., Perța-Crișan S., Munteanu F-D. Recent trends in biosensors for environmental quality monitoring. *Sensors*. 2022. Vol. 22, No. 4. P. 1513.

14. Gulati K., Boddu R. S. K., Kapila D., Bangare S. L., Chandnani N., Saravanan G. A review paper on wireless sensor network techniques in Internet of Things (IoT). *Materials Today: Proceedings*. 2022. Vol. 51. P. 161-165.

15. Habib M. K., Chimsom C. CPS: Role, characteristics, architectures and future potentials. *Procedia Computer Science*. 2022. Vol. 200. P. 1347-1358.

16. Hairom N. H. H., Soon C. F., Mohamed R. M. S. R., Morsin M., Zainal N., Nayan N., Zulkifli C. Z., Harun N. H. A review of nanotechnological applications to detect and control surface water pollution. *Environmental Technology & Innovation*. 2021. Vol. 24. P. 102032.

17. Hamdani, Pulungan A. B., Myori D. E., Elmubdi F., Hasannuddin T. Real time monitoring system on solar panel orientation control using visual basic. *Journal of Applied Engineering and Technological Science (JAETS)*. 2021. Vol. 2, No. 2. P. 112-124.

18. Jabbar W. A., Subramaniam T., Ong A. E., Shu'Ib M. I., Wu W., De Oliveira M. A. LoRaWAN-based IoT system implementation for long-range outdoor air quality monitoring. *Internet of Things*. 2022. Vol. 19. P. 100540.

19. Jabbar W. A., Ting T. M., Hamidun M. F. I., Kamarudin A. H. C., Wu W., Sultan J., Alsewari A., Ali M. A. H. Development of LoRaWAN-based IoT system for

water quality monitoring in rural areas. *Expert systems with applications*. 2024. Vol. 242. P. 122862.

20. Jan F., Min-Allah N., Düştegör D. Iot based smart water quality monitoring: Recent techniques, trends and challenges for domestic applications. *Water*. 2021. Vol. 13, No. 13. P. 1729.

21. Joesphine N., Martin R., Yoga T. O., Sundaram D. Evaluation Implementation of Ui/Ux in Monitoring & Controlling Study Improvement With User-Centered Design Method. *J. Theor. Appl. Inf. Technol*. 2022. Vol. 100, No. 9. P. 3129-3156.

22. Kapalanga T. S., Hoko Z., Gumindoga W., Chikwiramakomo L. Remote-sensing-based algorithms for water quality monitoring in Olushandja Dam, north-central Namibia. *Water Supply*. 2021. Vol. 21, No. 5. P. 1878-1894.

23. Kaplan G., Yalcinkaya F., Altıok E., Pietrelli A., Nastro R. A., Lovecchio N., Ieropoulos I. A., Tsipa A. The role of remote sensing in the evolution of water pollution detection and monitoring: A comprehensive review. *Physics and Chemistry of the Earth, Parts A/B/C*. 2024. P. 103712.

24. Kato T., Fukumoto N., Sasaki C., Tagami A., Nakao A. Challenges of CPS/IoT Network Architecture in 6G Era. *IEEE Access*. 2024.

25. Khandagale A. S., Nichit P., Kulkarni A., Pagare P., Thakare Y. Mobile App for Addressing Social Issue to PMC Using React Native.

26. Kumar A., Vijayalakshmi S., Baswaraj D., Selvarajan P., Chandramohan S., Tiwari M. Towards Internet of Things: Integration of Wireless Sensor Network to Cloud Services for Data Collection and Sharing. B: *2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)*. 2022. P. 363-369.

27. Lakshminarayana S., Praseed A., Thilagam P. S. Securing the IoT application layer from an MQTT protocol perspective: Challenges and research prospects. *IEEE Communications Surveys & Tutorials*. 2024.

28. Li R., Liang P., Soliman M., Avgeriou P. Understanding software architecture erosion: A systematic mapping study. *Journal of Software: Evolution and Process*. 2022. Vol. 34, No. 3. P. e2423.

29. Liang S., Khalafbeigi T., van Der Schaaf H., Miles B., Schleidt K., Grellet S., Beaufils M., Alzona M. OGC SensorThings API Part 1: sensing version 1.1. 2021.

30. Mahajan S. Design and development of an open-source framework for citizen-centric environmental monitoring and data analysis. *Scientific Reports*. 2022. Vol. 12, No. 1. P. 14416.

31. Majid M., Habib S., Javed A. R., Rizwan M., Srivastava G., Gadekallu T. R., Lin J. C-W. Applications of wireless sensor networks and internet of things frameworks in the industry revolution 4.0: A systematic literature review. *Sensors*. 2022. Vol. 22, No. 6. P. 2087.

32. Matilla D. M., Murciego A. L., Jiménez-Bravo D. M., Mendes A. S., Leithardt V. R. Q. Low-cost Edge Computing devices and novel user interfaces for monitoring pivot irrigation systems based on Internet of Things and LoRaWAN technologies. *Biosystems Engineering*. 2022. Vol. 223. P. 14-29.

33. Neale P. A., Escher B. I., de Baat M. L., Dechesne M., Dingemans M. M. L., Enault J., Pronk G. J., Smeets P. W. M. H., Leusch F. D. L. Application of effect-based methods to water quality monitoring: answering frequently asked questions by water quality managers, regulators, and policy makers. *Environmental science & technology*. 2023. Vol. 57, No. 15. P. 6023-6032.

34. Paramartha D. Y., Fitriyani A. L., Pramana S. Development of Automated Environmental Data Collection System and Environment Statistics Dashboard. *Indonesian Journal of Statistics and Its Applications*. 2021. Vol. 5, No. 2. P. 314-325.

35. Paris I. L. B. M., Habaebi M. H., Zyoud A. M. Implementation of SSL/TLS security with MQTT protocol in IoT environment. *Wireless Personal Communications*. 2023. Vol. 132, No. 1. P. 163-182.

36. Patel M., Mehta A., Chauhan N. C. Design of smart dashboard based on IoT & fog computing for smart cities. B: *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*. 2021. P. 458-462.

37. Qin H., Chen W., Li N., Wang T., Chen H., Yang G., Peng Y. CPS: Cross-interface network Partitioning and Scheduling towards QoS-aware data flow delivery in

multimedia IoT. *Journal of Network and Computer Applications*. 2023. Vol. 217. P. 103698.

38. Richards M., Ford N. Fundamentals of Software Architecture: A Modern Engineering Approach. O'Reilly Media, Inc., 2025.

39. Singh C., Basha S. A., Bhushan A. V., Venkatesan M., Chaturvedi A., Shrivastava A. A secure IoT based wireless sensor network data aggregation and dissemination system. *Cybernetics and Systems*. 2023. P. 1-13.

40. Tyagi A. K., Sreenath N. Cyber physical systems: Analyses, challenges and possible solutions. *Internet of Things and Cyber-Physical Systems*. 2021. Vol. 1. P. 22-33.

41. Wang Y-A., Shen B., Zou L., Han Q-L. A survey on recent advances in distributed filtering over sensor networks subject to communication constraints. *International Journal of Network Dynamics and Intelligence*. 2023. P. 100007-100007.

42. Wu X., Liu C., Wang L., Bilal M. Internet of things-enabled real-time health monitoring system using deep learning. *Neural Computing and Applications*. 2023. P. 1-12.

43. Younan M., Khattab S., Bahgat R. From the wireless sensor networks (WSNs) to the Web of Things (WoT): an overview. *J. Intell. Syst. Internet Things*. 2021. Vol. 4, No. 2. P. 56-68.

44. Yu Z., Gao H., Cong X., Wu N., Song H. H. A survey on cyber--physical systems security. *IEEE Internet of Things Journal*. 2023. Vol. 10, No. 24. P. 21670-21686.

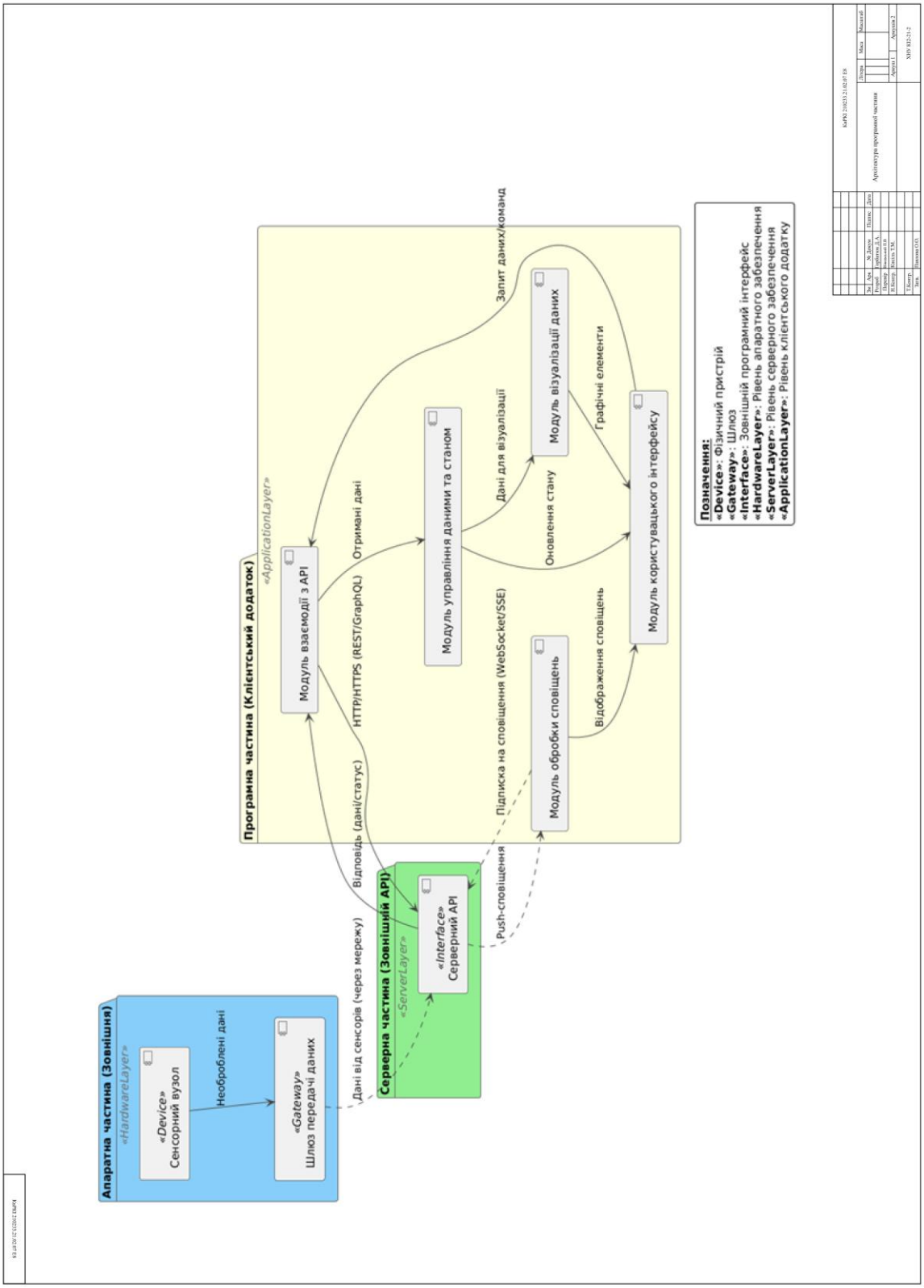
45. Zainuddin Z. Q. M., Yahya F., Moung E. G., Fazli B. M., Abdullah M. F. Effective dashboards for urban water security monitoring and evaluation. *Int. J. Electr. Comput. Eng*. 2023. Vol. 13. P. 4291-4305.

46. Zhou G., Zhang H., Xu C., Zhou X., Liu Z., Zhao D., Lin J., Wu G. A real-time data acquisition system for single-band bathymetric LiDAR. *IEEE Transactions on Geoscience and Remote Sensing*. 2023. Vol. 61. P. 1-21.

Додаток А

(обов’язковий)

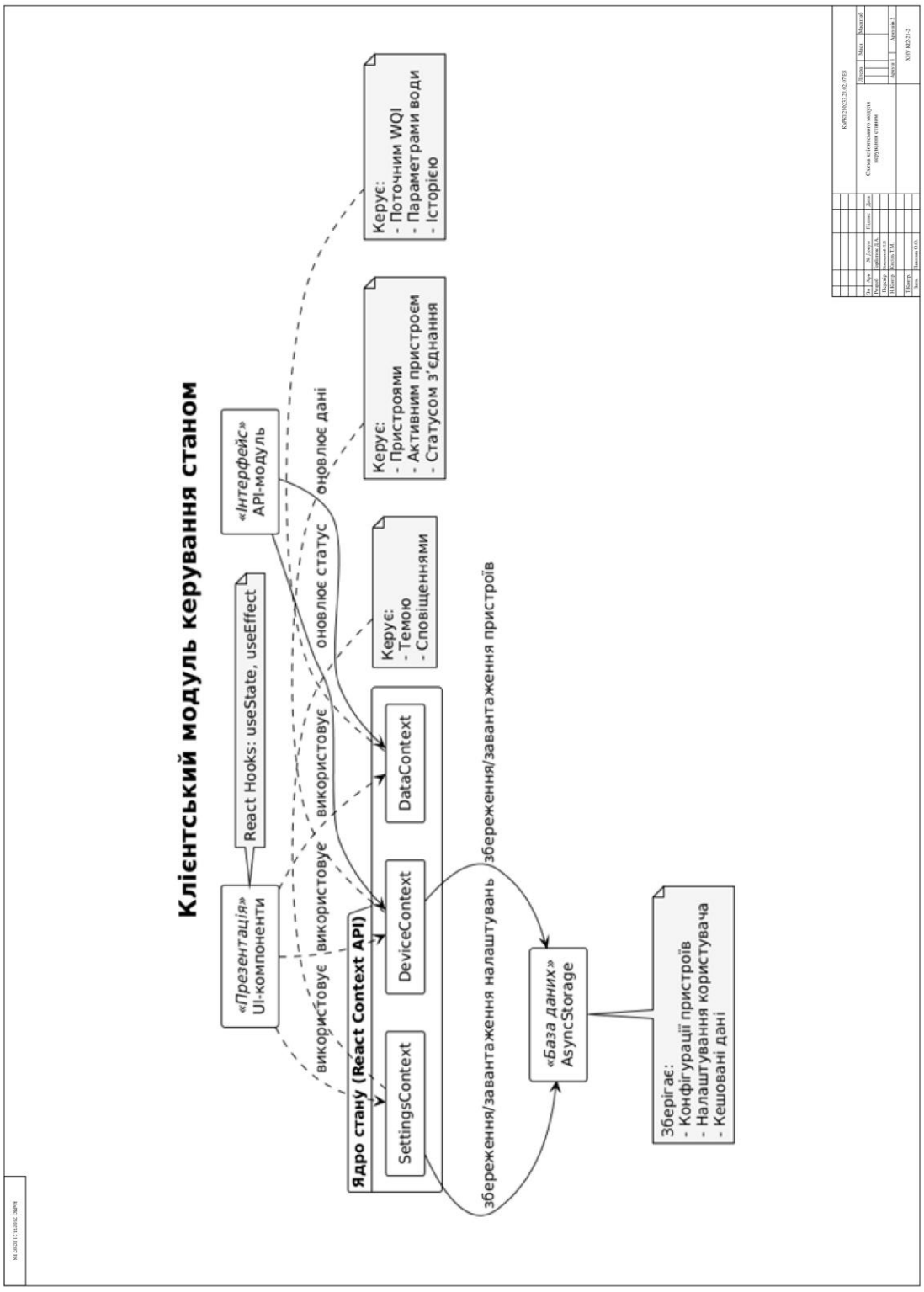
КОПІЯ КРЕСЛЕННЯ «АРХІТЕКТУРА ПРОГРАМНОЇ ЧАСТИНИ»



Додаток Б

(обов’язковий)

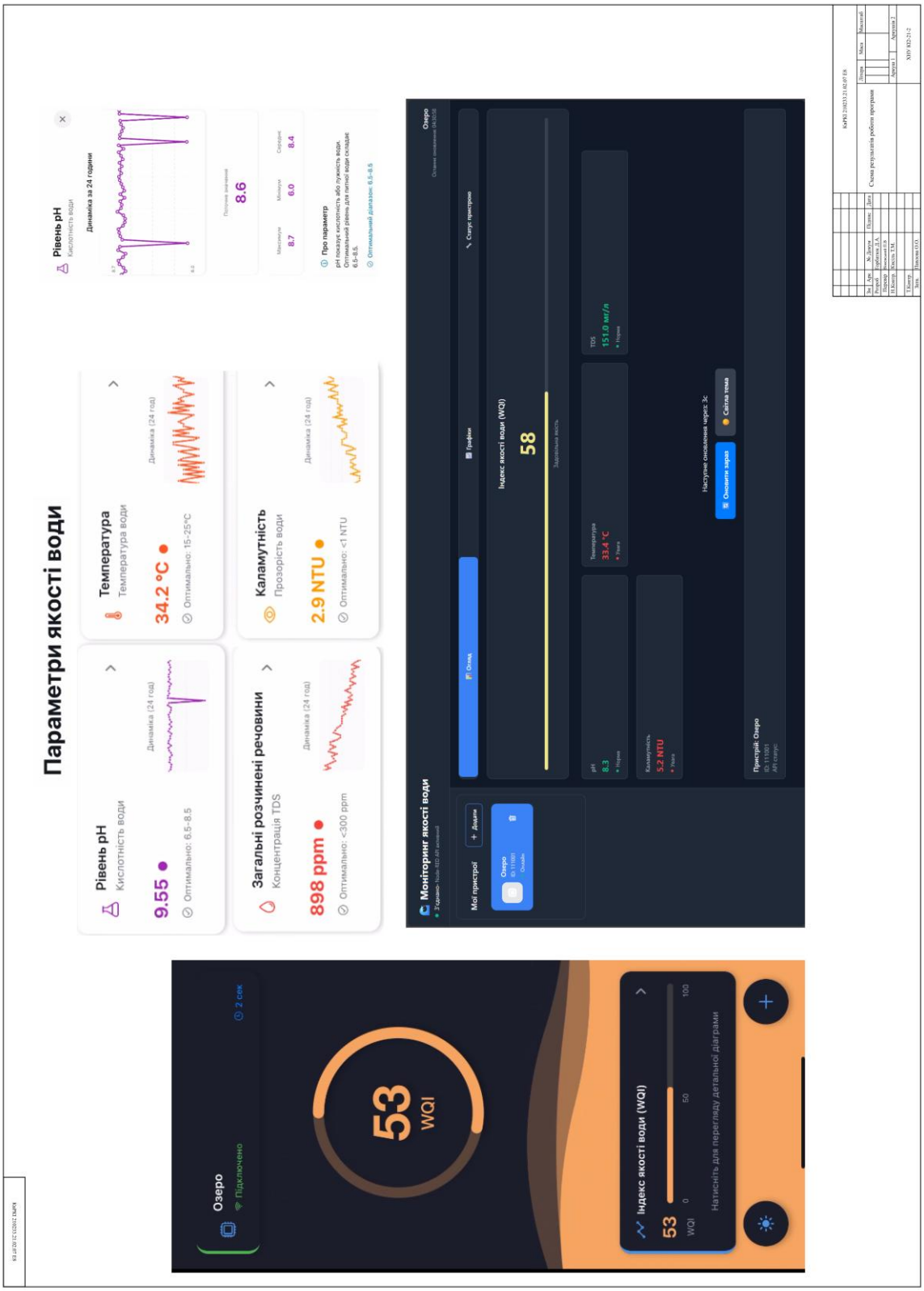
КОПІЯ КРЕСЛЕННЯ «КЛІЄНТСЬКИЙ МОДУЛЬ КЕРУВАННЯ СТАНОМ»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «СХЕМА РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМИ»



Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 12.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

ID: 244065 Title: БКР Кіберфізична система моніторингу стану води. Програмна частина Added in a DB: 2025-06-06 Authors: Дмитро ГОРБАТЮК Heads: Петро ВІЖЕВСЬКИЙ Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	113417	659	14733 (13%)	96 (15%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
240730	Title: Звіт ПДП Кіберфізична система моніторингу стану води. Програмна частина Added in a DB: 2025-05-01 Authors: Горбатюк Д.А. Heads: Говорущенко Т.О. Consultants: Opponents:	13576 (12.0%)	83 (13.0%)

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Дмитро ГОРБАТЮК

Співавтор:

Назва: Горбатюк_Кіберфізична система моніторингу стану води. Програмна частина

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:6%

Коефіцієнт подібності 2:3.4%

Мікропробіли: 14

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-06 19:21:38.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-07 Доцент Андрій Нічепорук

Дата експерт

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Дмитро ГОРБАТЮК

Тема: Кіберфізична система моніторингу стану води. Програмна частина

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___77___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є дослідження архітектурних та функціональних аспектів програмної частини кіберфізичної системи моніторингу стану води, а також розробка та тестування клієнтського мобільного додатку, призначеного для ефективного збору (через API), візуалізації та аналізу даних про якість водних ресурсів.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проведено аналіз предметної області та постановка задачі розробки програмної частини кіберфізичної системи моніторингу стану води. В другому розділі кваліфікаційної роботи виконано розробку архітектурно-функціональної моделі програмної частини кіберфізичної системи моніторингу води. В третьому розділі кваліфікаційної роботи описано практичну реалізацію програмних модулів мобільного додатку системи моніторингу стану води та оцінка його функціональності.

4. Позитивні сторони роботи: Висока практична цінність роботи.

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому інженерно-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (4.75/A)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Онищенко О.В.
доцент каф. 1173

"___" _____2025 р. _____(підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Дмитра ГОРБАТЮКА
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-21-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

9 червня 2025 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Кіберфізична система моніторингу стану води. Програмна частина

Автор: Дмитро ГОРБАТЮК

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Петро ВІЖЕВСЬКИЙ, асистент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) Запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) Усі запозичення є фрагментарними, а значна їх частина має належним чином оформлені посилання на використані джерела, що підтверджується наявністю маркерів цитування у тексті;
3) Окремі виявлені збіги є загальновживаними термінами, усталеними зворотами та стандартними формулюваннями, характерними для даної науково-технічної галузі, що не є об'єктом авторського права.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ідентичності/схожості StrikePlagiarism, складає 6.03% і адресується до 39 першоджерел; та системою Anti-Plagiarism складає 12%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Петро ВІЖЕВСЬКИЙ

Гарант ОП _____ Андрій НІЧЕПОРУК

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА