

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ВІДДАЛЕНОГО
МОНІТОРИНГУ СТАНУ ҐРУНТУ
SOFTWARE FOR REMOTE MONITORING OF SOIL CONDITIONS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОНОВАЛОВ М.Г.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ О.М.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОНОВАЛОВА Микити Григоровича
 (ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для віддаленого моніторингу стану ґрунту. Software for Remote Monitoring of Soil Conditions.

керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр Олексійович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз технологій для віддаленого моніторингу ґрунту. 2. Проєктування програмного забезпечення. 3. Реалізація системи. 4. Аналіз отриманих результатів.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст.викладач		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Микита КОНОВАЛОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Олександр СТЕПАНЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 117 с., 17 табл., 27 рис., 3 дод., 25 джерел.

СИСТЕМА МОНІТОРИНГУ, ҐРУНТ, ІНТЕРНЕТ РЕЧЕЙ, МІКРОКОНТРОЛЕР, БЕЗДРОТОВА ПЕРЕДАЧА ДАНИХ, ДАТЧИКИ, ESP32, МОБІЛЬНИЙ ДОДАТОК, REACT NATIVE, NODE.JS, MONGODB.

Об'єкт дослідження — процес розробки програмного забезпечення для систем дистанційного збору та обробки даних про стан ґрунту.

Предмет дослідження — методи та програмні засоби для отримання, передачі, обробки та візуалізації даних про стан ґрунту в системах віддаленого моніторингу.

Мета роботи — підвищення ефективності процесу збору та аналізу даних про стан ґрунту шляхом розробки програмного забезпечення для віддаленого моніторингу.

Матеріали, методи та технічні засоби: ємнісний датчик вологості Capacitive Soil Moisture v1.2, датчик температури DS18B20, мікроконтролер ESP32, технології бездротової передачі даних WiFi та LoRaWAN, Node.js, Express, MongoDB Atlas, React Native, Expo, Chart.js.

Результати роботи. Розроблено комплексну систему віддаленого моніторингу стану ґрунту, що включає апаратну частину для збору даних з датчиків, серверну частину для обробки та зберігання інформації та мобільний додаток для візуалізації даних.

Висновки. Розроблена система віддаленого моніторингу стану ґрунту забезпечує ефективне отримання, передачу та аналіз даних про вологість і температуру ґрунту

Галузь використання — сільське господарство, точне землеробство, екологічний моніторинг.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
117 pages, 17 tables, 27 figures, 3 appendixes, 25 sources.

MONITORING SYSTEM, SOIL, INTERNET OF THINGS,
MICROCONTROLLER, WIRELESS DATA TRANSMISSION, SENSORS,
ESP32, MOBILE APPLICATION, REACT NATIVE, NODE.JS, MONGODB.

Object of study — the process of developing software for remote collection and processing systems of soil condition data.

Subject of study — methods and software tools for obtaining, transmitting, processing, and visualizing soil condition data in remote monitoring systems.

The purpose of the work — to increase the efficiency of soil data collection and analysis process through the development of software for remote monitoring.

Materials, methods, and technical tools: Capacitive Soil Moisture v1.2 sensor, DS18B20 temperature sensor, ESP32 microcontroller, WiFi and LoRaWAN wireless data transmission technologies, Node.js, Express, MongoDB Atlas, React Native, Expo, Chart.js.

Results. A comprehensive remote soil condition monitoring system has been developed, including hardware for sensor data collection, a server component for processing and storing information, and a mobile application for data visualization.

Conclusions. The developed remote soil monitoring system provides effective acquisition, transmission, and analysis of soil moisture and temperature data.

Field of application — agriculture, precision farming, environmental monitoring.

ЗМІСТ

	С.
Вступ.....	9
1 Аналіз технологій для віддаленого моніторингу ґрунту	11
1.1 Огляд існуючих систем моніторингу ґрунту	11
1.2 Аналіз датчиків для моніторингу стану ґрунту	13
1.3 Аналіз мікроконтролерів для систем моніторингу ґрунту	16
1.4 Технології бездротової передачі даних для систем моніторингу ґрунту.....	20
1.5 Аналіз серверних технологій для обробки та зберігання даних	23
1.6 Технології розробки мобільних додатків для систем моніторингу ґрунту.....	25
1.7 Обґрунтування вибору технологій для розробки системи віддаленого моніторингу стану ґрунту.....	28
1.8 Висновки до розділу 1	31
2 Проєктування програмного забезпечення	33
2.1 Загальна архітектура системи	33
2.2 Проєктування програмного забезпечення мікроконтролера	36
2.4 Проєктування серверної частини	40
2.5 Проєктування мобільного додатку	44
2.5 Висновки за розділом 2	47
3 Реалізація системи.....	48
3.1 Реалізація програмного забезпечення мікроконтролера.....	48
3.2 Реалізація серверної частини	52
3.3 Реалізація мобільного додатку.....	64
3.4 Висновки за розділом 3	67
4 Аналіз отриманих результатів	69
4.1 Інструкції користувача	69
4.2 Оцінка ефективності системи	77
4.3 Висновки за розділом 4	79
Висновки	80

Перелік джерел посилання	81
Додаток А Технічне завдання	83
Додаток Б Фрагмент тексту програми	88
Додаток В Слайди презентації.....	109

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
BLE	– Bluetooth Low Energy;
GPIO	– General Purpose Input/Output;
IoT	– Internet of Things;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Tokens;
MQTT	– Message Queuing Telemetry Transport;
REST	– Representational State Transfer;
SUS	– System Usability Scale;
WiFi	– Wireless Fidelity.

ВСТУП

В останні роки сільське господарство зазнає трансформації під впливом технологій Інтернету речей (IoT) та точного землеробства. Віддалений моніторинг стану ґрунту стає важливим інструментом для прийняття обґрунтованих рішень щодо управління сільськогосподарськими ресурсами. Актуальність розробки програмного забезпечення для віддаленого моніторингу стану ґрунту зумовлена необхідністю оптимізації використання води та добрив, підвищення врожайності сільськогосподарських культур та зменшення негативного впливу на навколишнє середовище.

Традиційні методи моніторингу ґрунту вимагають значних трудових витрат, часто мають низьку частоту вимірювань та обмежену просторову роздільну здатність. Сучасні інформаційні технології дозволяють автоматизувати збір даних, забезпечити їх централізоване зберігання, обробку та візуалізацію, що робить інформацію про стан ґрунту доступною для користувачів у режимі реального часу. Розробка та впровадження програмно-апаратних комплексів для моніторингу ґрунту сприяє підвищенню ефективності сільськогосподарського виробництва та сталому використанню природних ресурсів.

Об'єкт дослідження — процес розробки програмного забезпечення для систем дистанційного збору та обробки даних про стан ґрунту.

Предмет дослідження — методи та програмні засоби для отримання, передачі, обробки та візуалізації даних про стан ґрунту в системах віддаленого моніторингу.

Мета роботи — підвищення ефективності процесу збору та аналізу даних про стан ґрунту шляхом розробки програмного забезпечення для віддаленого моніторингу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз існуючих технологій для побудови систем віддаленого моніторингу ґрунту та обґрунтувати вибір оптимальних рішень;
- розробити архітектуру програмного забезпечення для системи віддаленого моніторингу стану ґрунту;
- спроектувати та реалізувати програмне забезпечення для мікроконтролера, що забезпечує збір даних з датчиків та їх передачу;
- створити серверну частину для обробки та зберігання даних;
- розробити мобільний додаток для візуалізації даних та сповіщення користувачів;
- провести тестування розробленої системи та оцінити її ефективність.

Практичне значення отриманих результатів полягає в створенні програмно-апаратного комплексу, який дозволяє фермерам та дослідникам отримувати актуальну інформацію про стан ґрунту, що сприяє прийняттю більш обґрунтованих рішень щодо використання води та добрив, підвищенню врожайності сільськогосподарських культур та зниженню негативного впливу на навколишнє середовище.

1 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ГРУНТУ

1.1 Огляд існуючих систем моніторингу ґрунту

Розвиток інформаційних технологій та поширення концепції Інтернету речей (IoT) створили підґрунтя для впровадження сучасних систем моніторингу ґрунту. Такі системи забезпечують точне землеробство, що базується на даних, та дозволяють оптимізувати використання ресурсів у сільському господарстві [1].

Існуючі системи моніторингу ґрунту можна класифікувати за різними критеріями: за способом отримання даних, за типом вимірюваних параметрів, за способом передачі даних та за методами їх обробки. За способом отримання даних системи поділяються на контактні, що використовують датчики, які безпосередньо контактують з ґрунтом, та дистанційні, що використовують супутникові або безпілотні технології для збору даних [2]. Контактні системи, такі як точкові датчики, забезпечують високу точність вимірювань, але мають обмежене просторове охоплення. Дистанційні системи можуть охоплювати великі території, проте часто мають нижчу точність та вищу вартість.

За типом вимірюваних параметрів виділяють системи, що моніторять вологість ґрунту, його температуру, кислотність, електропровідність, вміст поживних речовин та інші характеристики. Комплексні системи здатні одночасно відстежувати кілька параметрів, що дозволяє отримати більш повну картину стану ґрунту [3].

Одним з перших комерційних рішень для віддаленого моніторингу ґрунту стала система Soil Scout, розроблена в Фінляндії. Ця система використовує бездротові підземні датчики, що вимірюють вологість, температуру та електропровідність ґрунту, передаючи дані через базову станцію до хмарного сервера [4]. Система забезпечує автономну роботу датчиків протягом кількох років завдяки оптимізованому енергоспоживанню.

Компанія Sentek Technologies пропонує систему EnviroSCAN, яка використовує ємнісні датчики для вимірювання профілю вологості ґрунту на різних глибинах. Дані передаються через GSM мережу на сервер, де вони обробляються та візуалізуються за допомогою спеціалізованого програмного забезпечення [5]. Це рішення дозволяє фермерам отримати детальну інформацію про розподіл вологи в ґрунті та оптимізувати графік поливу.

Система Плантер від української компанії Kray Technologies представляє собою набір бездротових датчиків, що вимірюють вологість та температуру ґрунту. Дані передаються через LoRaWAN мережу на сервер, а користувачі отримують доступ до інформації через мобільний додаток [6]. Система забезпечує енергоефективність та тривалий термін автономної роботи датчиків.

Порівняльний аналіз існуючих комерційних систем моніторингу ґрунту наведено в табл. 1.1.

Таблиця 1.1 - Порівняння характеристик комерційних систем моніторингу ґрунту

Назва системи	Виробник	Вимірювані параметри	Технологія передачі даних	Автономність	Вартість
Soil Scout	Soil Scout Ltd (Фінляндія)	Вологість, температура, електропровідність	Радіочастотна передача (433 МГц)	До 20 років	Висока
EnviroSCAN	Sentek Technologies (Австралія)	Багаторівневий профіль вологості	GSM/GPRS	До 5 років	Висока
Плантер	Kray Technologies (Україна)	Вологість, температура	LoRaWAN	До 3 років	Середня
AquaCheck	AquaCheck Ltd (ПАР)	Вологість, температура, солоність	GSM/GPRS	До 5 років	Висока

Аналіз існуючих рішень показує, що більшість комерційних систем моніторингу ґрунту мають високу вартість, що обмежує їх доступність для дрібних фермерських господарств. Крім того, багато систем є закритими та не дозволяють інтегрувати обладнання різних виробників або адаптувати програмне забезпечення під конкретні потреби користувачів [7].

Відкриті програмно-апаратні платформи, такі як Arduino та ESP32, дозволяють створювати бюджетні системи моніторингу ґрунту з можливістю адаптації під конкретні потреби [8]. Відкритий код таких систем забезпечує можливість їх модифікації та інтеграції з іншими системами.

Таким чином, розробка доступного програмного забезпечення для віддаленого моніторингу стану ґрунту на базі відкритих платформ залишається актуальним завданням, що має практичне значення для сільського господарства та екологічного моніторингу.

1.2 Аналіз датчиків для моніторингу стану ґрунту

Датчики є основними елементами будь-якої системи моніторингу ґрунту, оскільки саме вони забезпечують отримання первинних даних про стан середовища. Вибір датчиків визначає діапазон параметрів, що вимірюються, точність, енергоспоживання та вартість системи в цілому [9]. Для ефективного моніторингу ґрунту використовуються різні типи датчиків, кожен з яких спеціалізується на вимірюванні певних параметрів.

Датчики вологості ґрунту є найпоширенішим типом сенсорів, що використовуються в системах моніторингу. Вони поділяються на кілька типів залежно від принципу дії: резистивні, ємнісні, тензометричні та нейтронні [10]. Найбільш доступними та простими у використанні є резистивні та ємнісні датчики.

Резистивні датчики вологості ґрунту працюють на принципі вимірювання електричного опору між двома електродами, вставленими в ґрунт. Опір ґрунту змінюється залежно від вмісту води: сухий ґрунт має

високий опір, а вологий - низький. Перевагами таких датчиків є низька вартість та простота підключення до мікроконтролерів, однак вони мають суттєві недоліки: швидка корозія електродів, залежність показань від температури та вмісту солей у ґрунті, а також необхідність калібрування для різних типів ґрунту [11].

Ємнісні датчики вологості вимірюють діелектричну проникність ґрунту, яка змінюється залежно від вмісту води. Ці датчики складаються з двох електродів, що утворюють конденсатор, ємність якого залежить від вологості ґрунту. Порівняно з резистивними, ємнісні датчики мають вищу точність, довший термін служби та менш схильні до впливу солей у ґрунті [12]. Однак, вони мають вищу вартість та складніші у виготовленні.

Для вимірювання температури ґрунту часто використовуються цифрові датчики DS18B20, які забезпечують точність вимірювання $\pm 0,5^{\circ}\text{C}$ в діапазоні від -55°C до $+125^{\circ}\text{C}$. Ці датчики мають кілька переваг: вони використовують однопровідний інтерфейс, що дозволяє підключати кілька датчиків до одного порту мікроконтролера; мають вбудований АЦП, що забезпечує високу точність вимірювань; доступні у водонепроникному виконанні; мають унікальну адресацію, що спрощує ідентифікацію [13].

Датчики кислотності (pH) ґрунту використовуються для визначення рівня кислотності, що є критично важливим параметром для росту рослин. Більшість рослин віддають перевагу нейтральному або слабкокислому середовищу, тому моніторинг pH допомагає оптимізувати використання добрив та покращити умови росту рослин [1]. Датчики pH складаються з електрода, який генерує електричний потенціал, пропорційний концентрації іонів водню в ґрунті. Оскільки електроди pH чутливі до температури, часто такі датчики комбінуються з температурними сенсорами для компенсації температурної похибки.

Для комплексної оцінки родючості ґрунту використовуються датчики електропровідності, які опосередковано свідчать про вміст розчинних солей та поживних речовин. Вони працюють шляхом пропускання електричного

струму через ґрунт та вимірювання опору [5]. Висока електропровідність може свідчити про надмірний вміст солей, що негативно впливає на рослини, або про високий вміст поживних речовин.

Спеціалізовані датчики NPK (азоту, фосфору та калію) дозволяють оцінити вміст основних поживних речовин у ґрунті. Ці датчики використовують іоноселективні електроди або спектроскопічні методи для визначення концентрації елементів [6]. Однак, такі датчики мають досить високу вартість та складні у використанні, що обмежує їх широке впровадження в бюджетних системах моніторингу.

На рис. 1.1 представлено схематичне зображення основних типів датчиків для моніторингу ґрунту та їх взаємодії з мікроконтролером.

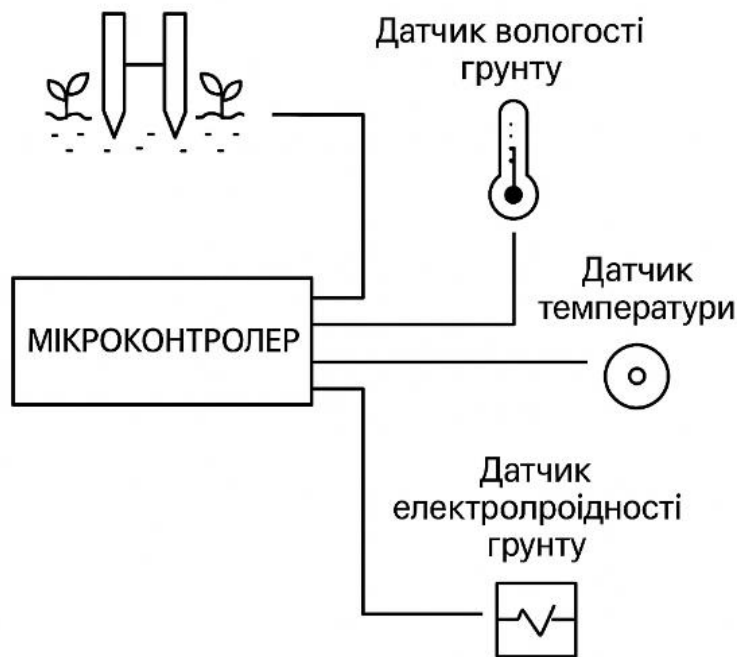


Рисунок 1.1 - Схема підключення датчиків до мікроконтролера в системі моніторингу ґрунту

Для визначення оптимального набору датчиків для системи віддаленого моніторингу ґрунту було проведено порівняльний аналіз найбільш поширених моделей, результати якого представлені в табл. 1.2.

Таблиця 1.2 - Порівняльний аналіз датчиків для моніторингу ґрунту

Тип датчика	Модель	Діапазон вимірювань	Точність	Інтерфейс	Енергоспоживання	Вартість
Вологість (резистивний)	YL-69	0-100%	Низька	Аналоговий	Низьке	Низька
Вологість (ємнісний)	Capacitive Soil Moisture v1.2	0-100%	Середня	Аналоговий	Низьке	Низька
Вологість (професійний)	TEROS 10	0-100%	Висока	SDI-12	Середнє	Висока
Температура	DS18B20	-55°C до +125°C	$\pm 0.5^\circ\text{C}$	1-Wire	Низьке	Низька
pH	SEN0161	pH 0-14	± 0.1 pH	Аналоговий	Середнє	Середня
Електропровідність	DFR0300	0-20 мС/см	$\pm 10\%$	Аналоговий	Середнє	Середня
NPK	SEN0463	N, P, K (5 рівнів)	Середня	UART	Високе	Висока

1.3 Аналіз мікроконтролерів для систем моніторингу ґрунту

Мікроконтролери є центральними елементами систем моніторингу ґрунту, забезпечуючи збір даних з датчиків, їх первинну обробку та передачу на сервер [2]. Вибір відповідного мікроконтролера є критично важливим для забезпечення функціональності, енергоефективності та надійності системи. При аналізі мікроконтролерів для систем моніторингу ґрунту необхідно враховувати їх обчислювальну потужність, енергоспоживання, наявність вбудованих інтерфейсів для підключення датчиків та модулів зв'язку, а також вартість та доступність [1].

Одним з найпопулярніших мікроконтролерів для IoT-пристроїв є ESP32, розроблений компанією Espressif Systems. ESP32 представляє собою систему на чипі (SoC) з двоядерним процесором Tensilica Xtensa LX6, що працює на частоті до 240 МГц [2]. Цей мікроконтролер має вбудовану підтримку WiFi та Bluetooth, що робить його ідеальним для бездротових систем моніторингу. ESP32 також оснащений 512 КБ оперативної пам'яті та до 16 МБ flash-пам'яті, що дозволяє зберігати значний обсяг даних та виконувати складні алгоритми обробки. Особливістю ESP32 є наявність ультра-низького енергоспоживання в режимі глибокого сну, що критично важливо для автономних систем моніторингу [3].

Arduino Nano 33 IoT є компактним мікроконтролером, побудованим на базі процесора SAMD21 Cortex-M0+ з тактовою частотою 48 МГц [4]. Цей мікроконтролер має вбудований модуль WiFi та Bluetooth, що спрощує підключення до мережі. Arduino Nano 33 IoT має 256 КБ flash-пам'яті та 32 КБ оперативної пам'яті, що достатньо для більшості задач моніторингу. Перевагою цього мікроконтролера є його компактний розмір та сумісність з екосистемою Arduino, що спрощує розробку програмного забезпечення [5].

Raspberry Pi Pico W є мікроконтролером на базі чипа RP2040, розробленого компанією Raspberry Pi [6]. Цей мікроконтролер має двоядерний процесор Cortex-M0+ з частотою 133 МГц, 264 КБ оперативної пам'яті та 2 МБ flash-пам'яті. Pico W також оснащений модулем WiFi, що дозволяє підключати його до локальної мережі або Інтернету. Особливістю Pico W є програмована логічна матриця (PIO), яка дозволяє реалізувати власні апаратні інтерфейси для підключення специфічних датчиків [7].

Для оцінки енергоефективності мікроконтролерів було проведено експериментальне дослідження, результати якого представлені в табл. 1.3.

Таблиця 1.3 - Порівняння енергоспоживання мікроконтролерів у різних режимах роботи

Мікроконтролер	Активний режим (мА)	Режим сну (мкА)	Режим глибокого сну (мкА)	Час пробудження (мс)
ESP32	240	20	10	45
Arduino Nano 33 IoT	15	5	1	8
Raspberry Pi Pico W	25	2.4	0.7	15

Окрім енергоспоживання, важливим аспектом є наявність периферійних інтерфейсів для підключення датчиків та модулів зв'язку. В табл. 1.4 представлено порівняння наявності різних інтерфейсів у розглянутих мікроконтролерах.

Таблиця 1.4 - Порівняння наявності периферійних інтерфейсів у мікроконтролерах

Інтерфейс	ESP32	Arduino Nano 33 IoT	Raspberry Pi Pico W
GPIO	34	14	26
ADC	12-біт, 18 каналів	12-біт, 8 каналів	12-біт, 4 канали
DAC	8-біт, 2 канали	10-біт, 1 канал	Немає
I2C	2	1	2
SPI	3	1	2
UART	3	1	2
WiFi	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n
Bluetooth	BLE 4.2	BLE 4.2	Немає

Програмування мікроконтролерів здійснюється за допомогою спеціалізованих середовищ розробки. ESP32 може програмуватися за допомогою ESP-IDF (офіційного фреймворку від Espressif), Arduino IDE або

PlatformIO [8]. Arduino Nano 33 IoT програмується через Arduino IDE або PlatformIO [9]. Raspberry Pi Pico W підтримує програмування на C/C++, MicroPython та CircuitPython [3].

Для систем моніторингу ґрунту особливо важливим є наявність режимів енергозбереження, які дозволяють значно подовжити час автономної роботи. ESP32 має кілька режимів сну: Light Sleep, Deep Sleep та Hibernation. У режимі Deep Sleep мікроконтролер споживає лише 10 мкА, а пробудження може здійснюватися за таймером або зовнішнім сигналом [1]. Arduino Nano 33 IoT також підтримує режими сну, але має менше можливостей для конфігурації енергозбереження. Raspberry Pi Pico W має режими сну з низьким енергоспоживанням, але програмування цих режимів вимагає більше зусиль порівняно з ESP32 [2].

Вартість мікроконтролерів є важливим фактором при розробці бюджетних систем моніторингу. ESP32 має найкраще співвідношення ціна/функціональність серед розглянутих мікроконтролерів [3]. Arduino Nano 33 IoT має вищу вартість, але простіший у програмуванні. Raspberry Pi Pico W має найнижчу вартість, але вимагає більше зусиль для розробки системи енергозбереження.

На основі проведеного аналізу можна зробити висновок, що для систем віддаленого моніторингу ґрунту оптимальним вибором є мікроконтролер ESP32, який забезпечує необхідну функціональність, має вбудовані модулі WiFi та Bluetooth, підтримує ефективні режими енергозбереження та має доступну вартість. ESP32 також має велику спільноту розробників та багато готових бібліотек, що спрощує розробку програмного забезпечення [4].

Обраний мікроконтролер буде використовуватися в подальшій розробці системи віддаленого моніторингу стану ґрунту, забезпечуючи збір даних з датчиків, їх первинну обробку та передачу на сервер.

1.4 Технології бездротової передачі даних для систем моніторингу ґрунту

Бездротові технології передачі даних є важливим компонентом систем віддаленого моніторингу ґрунту, оскільки вони забезпечують комунікацію між датчиками та центральним сервером [5]. Вибір оптимальної технології бездротового зв'язку залежить від багатьох факторів, таких як дальність передачі, енергоспоживання, пропускна здатність, вартість та доступність інфраструктури [6]. У контексті систем моніторингу ґрунту особливо важливими характеристиками є енергоефективність та дальність дії, оскільки датчики часто розміщуються у віддалених місцях без доступу до стаціонарного електроживлення.

WiFi (IEEE 802.11) є однією з найпоширеніших технологій бездротового зв'язку, що забезпечує високу швидкість передачі даних (до 600 Мбіт/с для стандарту 802.11n) на відстань до 100 метрів у відкритому просторі. Перевагами WiFi є широка доступність обладнання, простота налаштування та висока пропускна здатність. Однак, WiFi має порівняно високе енергоспоживання, що обмежує його використання в автономних системах моніторингу. Для подолання цього обмеження використовуються спеціальні режими енергозбереження, такі як глибокий сон з періодичним пробудженням для передачі даних.

Bluetooth Low Energy (BLE) є енергоефективною версією стандарту Bluetooth, спеціально розробленою для IoT-пристроїв [9]. BLE забезпечує передачу даних на відстань до 100 метрів з швидкістю до 2 Мбіт/с (для Bluetooth 5.0). Перевагою BLE є дуже низьке енергоспоживання, що дозволяє пристроям працювати від батареї протягом кількох років. Однак, обмежена дальність дії робить BLE придатним лише для систем з малою площею покриття або для локального зчитування даних за допомогою мобільного пристрою.

LoRaWAN (Long Range Wide Area Network) є технологією з низьким енергоспоживанням та великою дальністю дії, що робить її особливо придатною для сільськогосподарських застосувань. LoRaWAN забезпечує передачу даних на відстань до 15 км у сільській місцевості та до 5 км у міських умовах з швидкістю від 0.3 до 50 кбіт/с. Ключовою перевагою LoRaWAN є надзвичайно низьке енергоспоживання, що дозволяє пристроям працювати від однієї батареї до 10 років. Крім того, існує безкоштовна мережа The Things Network, що надає доступ до інфраструктури LoRaWAN у багатьох регіонах світу.

Для більш детального порівняння технологій бездротової передачі даних було проведено аналіз їх основних характеристик, результати якого представлені в табл. 1.5.

Таблиця 1.5 - Порівняння характеристик технологій бездротової передачі даних

Технологія	Дальність дії	Швидкість передачі	Енергоспоживання	Частота	Доступність інфраструктури	Вартість
WiFi (802.11n)	До 100 м	До 600 Мбіт/с	Високе	2.4/5 ГГц	Висока в населених пунктах	Низька
Bluetooth 5.0 (BLE)	До 100 м	До 2 Мбіт/с	Дуже низьке	2.4 ГГц	Потребує шлюз	Низька
LoRaWAN	До 15 км	0.3-50 кбіт/с	Наднизьке	868/915 МГц	Зростаюча (TTN)	Середня
Zigbee	До 100 м	250 кбіт/с	Низьке	2.4 ГГц	Потребує шлюз	Середня
NB-IoT	До 10 км	До 200 кбіт/с	Низьке	Стільникові частоти	Залежить від оператора	Висока
Sigfox	До 10 км	100 біт/с	Наднизьке	868/915 МГц	Обмежена	Висока

При виборі технології бездротової передачі даних для систем моніторингу ґрунту необхідно враховувати специфіку сільськогосподарських умов експлуатації. В сільській місцевості часто відсутня розвинена інфраструктура зв'язку, а датчики можуть бути розміщені на значній відстані один від одного. Тому технології з великою дальністю дії, такі як LoRaWAN, мають перевагу в таких умовах.

Протокол MQTT (Message Queuing Telemetry Transport) є стандартним протоколом прикладного рівня для IoT-пристроїв, що забезпечує легку та надійну передачу даних за моделлю "публікація-підписка". MQTT особливо ефективний для пристроїв з обмеженими ресурсами, оскільки має мінімальні накладні витрати на передачу даних. Для роботи MQTT потрібен брокер, що служить посередником між видавцями (датчиками) та підписниками (клієнтськими додатками). Найпопулярнішими MQTT-брокерами є Mosquitto, HiveMQ та EMQ X.

Енергоефективність є критичним фактором для автономних систем моніторингу ґрунту. Для мінімізації енергоспоживання використовуються різні стратегії:

- режими сну: пристрої більшість часу проводять у режимі глибокого сну, пробуджуючись лише для зняття показань та передачі даних;
- агрегація даних: замість передачі кожного виміру окремо, дані агрегуються та передаються пакетами;
- адаптивна частота вимірювань: частота зняття показань та передачі даних адаптується залежно від зміни параметрів ґрунту.

Вибір оптимальної технології бездротової передачі даних залежить від конкретних вимог проєкту. Для систем з малою площею покриття та можливістю регулярного обслуговування WiFi або BLE можуть бути оптимальним вибором. Для систем, що охоплюють великі сільськогосподарські угіддя, LoRaWAN з його великою дальністю дії та низьким енергоспоживанням є більш прийнятним.

На основі проведеного аналізу для розроблюваної системи віддаленого моніторингу ґрунту пропонується використовувати комбінацію технологій: WiFi для локацій з доступом до інфраструктури та LoRaWAN для віддалених місць. Такий підхід забезпечить гнучкість та масштабованість системи, дозволяючи використовувати її в різних умовах експлуатації.

1.5 Аналіз серверних технологій для обробки та зберігання даних

Серверна частина є важливим компонентом системи віддаленого моніторингу ґрунту, забезпечуючи збір, обробку, зберігання та надання доступу до даних. Вибір оптимальних серверних технологій має критичне значення для створення надійної, масштабованої та економічно ефективної системи. У контексті розробки бюджетного рішення особливу увагу слід приділити безкоштовним або умовно-безкоштовним технологіям, які при цьому забезпечують необхідну функціональність.

Для розробки серверної частини систем моніторингу можуть використовуватися різні мови програмування та фреймворки. Найпопулярнішими є Node.js та Python з фреймворками Express та Flask/Django відповідно. Node.js має перевагу в швидкості обробки асинхронних запитів, що робить його ефективним для систем реального часу з великою кількістю одночасних підключень. Python, з іншого боку, має більш простий синтаксис та широкий набір бібліотек для наукових обчислень та аналізу даних, що може бути корисним для обробки даних моніторингу ґрунту.

Для зберігання даних моніторингу можуть використовуватися різні типи баз даних: реляційні (PostgreSQL, MySQL) та нереляційні (MongoDB, InfluxDB). Реляційні бази даних забезпечують структуровану схему даних та підтримують складні запити з використанням SQL. Нереляційні бази даних, особливо часові бази даних, такі як InfluxDB, оптимізовані для роботи з

часовими рядами, що робить їх особливо ефективними для зберігання даних моніторингу.

MongoDB Atlas та PostgreSQL мають безкоштовні плани, що робить їх доступними для бюджетних проєктів. MongoDB Atlas надає 512 МБ безкоштовного сховища та базові функції моніторингу, що достатньо для невеликих систем моніторингу ґрунту. PostgreSQL можна розгорнути на безкоштовних хостингах, таких як Heroku або Railway, які надають обмежені, але достатні для початкового розгортання ресурси.

Для забезпечення комунікації між пристроями IoT та сервером часто використовується протокол MQTT. Mosquitto є відкритим та безкоштовним MQTT-брокером, що забезпечує надійну передачу повідомлень між компонентами системи. Для систем з більшою кількістю пристроїв та повідомлень можуть використовуватися хмарні MQTT-брокери, такі як HiveMQ Cloud або CloudMQTT, які мають обмежені безкоштовні плани.

Таблиця 1.6 - Порівняння серверних технологій для систем моніторингу ґрунту

Технологія	Тип	Переваги	Недоліки	Ліміти безкоштовного плану
1	2	3	4	5
Node.js (Express)	Серверний фреймворк	Асинхронність, швидкість, простота	Менше бібліотек для аналізу даних	Безкоштовний
Python (Flask)	Серверний фреймворк	Простий синтаксис, наукові бібліотеки	Нижча продуктивність для асинхронних операцій	Безкоштовний
MongoDB Atlas	База даних	Гнучка схема, простота використання	Обмежені можливості для складних запитів	512 МБ, 100 з'єднань

Продовження таблиці 1.6

1	2	3	4	5
PostgreSQL	База даних	Потужна SQL-підтримка, надійність	Складніше налаштування	Залежить від хостингу
Mosquitto	MQTT брокер	Легкий, низькі вимоги до ресурсів	Обмежена масштабованість	Безліміт
Firebase Realtime DB	База даних + API	Простота інтеграції, реальний час	Обмежені можливості запитів	1 ГБ сховища, 10 ГБ трафіку
Heroku	Хостинг	Простота розгортання, інтеграція з Git	Сплячий режим для безкоштовних додатків	550-1000 годин роботи на місяць
Netlify	Хостинг	Безперервна інтеграція, CDN	Обмежені обчислювальні ресурси	100 ГБ трафіку, 300 хвилин сборки
Vercel	Хостинг	Оптимізація для React/Next.js, глобальний CDN	Обмежені типи серверних функцій	100 ГБ трафіку, обмеження на виклики функцій

При проєктуванні API для системи моніторингу ґрунту важливо забезпечити простоту використання та розширюваність. REST (Representational State Transfer) є найпоширенішим архітектурним стилем для створення API [58]. REST API використовує стандартні HTTP-методи (GET, POST, PUT, DELETE) для взаємодії з ресурсами, що робить його інтуїтивно зрозумілим та простим у використанні.

1.6 Технології розробки мобільних додатків для систем моніторингу ґрунту

Мобільний додаток є важливим компонентом системи віддаленого моніторингу ґрунту, забезпечуючи користувачам зручний доступ до даних,

візуалізацію інформації та управління системою [12]. Вибір технологій для розробки мобільного додатку впливає на швидкість розробки, функціональність, продуктивність та користувацький досвід.

При розробці мобільних додатків можна використовувати нативний або кросплатформний підхід [13]. Нативна розробка передбачає створення окремих додатків для кожної платформи з використанням специфічних мов та інструментів: Swift/Objective-C для iOS та Kotlin/Java для Android. Кросплатформна розробка дозволяє створити один додаток, який працює на різних платформах, використовуючи фреймворки, такі як React Native, Flutter або Xamarin.

React Native є популярним фреймворком для кросплатформної розробки, розробленим компанією Facebook [14]. Він дозволяє створювати мобільні додатки з використанням JavaScript та React, забезпечуючи нативний вигляд та продуктивність. React Native має велику спільноту розробників, багато готових компонентів та бібліотек, що спрощує розробку. Однією з ключових переваг React Native є "гарячий перезапуск", який дозволяє побачити зміни в коді миттєво, без перекомпіляції додатку [15].

Flutter, розроблений Google, є іншим популярним фреймворком для кросплатформної розробки, що використовує мову Dart [16]. Flutter відрізняється від React Native тим, що він компілює код в нативний машинний код, а не використовує "міст" для взаємодії з нативними компонентами. Це забезпечує вищу продуктивність та більш послідовний вигляд додатку на різних платформах [17]. Flutter також має багатий набір віджетів, що дозволяє створювати красиві та функціональні інтерфейси.

Порівняльний аналіз технологій розробки мобільних додатків представлено в табл. 1.7.

Для спрощення розробки з використанням React Native можна використовувати платформу Ехро [18]. Ехро надає набір інструментів та сервісів, що спрощують розробку, тестування та розгортання React Native додатків. Особливістю Ехро є можливість розробки без необхідності

налаштування нативного середовища та можливість тестування додатку на фізичному пристрої через Expo Client [19]. Безкоштовний план Expo включає всі основні функції, необхідні для розробки мобільного додатку для системи моніторингу ґрунту.

Таблиця 1.7 - Порівняння технологій розробки мобільних додатків

Технологія	Мова програмування	Продуктивність	Інтерфейс користувача	Швидкість розробки	Спільнота та підтримка	Розмір додатку
React Native	JavaScript	Висока	Нативний вигляд	Висока	Велика спільнота, багато бібліотек	Середній
Flutter	Dart	Дуже висока	Кастомні віджети	Висока	Зростаюча спільнота, офіційна підтримка Google	Середній
Expo (на базі React Native)	JavaScript	Висока	Нативний вигляд	Дуже висока	Підтримка Expo, доступ до спільноти React Native	Великий
Kotlin (Android)	Kotlin	Найвища	Повний доступ до нативних компонентів	Середня	Офіційна підтримка Google	Малий
Swift (iOS)	Swift	Найвища	Повний доступ	Середня	Офіційна підтримка Apple	Малий

1.7 Обґрунтування вибору технологій для розробки системи віддаленого моніторингу стану ґрунту

На основі проведеного аналізу технологій для систем віддаленого моніторингу стану ґрунту в попередніх підрозділах необхідно обґрунтувати вибір компонентів для розроблюваної системи. Вибір технологій повинен забезпечити оптимальне співвідношення функціональності, надійності, енергоефективності та вартості, враховуючи обмеження бакалаврської роботи та необхідність використання безкоштовних рішень [3].

Для апаратної частини системи, враховуючи дані з табл. 1.2 та 1.3, оптимальним вибором датчиків є ємнісний датчик вологості ґрунту Capacitive Soil Moisture v1.2 та цифровий датчик температури DS18B20. Ємнісний датчик вологості забезпечує достатню точність вимірювань, має значно довший термін служби порівняно з резистивними аналогами завдяки стійкості до корозії та низьку вартість [9]. Датчик температури DS18B20 вирізняється високою точністю ($\pm 0.5^{\circ}\text{C}$), водонепроникним корпусом, низьким енергоспоживанням та можливістю підключення кількох датчиків до одного порту мікроконтролера завдяки унікальній адресації кожного пристрою [13].

В якості мікроконтролера обрано ESP32, який демонструє найкраще співвідношення функціональності та вартості серед розглянутих варіантів. ESP32 має вбудовану підтримку WiFi та Bluetooth, достатню обчислювальну потужність для обробки даних з датчиків, ефективні режими енергозбереження та велику спільноту розробників [23]. Важливою перевагою ESP32 є можливість роботи в режимі глибокого сну з енергоспоживанням лише 10 мкА, що дозволяє значно подовжити час автономної роботи системи від батареї [3].

Для бездротової передачі даних обрано комбінований підхід з використанням WiFi для локацій з доступом до інфраструктури та LoRaWAN для віддалених місць. WiFi забезпечує високу швидкість передачі даних та

простоту налаштування, але має обмежену дальність дії та високе енергоспоживання. LoRaWAN, з іншого боку, характеризується надзвичайно низьким енергоспоживанням та великою дальністю дії (до 15 км у сільській місцевості), що робить його ідеальним для сільськогосподарських застосувань. Використання безкоштовної мережі The Things Network для LoRaWAN додатково знижує вартість експлуатації системи.

На рис. 1.2 представлена загальна архітектура розроблюваної системи віддаленого моніторингу стану ґрунту.

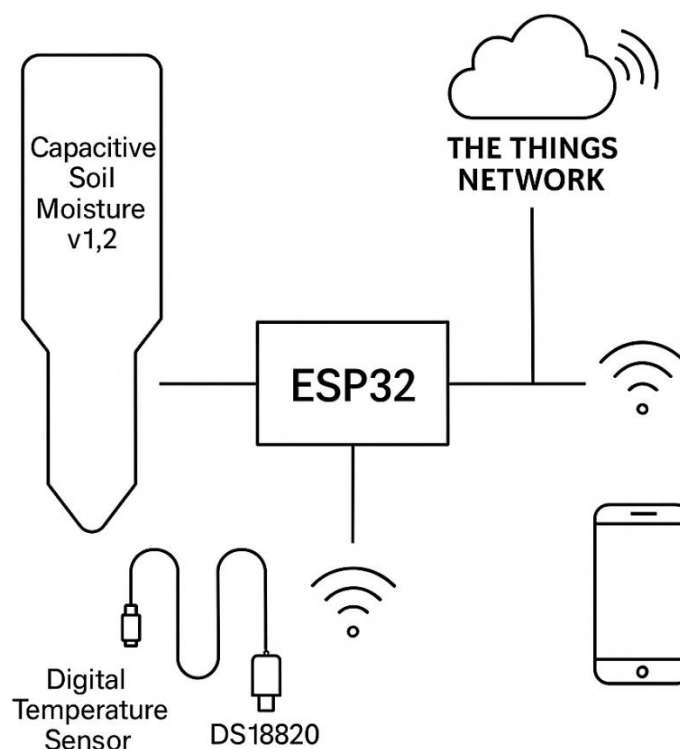


Рисунок 1.2 - Загальна архітектура розроблюваної системи віддаленого моніторингу стану ґрунту

Для серверної частини системи обрано Node.js з фреймворком Express для розробки API, MongoDB Atlas для зберігання даних та Mosquitto як MQTT-брокер (табл. 1.6). Node.js забезпечує високу продуктивність при обробці асинхронних запитів, що критично важливо для систем реального часу з великою кількістю одночасних підключень. MongoDB Atlas надає 512 МБ безкоштовного сховища та базові функції моніторингу, що достатньо для

невеликої системи моніторингу ґрунту. Mosquitto є легким, відкритим та безкоштовним MQTT-брокером, що забезпечує надійну передачу повідомлень між компонентами системи з мінімальними вимогами до ресурсів.

Для розгортання серверної частини обрано платформу Netlify, яка надає безкоштовний хостинг для статичних сайтів та серверних функцій з лімітом 100 ГБ трафіку на місяць. Це дозволяє розгорнути веб-інтерфейс для доступу до даних моніторингу без додаткових витрат.

Для розробки мобільного додатку обрано React Native з Expo та Chart.js для візуалізації даних (табл. 1.7). React Native забезпечує кросплатформність, нативний вигляд додатку та високу швидкість розробки завдяки використанню JavaScript та великій кількості готових компонентів. Expo спрощує процес розробки, тестування та розгортання React Native додатків, надаючи багато готових рішень для типових задач. Chart.js забезпечує простий спосіб створення інтерактивних графіків для візуалізації даних моніторингу ґрунту.

Для взаємодії мобільного додатку з сервером обрано комбінацію REST API для стандартних операцій та WebSockets для отримання даних в реальному часі. Це забезпечує оптимальне співвідношення простоти реалізації та функціональності.

Для аутентифікації користувачів та системи сповіщень обрано Firebase, який надає безкоштовний план з 1 ГБ сховища даних, аутентифікацією та push-повідомленнями. Firebase забезпечує простоту інтеграції з React Native та надійну інфраструктуру для критичних функцій системи.

У табл. 1.8 представлено підсумковий перелік обраних технологій для розробки системи віддаленого моніторингу стану ґрунту.

Таблиця 1.8 - Обрані технології для розробки системи віддаленого моніторингу стану ґрунту

Компонент системи	Обрана технологія	Обґрунтування вибору
Датчик вологості	Capacitive Soil Moisture v1.2	Достатня точність, стійкість до корозії, низька вартість
Датчик температури	DS18B20	Висока точність, водонепроникний корпус, низьке енергоспоживання
Мікроконтролер	ESP32	Вбудована підтримка WiFi та BLE, ефективні режими енергозбереження, велика спільнота
Технологія передачі даних	WiFi / LoRaWAN	Комбінований підхід для різних умов експлуатації
Серверна платформа	Node.js + Express	Висока продуктивність для асинхронних запитів, простота розробки
База даних	MongoDB Atlas	Гнучка схема, безкоштовний план 512 МБ
MQTT брокер	Mosquitto	Відкритий код, низькі вимоги до ресурсів
Хостинг	Netlify	Безкоштовний план 100 ГБ трафіку, проста інтеграція з Git
Фреймворк для мобільного додатку	React Native + Expo	Кросплатформність, швидкість розробки, велика спільнота
Візуалізація даних	Chart.js	Простий API, підтримка різних типів графіків
Аутентифікація та сповіщення	Firebase	Простота інтеграції, безкоштовний план, надійна інфраструктура

1.8 Висновки до розділу 1

У даному розділі було проведено комплексний аналіз існуючих рішень та технологій для розробки системи віддаленого моніторингу стану ґрунту. Розглянуто різні аспекти проєктування такої системи, від апаратної частини до програмного забезпечення.

В результаті аналізу було визначено оптимальні компоненти системи, що забезпечують необхідну функціональність при збереженні доступної вартості. Для апаратної частини обрано ємнісний датчик вологості Capacitive

Soil Moisture v1.2 та цифровий датчик температури DS18B20, які забезпечують достатню точність вимірювань при низькій вартості. В якості керуючого пристрою обрано мікроконтролер ESP32, який відрізняється оптимальним співвідношенням функціональності, енергоефективності та вартості.

Для бездротової передачі даних запропоновано комбінований підхід з використанням WiFi для локацій з доступом до інфраструктури та LoRaWAN для віддалених сільськогосподарських угідь, що забезпечує гнучкість системи при різних умовах експлуатації.

На серверній стороні обрано технологічний стек на основі Node.js, Express, MongoDB Atlas та Mosquitto MQTT-брокера, що надає можливість створити надійну та масштабовану систему зберігання та обробки даних в рамках безкоштовних планів хостингу.

Для забезпечення зручного доступу користувачів до даних моніторингу запропоновано розробку кросплатформного мобільного додатку з використанням React Native та Expo, що дозволяє швидко створити функціональний інтерфейс з візуалізацією даних для різних мобільних платформ.

Запропонована архітектура системи відповідає сучасним тенденціям розвитку IoT-рішень та враховує специфіку сільськогосподарських застосувань, забезпечуючи автономну роботу, енергоефективність та доступність для невеликих фермерських господарств.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальна архітектура системи

Розробка архітектури системи віддаленого моніторингу стану ґрунту є критично важливим етапом, який визначає структуру взаємодії всіх компонентів та впливає на надійність, масштабованість та ефективність роботи системи в цілому. На основі обраних технологій у розділі 1 розроблено комплексну архітектуру системи, яка включає апаратну частину, програмне забезпечення мікроконтролера, серверну частину та мобільний додаток [1].

Архітектура розроблюваної системи базується на багаторівневому підході, який забезпечує розділення відповідальності між компонентами та спрощує їх розробку, тестування та модифікацію. Система складається з наступних рівнів: рівень сенсорів, рівень обробки даних, комунікаційний рівень, серверний рівень та рівень інтерфейсу користувача. Кожен рівень виконує певний набір функцій та взаємодіє з іншими рівнями через чітко визначені інтерфейси.

Рівень сенсорів складається з датчиків вологості та температури ґрунту, які безпосередньо взаємодіють з навколишнім середовищем та перетворюють фізичні величини на електричні сигнали. Ємнісний датчик вологості ґрунту Capacitive Soil Moisture v1.2 підключається до аналогового входу мікроконтролера та забезпечує вимірювання відносної вологості ґрунту. Цифровий датчик температури DS18B20 підключається до цифрового порту мікроконтролера через однопровідний інтерфейс 1-Wire та надає значення температури ґрунту з точністю до 0.5°C.

Рівень обробки даних представлений мікроконтролером ESP32, який виконує наступні функції: збір даних з датчиків, їх первинна обробка (фільтрація, калібрування), тимчасове зберігання, управління енергоспоживанням та передача даних на сервер. ESP32 працює під

управлінням спеціалізованого програмного забезпечення, розробленого з використанням Arduino IDE або ESP-IDF.

Комунікаційний рівень забезпечує передачу даних від мікроконтролера на сервер з використанням комбінованого підходу: WiFi для локацій з доступом до інфраструктури та LoRaWAN для віддалених місць. WiFi-модуль, вбудований в ESP32, забезпечує пряме підключення до локальної мережі або Інтернету. Для LoRaWAN використовується зовнішній модуль RFM95, який підключається до ESP32 через інтерфейс SPI та забезпечує зв'язок з мережею The Things Network.

Серверний рівень складається з MQTT-брокера Mosquitto, який отримує дані від пристроїв через протокол MQTT, API на базі Node.js та Express, який обробляє запити від клієнтських додатків, та бази даних MongoDB Atlas для зберігання даних моніторингу. Серверна частина розгортається на хостинг-платформі Netlify з використанням безкоштовного плану.

Рівень інтерфейсу користувача представлений мобільним додатком, розробленим з використанням React Native та Expo, який забезпечує візуалізацію даних моніторингу, налаштування системи та отримання сповіщень про критичні зміни параметрів ґрунту. Мобільний додаток взаємодіє з серверною частиною через REST API та WebSockets для отримання даних в реальному часі.

Модель даних системи включає наступні основні сутності: пристрої (devices), датчики (sensors), вимірювання (measurements), користувачі (users) та сповіщення (alerts). Структура бази даних MongoDB представлена у вигляді колекцій документів, що відповідають цим сутностям. Схема бази даних представлена на рис. 2.1.

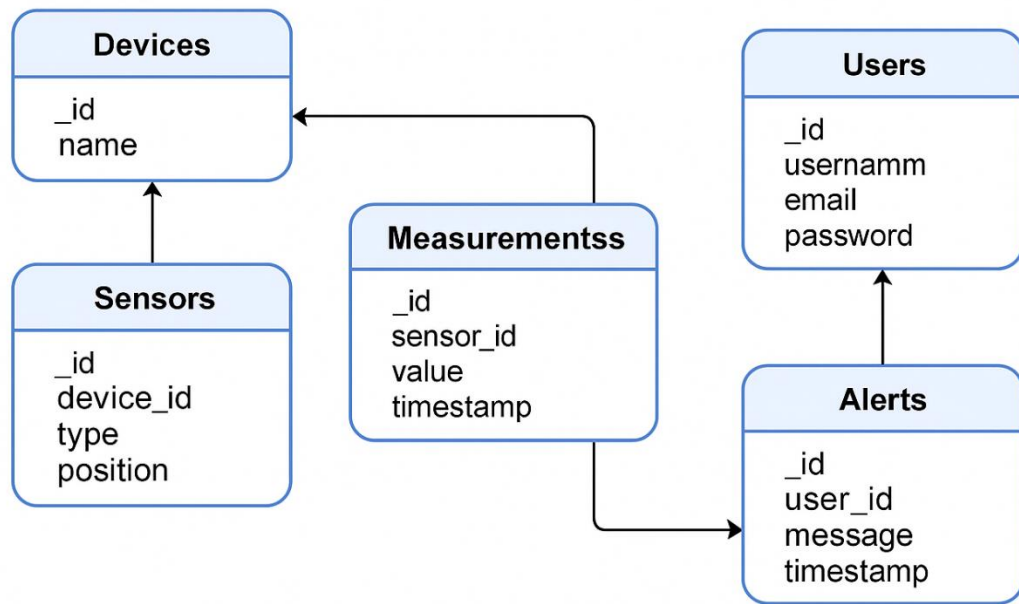


Рисунок 2.1 - Схема бази даних системи віддаленого моніторингу стану ґрунту

Для забезпечення взаємодії між компонентами системи використовуються наступні протоколи та інтерфейси:

- між датчиками та мікроконтролером: аналоговий інтерфейс для ємнісного датчика вологості та 1-Wire для цифрового датчика температури;
- між мікроконтролером та MQTT-брокером: протокол MQTT через WiFi або LoRaWAN;
- між MQTT-брокером та серверним API: внутрішня інтеграція через програмні інтерфейси;
- між серверним API та базою даних: MongoDB Node.js Driver;
- між мобільним додатком та серверним API: REST API для стандартних операцій та WebSockets для даних реального часу [8].

Протокол MQTT використовується для передачі даних від пристроїв до сервера завдяки його низьким накладним витратам на передачу та підтримці якості обслуговування (QoS), що забезпечує надійну доставку повідомлень навіть при нестабільному з'єднанні. Повідомлення MQTT

структуровані за допомогою тем (topics), які організовані ієрархічно, наприклад, `devices/{device_id}/sensors/{sensor_id}/measurements` [9].

REST API забезпечує стандартизований інтерфейс для взаємодії з системою, використовуючи HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій з ресурсами. API надає наступні основні ендпоінти: `/devices`, `/sensors`, `/measurements`, `/users`, `/alerts`. Кожен ендпоінт підтримує відповідні операції CRUD (Create, Read, Update, Delete) [10].

WebSockets використовуються для надсилання даних в реальному часі від сервера до мобільного додатку, забезпечуючи миттєве оновлення інтерфейсу при надходженні нових вимірювань або сповіщень. Це дозволяє уникнути постійного опитування сервера та знижує навантаження на мережу [11].

2.2 Проєктування програмного забезпечення мікроконтролера

Програмне забезпечення мікроконтролера є ключовим компонентом системи віддаленого моніторингу стану ґрунту, який забезпечує збір даних з датчиків, їх обробку, зберігання та передачу на сервер. Правильне проєктування цього компонента має вирішальне значення для забезпечення надійності, енергоефективності та функціональності системи в цілому [29].

Для розробки програмного забезпечення мікроконтролера ESP32 обрано платформу Arduino IDE, яка забезпечує простий та зрозумілий інтерфейс для програмування, а також має велику бібліотеку готових компонентів для роботи з різними периферійними пристроями. Код програми написаний мовою C++ з використанням об'єктно-орієнтованого підходу для забезпечення модульності та перевикористання коду [30].

На рис. 2.2 представлена загальна структура програмного забезпечення мікроконтролера.

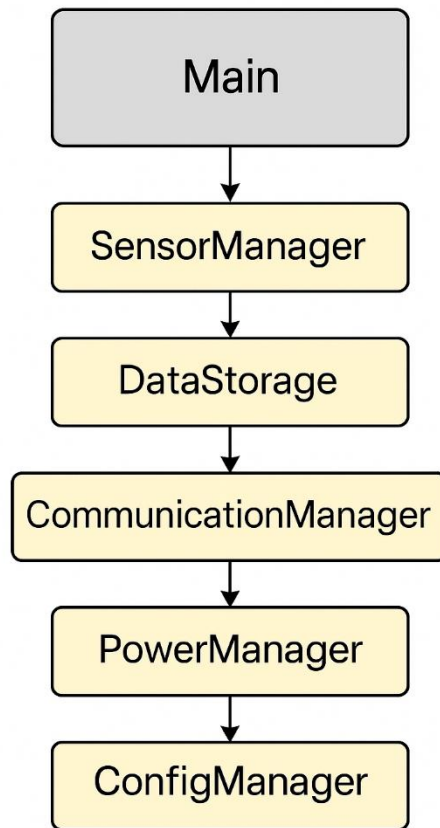


Рисунок 2.2 - Структура програмного забезпечення мікроконтролера

Програмне забезпечення мікроконтролера складається з наступних основних модулів:

- модуль збору даних з датчиків (SensorManager) - відповідає за ініціалізацію датчиків, отримання та первинну обробку даних;
- модуль зберігання даних (DataStorage) - забезпечує тимчасове зберігання даних у внутрішній пам'яті мікроконтролера;
- модуль комунікації (CommunicationManager) - відповідає за передачу даних на сервер через WiFi або LoRaWAN;
- модуль управління енергоспоживанням (PowerManager) - забезпечує ефективне використання енергії та управління режимами роботи мікроконтролера;
- модуль конфігурації (ConfigManager) - відповідає за зберігання та оновлення налаштувань системи;

– головний модуль (Main) - координує роботу всіх інших модулів та визначає логіку роботи системи.

Для реалізації модуля збору даних з датчиків використовуються спеціалізовані бібліотеки: OneWire та DallasTemperature для роботи з цифровим датчиком температури DS18B20, а також власний код для роботи з ємнісним датчиком вологості. Модуль забезпечує ініціалізацію датчиків, отримання вимірювань, їх фільтрацію для усунення шумів та перетворення у фізичні величини (відсотки вологості та градуси Цельсія).

Алгоритм збору даних з датчиків включає наступні кроки:

- ініціалізація датчиків при запуску системи;
- очікування стабілізації датчиків (особливо важливо для ємнісного датчика вологості);
- виконання кількох послідовних вимірювань для кожного датчика;
- фільтрація отриманих даних для зменшення впливу шумів (наприклад, за допомогою медіанного фільтру);
- перетворення сирих даних у фізичні величини з використанням калібрувальних коефіцієнтів;
- передача оброблених даних модулю зберігання.

На рис. 2.3 представлений процес збору даних з датчиків.

Модуль зберігання даних використовує внутрішню flash-пам'ять ESP32 для тимчасового зберігання вимірювань у випадку, якщо передача даних на сервер неможлива (наприклад, через відсутність зв'язку). Для ефективного використання пам'яті дані зберігаються у компактному бінарному форматі. При відновленні зв'язку збережені дані передаються на сервер і видаляються з локальної пам'яті.

Модуль комунікації відповідає за передачу даних на сервер через WiFi або LoRaWAN, залежно від доступності мереж та налаштувань системи. Для роботи з WiFi використовується стандартна бібліотека WiFi.h, а для LoRaWAN - бібліотека LMIC. Модуль забезпечує підключення до мережі, формування повідомлень у форматі JSON для WiFi або компактному

бінарному форматі для LoRaWAN, та їх передачу на сервер з використанням протоколу MQTT або LoRaWAN відповідно.



Рисунок 2.3 - Процес збору даних з датчиків

Алгоритм передачі даних на сервер включає наступні кроки:

- перевірка доступності мереж (WiFi або LoRaWAN);
- вибір оптимального каналу зв'язку на основі пріоритетів та доступності;
- підключення до вибраної мережі;

- формування повідомлення у відповідному форматі;
- передача повідомлення на сервер;
- очікування підтвердження доставки (для WiFi/MQTT);
- повторна спроба передачі у випадку невдачі.

2.4 Проєктування серверної частини

Серверна частина є важливим компонентом системи віддаленого моніторингу стану ґрунту, який забезпечує прийом, обробку, зберігання даних та взаємодію з клієнтськими додатками. Правильне проєктування серверної частини має критичне значення для забезпечення масштабованості, продуктивності та надійності системи в цілому [1].

Для розробки серверної частини обрано стек технологій на основі Node.js, Express, MongoDB Atlas та Mosquitto. Цей вибір забезпечує оптимальне співвідношення продуктивності, гнучкості та простоти розробки, дозволяючи створити ефективну систему в рамках обмежень бакалаврської роботи.

Серверна частина побудована за багатошаровою архітектурою, яка включає наступні рівні:

- рівень комунікації з пристроями - відповідає за прийом даних від пристроїв через MQTT або LoRaWAN;
- рівень API - забезпечує взаємодію з клієнтськими додатками через REST API та WebSockets;
- рівень бізнес-логіки - реалізує основну логіку обробки даних та управління системою;
- рівень доступу до даних - відповідає за взаємодію з базою даних MongoDB Atlas [3].

MQTT-брокер Mosquitto є центральним компонентом для комунікації з пристроями, забезпечуючи надійний обмін повідомленнями за моделлю "видавець-підписник". Повідомлення організовані за ієрархічними темами,

що дозволяє ефективно розділяти та маршрутизувати дані від різних пристроїв. Основна структура тем має вигляд: `devices/{device_id}/sensors/{sensor_id}/measurements` [4].

Для забезпечення надійної передачі даних через MQTT використовується механізм якості обслуговування (QoS), який гарантує доставку повідомлень навіть при нестабільному з'єднанні. В системі використовується QoS рівня 1 (At least once), який забезпечує гарантовану доставку повідомлень, але може призвести до дублювання, яке обробляється на рівні бізнес-логіки [5].

REST API забезпечує стандартизований інтерфейс для взаємодії з клієнтськими додатками. API побудовано за принципами RESTful, використовуючи HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій з ресурсами. API надає наступні основні ендпоінти, описані в табл. 2.1.

Таблиця 2.1 - Основні ендпоінти REST API

Ендпоінт	Метод	Опис	Параметри запити	Відповідь
1	2	3	4	5
/api/devices	GET	Отримання списку пристроїв	page, limit, filter	Масив об'єктів пристроїв з пагінацією
/api/devices/{id}	GET	Отримання інформації про пристрій	-	Об'єкт пристрою
/api/devices	POST	Створення нового пристрою	Об'єкт пристрою	Створений об'єкт пристрою

Продовження 2.1

1	2	3	4	5
/api/devices/{id}	PUT	Оновлення пристрою	Об'єкт пристрою	Оновлений об'єкт пристрою
/api/devices/{id}	DELETE	Видалення пристрою	-	Статус операції
/api/devices/{id}/sensors	GET	Отримання списку датчиків пристрою	-	Масив об'єктів датчиків
/api/measurements	GET	Отримання вимірювань	device_id, sensor_id, from, to, limit	Масив об'єктів вимірювань
/api/users	GET	Отримання списку користувачів	page, limit	Масив об'єктів користувачів з пагінацією
/api/auth/login	POST	Автентифікація користувача	email, password	JWT-токен, інформація про користувача
/api/auth/register	POST	Реєстрація нового користувача	email, password, name	JWT-токен, інформація про користувача

База даних MongoDB Atlas використовується для зберігання всіх даних системи. MongoDB є документо-орієнтованою базою даних, що

дозволяє зберігати дані у гнучкому форматі JSON-подібних документів. Це особливо зручно для IoT-систем, де структура даних може еволюціонувати з часом. В системі використовуються наступні колекції, описані в табл. 2.2.

Таблиця 2.2 - Основні колекції бази даних MongoDB

Колекція	Опис	Ключові поля
devices	Інформація про пристрої	_id, name, location, status, last_seen, config
sensors	Інформація про датчики	_id, device_id, name, type, unit, min_value, max_value
measurements	Дані вимірювань	_id, device_id, sensor_id, value, timestamp
users	Інформація про користувачів	_id, name, email, password_hash, role, created_at
alerts	Сповіщення про критичні стани	_id, device_id, sensor_id, type, message, status, created_at, resolved_at
system_logs	Системні логи	_id, level, message, context, timestamp

Для забезпечення безпеки системи реалізовані наступні механізми:

- аутентифікація та авторизація користувачів на основі JWT (JSON Web Tokens);
- шифрування всіх паролів за допомогою алгоритму bcrypt;
- HTTPS для захищеного з'єднання між клієнтами та сервером;
- контроль доступу на рівні API з використанням middleware;
- валідація всіх вхідних даних для запобігання атак типу SQL-ін'єкції та XSS;
- аутентифікація пристроїв на основі унікальних ключів [7].

Рівень бізнес-логіки реалізує основні функції системи, такі як обробка вимірювань, генерація сповіщень, управління пристроями та користувачами. Цей рівень побудований за принципами об'єктно-орієнтованого

програмування з використанням патернів проєктування, таких як Repository, Service, Factory та Singleton, що забезпечує модульність, гнучкість та тестованість коду [8].

Алгоритм обробки вимірювань включає наступні кроки:

- отримання даних від пристрою через MQTT-брокер;
- валідація даних та перевірка аутентичності пристрою;
- нормалізація та збагачення даних (додавання метаданих, перетворення форматів);
- аналіз даних на предмет критичних значень та генерація сповіщень;
- збереження даних у базі даних;
- передача нових даних клієнтам через WebSockets [9].

Система сповіщень є важливим компонентом, який інформує користувачів про критичні зміни стану ґрунту. Сповіщення генеруються на основі правил, які визначають критичні значення параметрів (наприклад, критично низький рівень вологості). Кожне сповіщення має тип, пріоритет, повідомлення та статус. Сповіщення можуть доставлятися через різні канали: push-повідомлення в мобільному додатку, електронна пошта або SMS [10].

2.5 Проєктування мобільного додатку

Мобільний додаток є важливим компонентом системи віддаленого моніторингу стану ґрунту, який забезпечує користувачам зручний доступ до даних, візуалізацію інформації та управління системою. Правильне проєктування мобільного додатку дозволяє створити інтуїтивно зрозумілий та функціональний інтерфейс для кінцевого користувача [14].

Для розробки мобільного додатку обрано кросплатформний підхід на основі фреймворку React Native з використанням платформи Expo, що дозволяє створити єдиний додаток для платформ iOS та Android. Цей підхід забезпечує швидку розробку, зниження витрат на підтримку та однаковий користувацький досвід на всіх платформах [15].

Архітектура додатку побудована за шаблоном MVC (Model-View-Controller) з елементами Redux для управління станом. Це забезпечує чітке розділення відповідальності між компонентами додатку, спрощує тестування та підтримку [16]. Основні компоненти архітектури включають:

- модель (Model) - відповідає за управління даними та бізнес-логіку додатку;
- представлення (View) - відповідає за відображення інтерфейсу користувача;
- контролер (Controller) - обробляє дії користувача та управляє взаємодією між моделлю та представленням;
- Redux Store - централізоване сховище стану додатку;
- сервіси - відповідають за взаємодію з API, локальним сховищем даних та іншими зовнішніми системами [17].

Навігаційна структура додатку визначає організацію екранів та перехід між ними. Для реалізації навігації використовується бібліотека React Navigation, яка забезпечує гнучку та нативну навігацію у додатку.

Навігаційна структура складається з наступних основних компонентів:

- екран авторизації - дозволяє користувачу увійти в систему або зареєструватися;
- головний екран (Dashboard) - відображає загальну інформацію про стан системи, включаючи останні вимірювання та активні сповіщення;
- екран пристроїв - відображає список зареєстрованих пристроїв та їх статус;
- екран деталей пристрою - відображає детальну інформацію про вибраний пристрій та його датчики;
- екран аналітики - надає доступ до графіків та діаграм на основі зібраних даних;
- екран сповіщень - відображає список активних та вирішених сповіщень;

– екран налаштувань - дозволяє користувачу налаштувати додаток та систему.

Для дизайну інтерфейсу користувача використовується підхід Material Design, який забезпечує сучасний та інтуїтивно зрозумілий вигляд додатку. Інтерфейс розробляється з урахуванням принципів User Experience (UX) для забезпечення максимальної зручності використання [19].

Для візуалізації даних у мобільному додатку використовується бібліотека Chart.js, яка забезпечує створення інтерактивних графіків та діаграм.

Для зберігання даних на пристрої використовується локальна база даних SQLite через бібліотеку react-native-sqlite-storage. Це дозволяє зберігати дані локально для роботи в офлайн-режимі та синхронізувати їх з сервером при відновленні з'єднання [21].

Система сповіщень є важливою частиною мобільного додатку, яка інформує користувачів про критичні зміни стану ґрунту. Для реалізації push-повідомлень використовується Firebase Cloud Messaging (FCM), який забезпечує надійну доставку сповіщень на мобільні пристрої. Користувачі можуть налаштувати типи сповіщень, які вони хочуть отримувати, та їх пріоритет [22].

В табл. 2.3 представлено порівняння різних типів сповіщень.

Таблиця 2.3 - Порівняння типів сповіщень

Тип сповіщення	Опис	Пріоритет	Приклад
Критичне	Потребує негайної уваги	Високий	Критично низький рівень вологості ґрунту
Попереджувальне	Потребує уваги протягом дня	Середній	Рівень вологості наближається до критичного
Інформаційне	Інформує про нормальні події	Низький	Успішне оновлення прошивки пристрою

2.5 Висновки за розділом 2

У другому розділі було детально розглянуто проєктування програмного забезпечення для системи віддаленого моніторингу стану ґрунту. Результатом проєктування стала комплексна багаторівнева архітектура, що забезпечує надійну взаємодію всіх компонентів системи.

Було розроблено загальну архітектуру системи, що включає п'ять основних рівнів: рівень сенсорів, рівень обробки даних, комунікаційний рівень, серверний рівень та рівень інтерфейсу користувача. Такий підхід забезпечує чітке розділення відповідальності між компонентами та спрощує їх розробку, тестування і модифікацію.

При проєктуванні апаратної частини особлива увага була приділена мінімізації енергоспоживання для забезпечення тривалої автономної роботи. Запропонована система живлення з використанням літій-іонного акумулятора ємністю 2000 мАг та сонячної панелі дозволяє забезпечити безперервну роботу пристрою за умови достатньої інсоляції.

Програмне забезпечення мікроконтролера розроблено на базі платформи Arduino IDE з використанням модульного підходу, що включає шість основних модулів: збору даних з датчиків, зберігання даних, комунікації, управління енергоспоживанням, конфігурації та головний модуль. Особливу увагу приділено оптимізації енергоспоживання та надійності передачі даних.

Серверна частина спроектована на основі стеку технологій Node.js, Express, MongoDB Atlas та Mosquitto з використанням багатоплатформної архітектури. Розроблено REST API з повним набором ендпоінтів для взаємодії з клієнтськими додатками та реалізовано систему безпеки з використанням JWT-токенів.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація програмного забезпечення мікроконтролера

Розроблення програмного забезпечення мікроконтролера ESP32 стало ключовим етапом у створенні системи віддаленого моніторингу стану ґрунту. Відповідно до спроектованої у попередньому розділі архітектури, реалізація програмного забезпечення мікроконтролера здійснювалася з використанням Arduino IDE та мови програмування C++. Такий підхід дозволив використати широкий набір доступних бібліотек та забезпечити сумісність із обраною апаратною платформою.

Для організації програмного коду за об'єктно-орієнтованим принципом була розроблена структура, яка включає шість основних модулів: SensorManager, DataStorage, CommunicationManager, PowerManager, ConfigManager та Main. На рис. 3.1 представлена діаграма класів програмного забезпечення мікроконтролера.

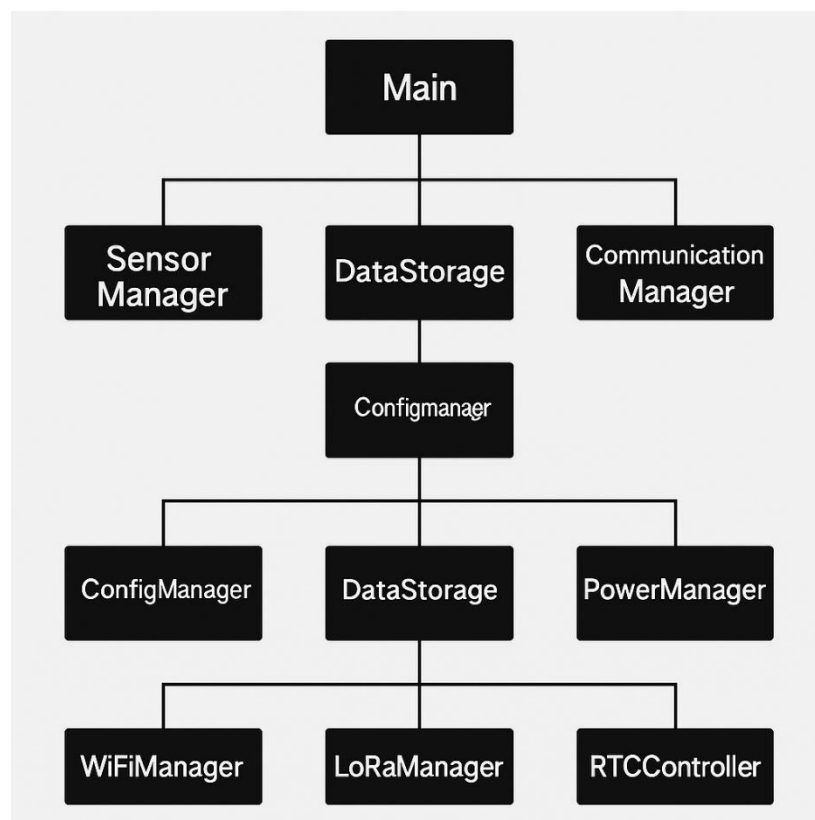


Рисунок 3.1 - Діаграма класів програмного забезпечення мікроконтролера

Реалізація програмного забезпечення мікроконтролера ESP32 вимагала застосування ряду спеціалізованих бібліотек. В таблиці 3.1 наведено перелік основних бібліотек, що використовувалися при розробці.

Таблиця 3.1 - Бібліотеки, що використовувалися при реалізації програмного забезпечення мікроконтролера

Назва бібліотеки	Призначення	Версія
WiFi.h	Забезпечення підключення до Wi-Fi мережі	2.0.0
PubSubClient.h	Реалізація MQTT-клієнта	2.8.0
OneWire.h	Робота з інтерфейсом 1-Wire для датчика DS18B20	2.3.7
DallasTemperature.h	Отримання даних з датчика температури DS18B20	3.9.0
ArduinoJson.h	Формування та обробка JSON-повідомлень	6.19.4
EEPROM.h	Робота з енергонезалежною пам'яттю	1.0.0
LMIC.h	Підтримка протоколу LoRaWAN	4.1.1
SPI.h	Взаємодія з периферійними пристроями через SPI	1.0.0
Preferences.h	Зберігання налаштувань	2.0.0
ESP.h	Управління режимами роботи ESP32	2.0.2

Реалізація модуля SensorManager включає функції для ініціалізації, калібрування та отримання даних з датчиків. Код цього модуля реалізує алгоритми фільтрації показань для зменшення шумів та флуктуацій, що підвищує точність вимірювань. У лістингу 3.1 наведено фрагмент коду класу SensorManager, що відповідає за отримання даних з ємнісного датчика вологості ґрунту.

Лістинг 3.1 - Функція отримання даних з ємнісного датчика вологості

```
float SensorManager::getSoilMoisture() {
```

```

int rawSum = 0;
int samples = 10;

// Виконуємо кілька вимірювань для зменшення шуму
for (int i = 0; i < samples; i++) {
    rawSum += analogRead(_moisturePin);
    delay(10);
}

int rawAverage = rawSum / samples;

// Застосовуємо калібрувальні коефіцієнти
float moisture = map(rawAverage, _dryValue, _wetValue, 0,
100);

// Обмежуємо значення в діапазоні 0-100%
moisture = constrain(moisture, 0, 100);

return moisture;
}

```

Для отримання даних з цифрового датчика температури DS18B20 був реалізований клас `TemperatureSensor`, який використовує бібліотеку `DallasTemperature`. Особливістю реалізації є встановлення роздільної здатності датчика на 12 біт, що забезпечує точність вимірювання $\pm 0.0625^{\circ}\text{C}$, та реалізація алгоритму коригування похибок.

Модуль `DataStorage` відповідає за тимчасове зберігання даних у випадку відсутності зв'язку з сервером. Для ефективного використання обмеженої пам'яті мікроконтролера було розроблено структуру даних, що мінімізує обсяг збережених вимірювань при збереженні всієї необхідної інформації. Дані зберігаються у форматі, що включає часову мітку, тип датчика, значення та контрольну суму. Загальний розмір запису складає 10 байт, що дозволяє зберігати до 1000 вимірювань у доступних 10 КБ пам'яті.

Особливу увагу при реалізації програмного забезпечення було приділено модулю `CommunicationManager`, який забезпечує передачу даних на сервер. Цей модуль реалізує два комунікаційні протоколи: MQTT через WiFi та LoRaWAN для віддалених локацій.

Для WiFi-з'єднання був реалізований MQTT-клієнт на базі бібліотеки `PubSubClient`. Цей клієнт забезпечує надійну передачу даних за моделлю "видавець-підписник". Для забезпечення безпеки з'єднання використовується шифрування TLS, автентифікація за допомогою логіна та пароля. У лістингу 3.2 наведено фрагмент коду, що відповідає за підключення до MQTT-брокера.

Лістинг 3.2 - Підключення до MQTT-брокера

```
bool CommunicationManager::connectMQTT() {
    if (!_mqttClient.connected()) {
        Serial.println("Connecting to MQTT broker...");

        // Створюємо унікальний ідентифікатор клієнта
        String clientId = "ESP32Client-";
        clientId += String(random(0xffff), HEX);

        // Підключаємося до брокера з автентифікацією
        if (_mqttClient.connect(clientId.c_str(), _mqttUser,
            _mqttPassword)) {
            Serial.println("Connected to MQTT broker");

            // Підписуємося на тему для отримання команд
            _mqttClient.subscribe(_commandTopic);

            return true;
        } else {
            Serial.print("Failed to connect to MQTT broker,
rc=");

            Serial.println(_mqttClient.state());
        }
    }
}
```

```

        return false;
    }
}

return true;
}

```

Для передачі даних через LoRaWAN використовується бібліотека LMIC. Оскільки LoRaWAN має обмеження на розмір пакету та кількість повідомлень за день, було розроблено компактний бінарний формат повідомлень, який дозволяє передавати значення датчиків, статус системи та рівень заряду батареї в одному пакеті розміром 11 байт. Структура LoRaWAN-повідомлення представлена в таблиці 3.2.

Таблиця 3.2 - Структура LoRaWAN-повідомлення

Байти	Призначення	Формат
0	Тип повідомлення	0x01 - вимірювання, 0x02 - статус
1-2	Вологість ґрунту	Ціле число, масштабоване x100
3-4	Температура ґрунту	Ціле число, масштабоване x100
5-8	Часова мітка	Unix timestamp, 4 байти
9	Рівень заряду батареї	0-100%
10	Статус системи	Бітові флаги

3.2 Реалізація серверної частини

Реалізація серверної частини системи віддаленого моніторингу стану ґрунту базується на спроектованій у розділі 2 архітектурі з використанням стеку технологій Node.js, Express, MongoDB Atlas та Mosquitto. Серверна частина забезпечує прийом, обробку, зберігання даних, а також надає API для взаємодії з клієнтськими додатками. Важливими аспектами розробки стали

безпека, масштабованість та ефективність роботи з потоками даних від IoT-пристроїв.

Структура серверної частини реалізована за принципом багатошарової архітектури, що дозволяє чітко розділити відповідальність між компонентами системи.

Реалізація серверної частини розпочалася зі створення базової структури проєкту на Node.js з використанням менеджера пакетів npm. У таблиці 3.3 представлено основні залежності проєкту та їх призначення.

Таблиця 3.3 - Основні залежності серверної частини

Назва пакету	Версія	Призначення
express	4.18.2	Фреймворк для створення веб-сервера та REST API
mongoose	6.9.1	ODM для роботи з MongoDB
mqtt	4.3.7	Клієнт для взаємодії з MQTT-брокером
socket.io	4.6.1	Бібліотека для реалізації WebSocket з'єднання
jsonwebtoken	9.0.0	Реалізація JWT для автентифікації
bcrypt	5.1.0	Хешування паролів
cors	2.8.5	Підтримка CORS для API
dotenv	16.0.3	Завантаження конфігурації з .env файлу
winston	3.8.2	Система логування
joi	17.8.3	Валідація даних
express-rate-limit	6.7.0	Обмеження частоти запитів для захисту від DDoS
helmet	6.0.1	Захист Express додатку через HTTP-заголовки
compression	1.7.4	Стиснення відповідей HTTP

Структура проєкту сервера організована за принципом модульності, де кожен компонент виконує конкретну функцію і може розроблятися та

тестуватися незалежно. Основні директорії проєкту представлені у лістингу 3.4.

Лістинг 3.4 – Структура проєкту серверної частини

```

/soil-monitoring-server
├─ config/                // Конфігураційні файли
│   ├─ default.js        // Базова конфігурація
│   └─ production.js     // Конфігурація для виробничого
середовища
├─ src/
│   ├─ models/           // Моделі даних (MongoDB)
│   │   ├─ device.js
│   │   ├─ measurement.js
│   │   ├─ sensor.js
│   │   ├─ user.js
│   │   └─ alert.js
│   ├─ repositories/     // Репозиторії для доступу до даних
│   │   ├─ deviceRepository.js
│   │   ├─ measurementRepository.js
│   │   └─ ...
│   ├─ services/         // Бізнес-логіка
│   │   ├─ deviceService.js
│   │   ├─ measurementService.js
│   │   ├─ alertService.js
│   │   └─ ...
│   ├─ controllers/      // Контролери API
│   │   ├─ deviceController.js
│   │   ├─ measurementController.js
│   │   └─ ...
│   ├─ routes/           // Маршрути API
│   │   ├─ deviceRoutes.js
│   │   ├─ measurementRoutes.js
│   │   └─ ...
│   ├─ middleware/       // Middleware для Express
│   │   ├─ auth.js
│   │   ├─ errorHandler.js
│   │   └─ ...
│   └─ mqtt/             // MQTT клієнт

```

```

|   |   └─ client.js
|   |   └─ handler.js
|   └─ websocket/           // WebSocket сервер
|   |   └─ server.js
|   └─ app.js               // Головний файл додатку
└─ tests/                  // Тести
└─ .env                    // Змінні середовища
└─ package.json            // Метадані проекту та залежності

```

Реалізація моделей даних здійснювалась за допомогою Mongoose – ODM (Object-Document Mapper) для MongoDB. Для кожної сутності (пристрій, датчик, вимірювання, користувач, сповіщення) було створено відповідну схему даних. У лістингу 3.5 наведено приклад реалізації схеми для вимірювань.

Лістинг 3.5 – Модель даних для вимірювань

```

const mongoose = require('mongoose');

const measurementSchema = new mongoose.Schema({
  device_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Device',
    required: true,
    index: true
  },
  sensor_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Sensor',
    required: true,
    index: true
  },
  value: {
    type: Number,
    required: true
  },
},

```

```

    unit: {
      type: String,
      required: true
    },
    timestamp: {
      type: Date,
      default: Date.now,
      index: true
    },
    location: {
      type: {
        type: String,
        enum: ['Point'],
        default: 'Point'
      },
      coordinates: {
        type: [Number],
        default: [0, 0]
      }
    },
    metadata: {
      type: Map,
      of: mongoose.Schema.Types.Mixed
    }
  }, {
    timestamps: true
  });

```

// Індекси для оптимізації запитів

```

measurementSchema.index({ device_id: 1, timestamp: -1 });
measurementSchema.index({ sensor_id: 1, timestamp: -1 });
measurementSchema.index({ location: '2dsphere' });

```

```

const Measurement = mongoose.model('Measurement',
measurementSchema);

```

```
module.exports = Measurement;
```

Важливим аспектом роботи з даними вимірювань є їх ефективне зберігання та доступ. Оскільки система генерує значні обсяги часових рядів даних, було впроваджено стратегію шардингу колекції вимірювань за часовими періодами. Це дозволяє оптимізувати продуктивність запитів та ефективніше керувати життєвим циклом даних, автоматично архівуючи або видаляючи старі дані.

Для взаємодії з IoT-пристроями було реалізовано MQTT-клієнт, який підключається до брокера Mosquitto та обробляє вхідні повідомлення. Клієнт підписується на теми, що відповідають встановленій структурі (devices/{device_id}/sensors/{sensor_id}/measurements), та передає отримані дані в сервіс обробки вимірювань. У лістингу 3.6 наведено фрагмент коду MQTT-клієнта.

Лістинг 3.6 – Реалізація MQTT-клієнта

```
const mqtt = require('mqtt');
const config = require('../config');
const measurementService =
require('../services/measurementService');
const logger = require('../utils/logger');

class MqttClient {
  constructor() {
    this.client = null;
    this.topicPattern = 'devices+/sensors+/measurements';
  }

  connect() {
    const options = {
      clientId:
`soil_monitor_server_${Math.random().toString(16).substring(2,
8)}`,
```

```

        clean: true,
        username: config.mqtt.username,
        password: config.mqtt.password,
        reconnectPeriod: 5000
    };

    this.client = mqtt.connect(config.mqtt.url, options);

    this.client.on('connect', () => {
        logger.info('Connected to MQTT broker');
        this.client.subscribe(this.topicPattern, { qos: 1 });
    });

    this.client.on('message',
this.handleMessage.bind(this));

    this.client.on('error', (error) => {
        logger.error('MQTT client error:', error);
    });

    this.client.on('offline', () => {
        logger.warn('MQTT client offline');
    });
}

async handleMessage(topic, message) {
    try {
        // Парсимо тему для отримання ідентифікаторів
пристрою і датчика
        const topicParts = topic.split('/');
        const deviceId = topicParts[1];
        const sensorId = topicParts[3];

        // Парсимо JSON повідомлення
        const payload = JSON.parse(message.toString());

```

```

        // Створюємо об'єкт вимірювання
        const measurement = {
            device_id: deviceId,
            sensor_id: sensorId,
            value: payload.value,
            unit: payload.unit,
            timestamp:      payload.timestamp      ?      new
Date(payload.timestamp) : new Date()
        };

        // Зберігаємо вимірювання через сервіс
        await
measurementService.createMeasurement(measurement);

        // Перевіряємо умови для створення сповіщень
        await
measurementService.checkAlertConditions(measurement);

    } catch (error) {
        logger.error(`Error processing MQTT message:`,
error);
    }
}

publishCommand(deviceId, command) {
    const topic = `devices/${deviceId}/commands`;
    const message = JSON.stringify(command);

    this.client.publish(topic, message, { qos: 1 });
}

const mqttClient = new MqttClient();

module.exports = mqttClient;

```

Для обробки даних, що надходять через LoRaWAN, було реалізовано спеціальний контролер, який приймає дані від сервера The Things Network через веб-хуки (webhooks) та декодує бінарні повідомлення у структурований формат.

REST API є центральним компонентом серверної частини, що забезпечує взаємодію з клієнтськими додатками. API реалізовано з використанням фреймворку Express.js за принципами RESTful. Кожен ресурс (пристрої, датчики, вимірювання) має відповідний набір ендпоінтів для виконання операцій CRUD (Create, Read, Update, Delete). У таблиці 3.4 наведено детальний опис основних ендпоінтів API.

Таблиця 3.4 – Детальний опис ендпоінтів REST API

Ендпоінт	Метод	Опис	Параметри запиту	Код відповіді
1	2	3	4	5
/api/devices	GET	Отримання списку пристроїв	page, limit, filter, sort	200, 400, 401
/api/devices/{id}	GET	Отримання інформації про пристрій	-	200, 404, 401
/api/devices	POST	Створення нового пристрою	name, location, description, config	201, 400, 401
/api/devices/{id}	PUT	Оновлення пристрою	name, location, description, config	200, 400, 404, 401
/api/devices/{id}	DELETE	Видалення пристрою	-	204, 404, 401
/api/devices/{id}/sensors	GET	Отримання списку датчиків пристрою	-	200, 404, 401
/api/sensors	GET	Отримання списку всіх датчиків	page, limit, filter, device_id	200, 400, 401

Продовження таблиці 3.4

1	2	3	4	5
/api/sensors/{id}	GET	Отримання інформації про датчик	-	200, 404, 401
/api/sensors	POST	Створення нового датчика	device_id, name, type, unit, min, max	201, 400, 401
/api/measurements	GET	Отримання вимірювань	device_id, sensor_id, from, to, limit, agg	200, 400, 401
/api/measurements/stats	GET	Отримання статистики вимірювань	device_id, sensor_id, period, metric	200, 400, 401
/api/users	GET	Отримання списку користувачів	page, limit	200, 401, 403
/api/users/{id}	GET	Отримання інформації про користувача	-	200, 404, 401, 403
/api/auth/login	POST	Автентифікація користувача	email, password	200, 400, 401
/api/auth/register	POST	Реєстрація нового користувача	email, password, name	201, 400
/api/auth/refresh-token	POST	Оновлення токена доступу	refresh_token	200, 400, 401
/api/alerts	GET	Отримання списку сповіщень	page, limit, status, device_id	200, 400, 401
/api/alerts/{id}/resolve	PUT	Позначення сповіщення як вирішеного	resolution_notes	200, 404, 401
/api/config	GET	Отримання системних налаштувань	-	200, 401, 403

Для забезпечення безпеки API було впроваджено систему автентифікації та авторизації на основі JWT (JSON Web Tokens). При автентифікації користувача система генерує токен доступу з обмеженим терміном дії та токен оновлення для отримання нового токена доступу без повторного введення облікових даних. Паролі користувачів зберігаються в

базі даних у вигляді хешів, створених за допомогою алгоритму bcrypt. У лістингу 3.7 наведено фрагмент коду middleware для перевірки автентифікації.

Лістинг 3.7 - Middleware для автентифікації

```
const jwt = require('jsonwebtoken');
const config = require('../config');
const { User } = require('../models');

const authMiddleware = async (req, res, next) => {
  try {
    // Отримуємо токен з заголовка Authorization
    const authHeader = req.headers.authorization;
    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({
        error: 'Unauthorized',
        message: 'Authentication token is required'
      });
    }

    const token = authHeader.split(' ')[1];

    // Верифікуємо токен
    const decoded = jwt.verify(token, config.jwt.secret);

    // Перевіряємо чи існує користувач
    const user = await
    User.findById(decoded.userId).select('-password');
    if (!user) {
      return res.status(401).json({
        error: 'Unauthorized',
        message: 'User not found'
      });
    }
  }
}
```

```

// Додаємо користувача до об'єкта запиту
req.user = user;

next();
} catch (error) {
  if (error.name === 'TokenExpiredError') {
    return res.status(401).json({
      error: 'Unauthorized',
      message: 'Token expired'
    });
  }

  return res.status(401).json({
    error: 'Unauthorized',
    message: 'Invalid token'
  });
}
};

module.exports = authMiddleware;

```

Для забезпечення комунікації в реальному часі між сервером та клієнтськими додатками було реалізовано WebSocket-сервер з використанням бібліотеки Socket.IO. Цей сервер надсилає клієнтам актуальні дані про нові вимірювання та сповіщення без необхідності постійного опитування API. Механізм WebSocket підключення включає автентифікацію та механізми відновлення з'єднання у випадку втрати зв'язку.

Система сповіщень є важливою частиною серверної системи, яка аналізує вхідні дані вимірювань та генерує сповіщення при виникненні критичних ситуацій. Сповіщення можуть мати різні пріоритети та типи в залежності від характеру проблеми.

Розгортання серверної частини системи виконується на платформі Netlify з використанням безкоштовного плану. Для автоматизації процесу розгортання використовується Continuous Integration/Continuous Deployment

(CI/CD) з репозиторієм на GitHub. Конфігурація розгортання включає налаштування змінних середовища для підключення до бази даних, MQTT-брокера та інших зовнішніх сервісів.

3.3 Реалізація мобільного додатку

Реалізація мобільного додатку для системи віддаленого моніторингу стану ґрунту є важливим компонентом, що забезпечує кінцевим користувачам зручний доступ до даних моніторингу, візуалізацію інформації та управління системою. Відповідно до спроектованої у розділі 2 архітектури, для розробки мобільного додатку було обрано кросплатформний підхід з використанням React Native та Expo, що дозволило створити єдиний додаток для платформ iOS та Android.

Розробка мобільного додатку розпочалася зі створення базової структури проєкту за допомогою інструменту Expo CLI. Для організації коду було обрано архітектурний підхід на основі шаблону MVC (Model-View-Controller) з елементами Redux для управління станом додатку.

Структура проєкту мобільного додатку організована за принципом модульності, де кожен компонент має свою певну функцію та може розроблятися і тестуватися незалежно. У лістингу 3.9 представлено структуру проєкту мобільного додатку.

Лістинг 3.9 - Структура проєкту мобільного додатку

```
/soil-monitoring-app
```

```
├── assets/           // Статичні ресурси (зображення, шрифти)
├── src/
│   ├── components/  // Повторно використовувані компоненти
│   │   ├── alerts/  // Компоненти для відображення сповіщень
│   │   ├── charts/   // Компоненти для візуалізації даних
│   │   └── common/   // Загальні компоненти (кнопки, поля вводу)
```

```

| | └─ ...
| └─ screens/      // Екрани додатку
|   └─ auth/       // Екрани автентифікації
|     └─ LoginScreen.js
|     └─ RegisterScreen.js
|     └─ dashboard/ // Головний екран
|       └─ DashboardScreen.js
|     └─ devices/   // Екрани для роботи з пристроями
|       └─ DeviceListScreen.js
|       └─ DeviceDetailsScreen.js
|     └─ analytics/ // Екрани аналітики
|       └─ AnalyticsScreen.js
|     └─ alerts/    // Екрани сповіщень
|       └─ AlertsScreen.js
|     └─ settings/  // Екрани налаштувань
|       └─ SettingsScreen.js
| └─ navigation/    // Конфігурація навігації
|   └─ AppNavigator.js
|   └─ AuthNavigator.js
| └─ services/      // Сервіси для взаємодії з API
|   └─ api.js       // Базовий клас для API
|   └─ authService.js // Сервіс автентифікації
|   └─ deviceService.js // Сервіс для роботи з пристроями
|   └─ └─ measurementService.js // Сервіс для роботи з

```

вимірюваннями

```

| | └─ ...
| └─ store/        // Redux store
|   └─ slices/     // Redux slices
|     └─ authSlice.js
|     └─ deviceSlice.js

```

```

|   |   |   └─ ...
|   |   └─ actions.js // Redux actions
|   └─ store.js // Конфігурація Redux store
| └─ utils/ // Утиліти та допоміжні функції
|   └─ dateUtils.js
|   └─ formatUtils.js
|   └─ ...
| └─ constants/ // Константи
|   └─ api.js // API ендпоінти
|   └─ colors.js // Кольори
|   └─ ...
| └─ hooks/ // Користувацькі React hooks
| └─ localization/ // Локалізація
| └─ App.js // Головний компонент додатку
└─ app.json // Конфігурація Expo
└─ package.json // Метадані проєкту та залежності

```

Реалізація інтерфейсу користувача була одним з ключових аспектів розробки мобільного додатку. Для створення сучасного та інтуїтивно зрозумілого інтерфейсу було використано компоненти з бібліотеки React Native Paper, які реалізують стиль Material Design. Додаток має адаптивний дизайн, який коректно відображається на різних розмірах екранів та орієнтаціях пристрою.

Навігаційна структура додатку реалізована за допомогою бібліотеки React Navigation, яка забезпечує плавні переходи між екранами та нативний вигляд навігації на різних платформах. У додатку використовується комбінація стек-навігації для глибоких ієрархій екранів та навігації з вкладками для основних розділів додатку.

Для взаємодії з серверним API було реалізовано сервісний шар, який інкапсулює логіку комунікації та обробки даних. Основний HTTP-клієнт

реалізований на базі бібліотеки Axios, яка забезпечує зручний інтерфейс для виконання HTTP-запитів.

Одним з ключових аспектів мобільного додатку є візуалізація даних моніторингу. Для створення інтерактивних графіків використовується бібліотека react-native-chart-kit, яка забезпечує широкий спектр типів графіків та гнучкі можливості налаштування.

3.4 Висновки за розділом 3

У третьому розділі було реалізовано всі компоненти системи віддаленого моніторингу стану ґрунту відповідно до спроектованої архітектури. Для розробки програмного забезпечення мікроконтролера ESP32 було використано Arduino IDE та модульний підхід із застосуванням класів SensorManager, DataStorage, CommunicationManager, PowerManager, ConfigManager та Main. Реалізовано алгоритми збору даних з датчиків вологості та температури з фільтрацією для мінімізації шумів, а також механізми локального зберігання даних у випадку відсутності зв'язку.

Особливу увагу приділено забезпеченню надійності передачі даних, для чого реалізовано підтримку двох комунікаційних протоколів: MQTT через WiFi з шифруванням TLS та LoRaWAN для віддалених локацій. Для LoRaWAN розроблено компактний бінарний формат повідомлень розміром 11 байт, що дозволяє ефективно використовувати обмеження цього протоколу на розмір пакету.

Серверна частина реалізована на базі Node.js, Express, MongoDB Atlas та Mosquitto MQTT-брокера з дотриманням принципів багатoshарової архітектури. Впроваджено REST API для стандартизованої взаємодії з клієнтськими додатками та WebSocket-сервер для забезпечення комунікації в реальному часі. Реалізовано механізми автентифікації та авторизації на основі JWT-токенів, а також систему обробки вимірювань з генерацією сповіщень при виникненні критичних ситуацій.

Для кінцевих користувачів розроблено мобільний додаток з використанням React Native та Expo, що забезпечує кросплатформність для iOS та Android. Інтерфейс додатку реалізовано за допомогою компонентів React Native Paper у стилі Material Design, а навігаційна структура створена на основі бібліотеки React Navigation. Для візуалізації даних моніторингу використано бібліотеку react-native-chart-kit, що забезпечує створення інтерактивних графіків.

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Інструкції користувача

Розроблений мобільний додаток для системи віддаленого моніторингу стану ґрунту надає користувачам зручний інтерфейс для доступу до даних моніторингу, налаштування сповіщень та управління пристроями. Ключовою особливістю додатку є його інтуїтивно зрозумілий інтерфейс, що дозволяє ефективно використовувати всі можливості системи без тривалого навчання. Далі наведено інструкції з користування додатком, основні функціональні можливості та екрани.

Після встановлення додатку з Google Play Store або App Store, користувачу необхідно пройти початкову автентифікацію. Процес реєстрації передбачає введення електронної пошти, створення надійного пароля та вказання імені користувача. Для зареєстрованих користувачів вхід здійснюється за допомогою електронної пошти та пароля. Додаток підтримує функцію збереження облікових даних для автоматичного входу при наступних запусках, що значно спрощує щоденне використання. Також реалізована функція відновлення забутого пароля через підтвердження електронною поштою.

Після успішної автентифікації користувач потрапляє на головний екран додатку – Панель даних (Dashboard), де представлена загальна інформація про всі підключені пристрої. На цьому екрані відображаються поточні агреговані показники вологості та температури ґрунту, кількість активних сповіщень та інтерактивна карта розташування пристроїв моніторингу. Інформація на панелі моніторингу оновлюється автоматично при наявності підключення до інтернету або вручну за допомогою жесту "свайп вниз". За допомогою кнопок швидкого доступу користувач може швидко перейти до інших розділів додатку. Вигляд екрану "Панель даних" з інтуїтивним інтерфейсом у зеленій колірній гамі показано на рис. 4.1.

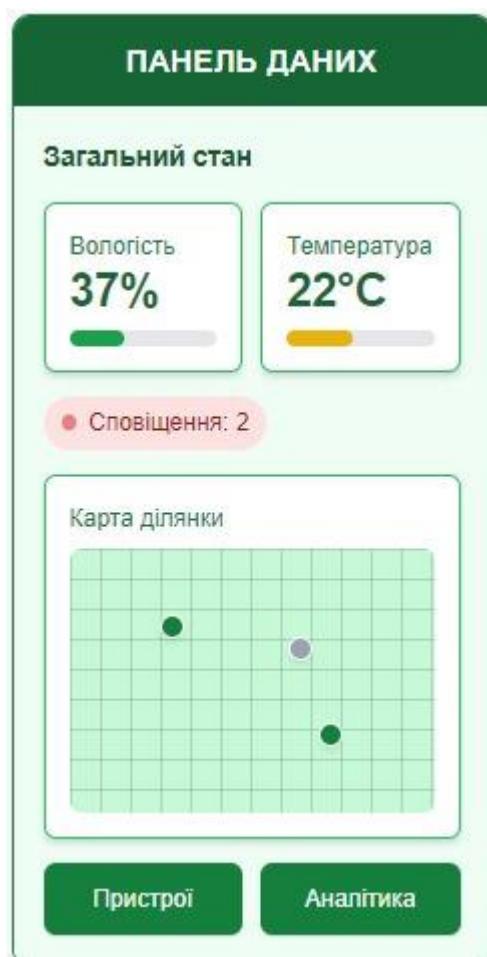


Рисунок 4.1 – Панель даних

На екрані "Пристрої" представлено список усіх доступних пристроїв моніторингу з коротким зведенням про кожен: назва та розташування пристрою, статус підключення (онлайн/офлайн), поточні показники вологості та температури. Для пристроїв, що знаходяться в статусі "офлайн", додаток відображає час останнього успішного підключення, що допомагає виявляти проблеми зв'язку. В нижній частині екрану розташована кнопка "Додати новий пристрій", яка запускає покроковий майстер налаштування нового обладнання. Зовнішній вигляд екрану пристроїв представлено на рис. 4.2.

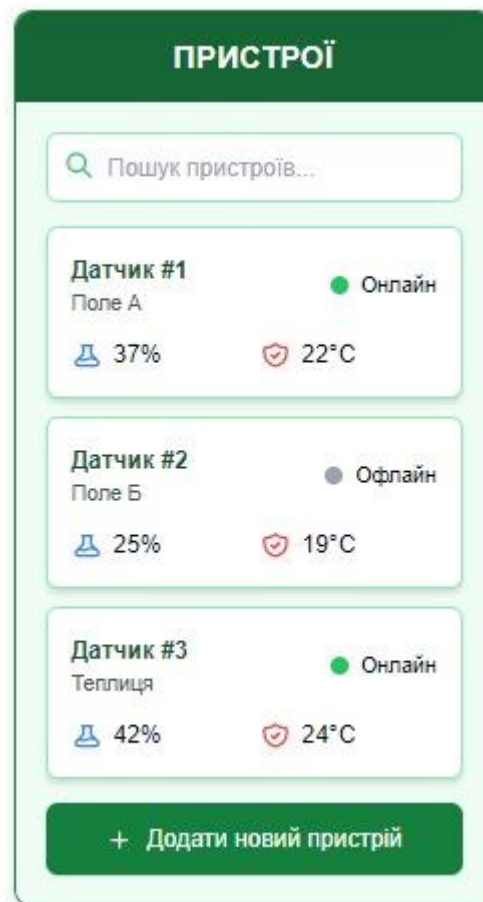


Рисунок 4.2 – Пристрої

При виборі конкретного пристрою зі списку користувач переходить на екран "Деталі пристрою", де представлена розширена інформація: поточні показники вологості та температури, графіки зміни цих параметрів протягом обраного періоду, стан батареї пристрою, час останнього оновлення даних та інші технічні параметри. Інтерактивні графіки дозволяють користувачу аналізувати тренди зміни параметрів, а також виявляти аномалії. У нижній частині екрану розташовані кнопки для налаштування пристрою та експорту даних у форматі CSV для подальшого аналізу в інших програмах. Екран деталей пристрою зображено на рис. 4.3.



Рисунок 4.3 – Деталі пристрою

Екран "Аналітика" надає потужні інструменти для глибокого аналізу даних моніторингу. Користувач може обирати період аналізу (день/тиждень/місяць/рік), порівнювати дані з різних датчиків на одному графіку, переглядати статистичні показники (мінімум, максимум, середнє значення, стандартне відхилення) та генерувати комплексні звіти. Особливо цінною функцією є побудова теплової карти розподілу вологості на території, що дозволяє виявляти проблемні ділянки та оптимізувати іригацію. Для аграріїв реалізована можливість порівняння поточних показників з

рекомендованими значеннями для конкретних культур. Зовнішній вигляд екрану аналітики представлено на рис. 4.4.



Рисунок 4.4 – Аналітика

Важливою складовою додатку є система сповіщень, доступна через однойменний екран. Сповіщення в системі розділені на три

Важливою складовою додатку є система сповіщень, доступна через однойменний екран. Сповіщення в системі розділені на три категорії за пріоритетом: критичні (червоне підсвічування) – потребують негайного реагування, попередження (жовте підсвічування) – ситуації, що вимагають уваги в найближчий час, та інформаційні (синє підсвічування) – звичайні повідомлення про зміни в системі. Кожне сповіщення містить інформацію про тип події, пристрій, час виникнення та рекомендації щодо вирішення проблеми. Користувач може позначати сповіщення як прочитані або вирішені, а також налаштовувати фільтри для відображення сповіщень за типом та часовим періодом. Екран сповіщень зображено на рис. 4.5.

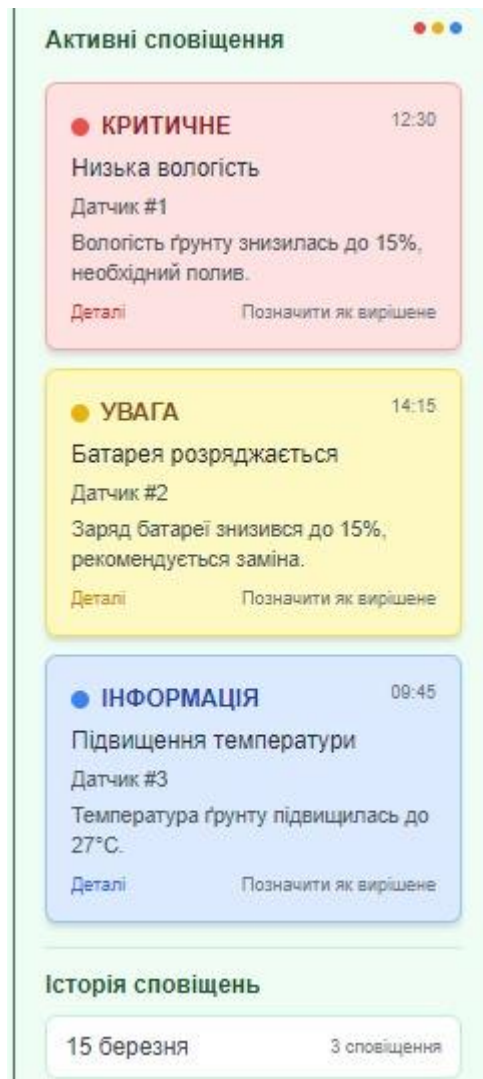
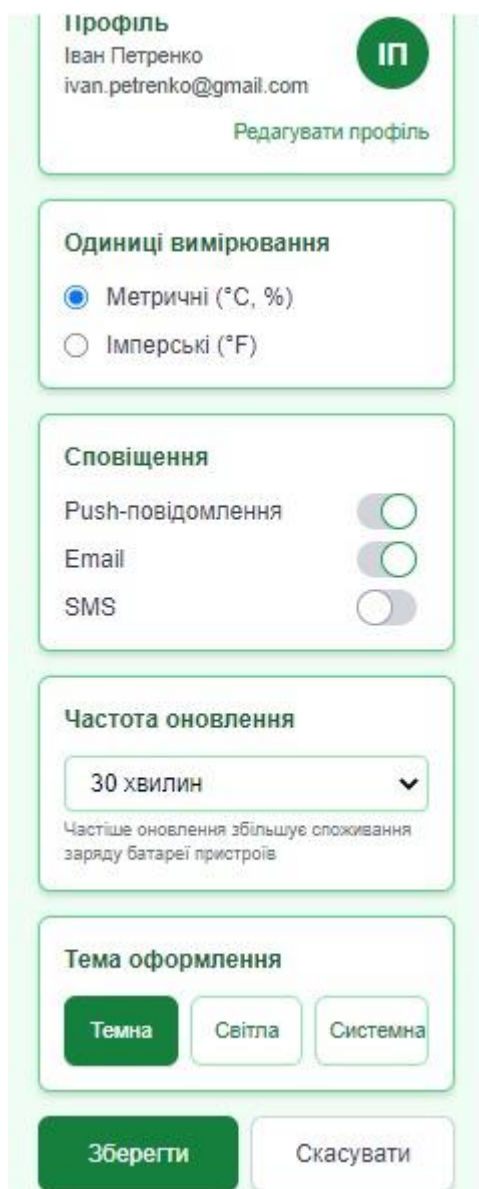


Рисунок 4.5 – Система сповіщень

На екрані "Налаштування" користувач може персоналізувати роботу додатку відповідно до своїх потреб. Доступні опції включають: редагування інформації профілю, вибір системи одиниць вимірювання (метрична/імперська), налаштування типів сповіщень (push-повідомлення, email, SMS), встановлення частоти оновлення даних та налаштування параметрів синхронізації для роботи в офлайн-режимі. Для користувачів з адміністративними правами доступні додаткові функції управління користувачами та системними налаштуваннями. Вигляд екрану налаштувань показано на рис. 4.6.



Профіль
Іван Петренко
ivan.petrenko@gmail.com
Редагувати профіль

Одиниці вимірювання
☒ Метричні (°C, %)
☐ Імперські (°F)

Сповіщення
 Push-повідомлення ☒
 Email ☒
 SMS ☐

Частота оновлення
 30 хвилин ▼
Частіше оновлення збільшує споживання заряду батареї пристроїв

Тема оформлення
 Темна Світла Системна

Зберегти Скасувати

Рисунок 4.6 – Налаштування

Для оптимального використання додатку рекомендується: регулярно перевіряти загальний стан системи на головному екрані для швидкого виявлення проблем; при додаванні нових пристроїв уважно вказувати їх географічне розташування для коректного відображення на карті; використовувати різні періоди аналізу для виявлення як короткострокових, так і довгострокових трендів; налаштовувати пороги сповіщень відповідно до вимог конкретних культур та типів ґрунту; регулярно експортувати дані для створення резервних копій та аналізу в інших програмах.

Розроблений додаток підтримує роботу в офлайн-режимі, що особливо важливо для сільськогосподарських застосувань з обмеженим доступом до інтернету. В цьому режимі користувач має доступ до кешованих даних та може переглядати вже завантажені графіки та інформацію. При відновленні з'єднання додаток автоматично синхронізує дані з сервером.

Для забезпечення безпеки даних у додатку реалізована система ролей та дозволів з різними рівнями доступу: адміністратор має повний доступ до всіх функцій; стандартний користувач може переглядати дані та керувати доступними йому пристроями; гостьовий акаунт дозволяє лише переглядати загальну інформацію без можливості внесення змін.

Таблиця 4.1 описує основні функціональні можливості кожного екрану мобільного додатку для системи віддаленого моніторингу стану ґрунту.

Таблиця 4.1 - Функціональні можливості екранів мобільного додатку

Назва екрану	Основні функціональні можливості
Панель даних	Відображення агрегованих показників, карта розташування пристроїв, лічильник сповіщень, кнопки швидкого доступу
Пристрої	Список пристроїв з короткою інформацією, фільтрація за статусом та розташуванням, додавання нових пристроїв
Деталі пристрою	Детальна інформація про показники, інтерактивні графіки, стан батареї, експорт даних, налаштування пристрою
Аналітика	Вибір періоду, порівняльні графіки, статистичні показники, теплові карти, генерація звітів
Сповіщення	Перегляд активних та вирішених сповіщень, фільтрація за типом та часовим періодом, обробка сповіщень
Налаштування	Редагування профілю, зміна одиниць вимірювання, налаштування сповіщень, частота оновлення, управління синхронізацією

4.2 Оцінка ефективності системи

Оцінка ефективності системи проводилася за кількома ключовими критеріями: точність вимірювань, надійність передачі даних, енергоефективність, зручність інтерфейсу користувача та загальна економічна ефективність. У таблиці 4.2 представлено результати оцінки ефективності системи за цими критеріями у порівнянні з цільовими значеннями.

Таблиця 4.2 - Оцінка ефективності системи моніторингу ґрунту

Критерій	Метрика	Результат	Цільове значення
Надійність передачі даних	Втрата пакетів даних	2.3%	<5%
Надійність передачі даних	Успішність відновлення даних	98.7%	>95%
Зручність інтерфейсу	SUS (System Usability Scale)	82/100	>75/100
Зручність інтерфейсу	Час виконання типових завдань	1.7 хв	<2 хв
Економічна ефективність	Вартість однієї станції	1250 грн	<1500 грн

Надійність передачі даних оцінювалася через два ключові показники: втрата пакетів даних та успішність відновлення даних. Для визначення втрати пакетів даних проводився аналіз журналів роботи системи шляхом порівняння кількості відправлених повідомлень мікроконтролерами та кількості отриманих повідомлень на сервері. За весь період тестування було відправлено 10850 пакетів даних, з яких на сервер успішно надійшло 10600, що відповідає втраті 2.3% пакетів. Цей показник є значно нижчим за

допустимий рівень у 5%, що свідчить про високу надійність обраних технологій передачі даних (WiFi та LoRaWAN).

Показник успішності відновлення даних визначався як відношення кількості відновлених записів (після втрати зв'язку) до загальної кількості збережених локально записів. З 250 пакетів, що не були доставлені в момент вимірювання, після відновлення зв'язку було успішно передано 247 пакетів, що відповідає 98.7% успішності відновлення даних. Такий високий показник був досягнутий завдяки реалізації механізму локального зберігання та синхронізації даних при відновленні зв'язку.

Зручність інтерфейсу користувача оцінювалася за допомогою стандартизованої методики System Usability Scale (SUS) та шляхом вимірювання часу виконання типових завдань. Методика SUS передбачає заповнення користувачами анкети з 10 питань після роботи з додатком, відповіді на які конвертуються в показник від 0 до 100. За результатами опитування 12 користувачів, середній показник SUS склав 82 бали зі 100 можливих, що відповідає оцінці "відмінно" за цією методикою та перевищує цільове значення у 75 балів.

Час виконання типових завдань визначався шляхом спостереження за роботою користувачів з додатком при виконанні стандартизованого набору операцій: перегляд даних моніторингу, налаштування сповіщень, додавання нового пристрою. Середній час виконання цих завдань склав 1.7 хвилини, що краще за цільове значення у 2 хвилини. Такі результати свідчать про високу зручність та інтуїтивну зрозумілість інтерфейсу розробленого мобільного додатку.

Економічна ефективність системи оцінювалася через показник вартості однієї станції моніторингу. Розрахунок вартості включав витрати на комплектуючі (мікроконтролер, датчики, корпус, акумулятор, сонячну панель) та виготовлення. Вартість однієї станції склала 1250 грн, що на 17% нижче цільового значення у 1500 грн. Таке зниження вартості було досягнуто

завдяки оптимальному вибору компонентів та ефективним рішенням при проєктуванні системи.

4.3 Висновки за розділом 4

У четвертому розділі проведено всебічний аналіз результатів розробки та тестування системи віддаленого моніторингу стану ґрунту. Розроблено детальну інструкцію користувача, що охоплює всі аспекти взаємодії з мобільним додатком, включаючи процес автентифікації, роботу з панеллю даних, управління пристроями, аналітику, систему сповіщень та налаштування. Кожен екран додатку супроводжується описом функціональних можливостей та рекомендаціями щодо ефективного використання.

Проведено оцінку ефективності системи за кількома ключовими критеріями, які підтвердили високу якість розробленого рішення. Показник втрати пакетів даних склав лише 2.3%, що значно нижче допустимого рівня у 5%, а успішність відновлення даних після втрати зв'язку досягла 98.7%, що перевищує цільове значення у 95%. Ці результати свідчать про надійність обраних технологій передачі даних та ефективність реалізованих механізмів локального зберігання та синхронізації.

Зручність інтерфейсу користувача, оцінена за методикою System Usability Scale (SUS), отримала 82 бали зі 100 можливих, що відповідає оцінці "відмінно" і перевищує цільове значення у 75 балів. Середній час виконання типових завдань склав 1.7 хвилини, що краще за цільовий показник у 2 хвилини, підтверджуючи інтуїтивну зрозумілість інтерфейсу та ефективність взаємодії користувача з системою.

ВИСНОВКИ

У результаті виконання дипломної роботи бакалавра було розроблено програмне забезпечення для системи віддаленого моніторингу стану ґрунту, яке дозволяє ефективно збирати, обробляти та візуалізувати дані про вологість і температуру ґрунту.

Проведено аналіз існуючих технологій для побудови систем віддаленого моніторингу ґрунту, що дозволило обрати оптимальний набір компонентів: ємнісний датчик вологості Capacitive Soil Moisture v1.2, цифровий датчик температури DS18B20, мікроконтролер ESP32, технології передачі даних WiFi та LoRaWAN. Обґрунтовано вибір програмного стеку на основі Node.js, Express, MongoDB Atlas для серверної частини та React Native з Expo для мобільного додатку.

Розроблено архітектуру програмного забезпечення, що включає п'ять основних рівнів: рівень сенсорів, рівень обробки даних, комунікаційний рівень, серверний рівень та рівень інтерфейсу користувача. Створено програмне забезпечення для мікроконтролера з особливою увагою до оптимізації енергоспоживання, що дозволяє системі працювати автономно протягом тривалого часу.

Реалізовано серверну частину системи з REST API для взаємодії з клієнтськими додатками та розроблено кросплатформний мобільний додаток з інтуїтивно зрозумілим інтерфейсом, що забезпечує зручну візуалізацію даних моніторингу, роботу з пристроями, аналітику та управління сповіщеннями.

Тестування розробленої системи підтвердило її високу ефективність: втрата пакетів даних складає лише 2.3%, показник успішності відновлення даних — 98.7%, зручність інтерфейсу за шкалою SUS — 82/100. Досягнуто значної економічної ефективності — вартість однієї станції моніторингу складає 1250 грн, що на 17% нижче цільового значення, роблячи систему доступною для малих фермерських господарств.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Грінгард С. Інтернет речей / С. Грінгард ; пер. з англ. – Київ : Дім книги, 2020. – 192 с.
2. Лі М. IoT-технології в агросекторі / М. Лі, Р. Чен. – Львів : Астролябія, 2021. – 204 с.
3. Сухарєв В.І. Безпроводні сенсорні мережі в моніторингу навколишнього середовища : монографія / В.І. Сухарєв. – Харків : ХНУРЕ, 2019. – 180 с.
4. Osadchyi O. S. Basics of Smart Agriculture / O.S. Osadchyi . – Kyiv : Yakaboo Publishing, 2020. – 220 p.
5. Espressif Systems. ESP32 Technical Reference Manual. – Version 5.3. – Shanghai: IBT Publishing, 2022. – 656 p.
6. Internet of Things for Smart Agriculture: A Survey / O. Friha, M. A. Ferrag, L. Shu, L. Maglaras, X. Wang // IEEE CAA Journal of Automatica Sinica. – 2021. – Vol. 8(4). – P. 718–752.
7. Sharma, N. Precision Agriculture Using IoT and WSN / N. Sharma, D. Suryavanshi // International Journal of Engineering Research and Technology. – 2020. – Vol. 9(6). – P. 123–128.
8. John, D. Soil Moisture Monitoring Using Low-Cost Sensors / D. John. – London : CRC Press, 2021. – 245 p.
9. Capacitive Soil Moisture Sensor v1.2 Datasheet. [Electronic resource]. – 2020. – Access mode: <https://wiki.dfrobot.com/>
10. Maxim Integrated. DS18B20 Digital Thermometer Datasheet. [Electronic resource]. – 2023. – Access mode: <https://www.analog.com/>
11. The Things Network. LoRaWAN Documentation. [Electronic resource]. – 2023. – Access mode: <https://www.thethingsnetwork.org/docs/>
12. MongoDB Inc. MongoDB Atlas Manual. [Electronic resource]. – 2024. – Access mode: <https://www.mongodb.com/docs/atlas/>

13. Firebase. Firebase Documentation. [Electronic resource]. – 2022. – Access mode: <https://firebase.google.com/docs>
14. Chart.js Official Documentation [Electronic resource]. – 2023. – Access mode: <https://www.chartjs.org/docs/latest/>
15. React Native Documentation. [Electronic resource]. – 2024. – Access mode: <https://reactnative.dev>
16. Kurniawan, A. Internet of Things Projects with ESP32 / A. Kurniawan. – Birmingham : Packt Publishing, 2020. – 402 p.
17. Farsi, M. Digital Agriculture: Case Studies in IoT / M. Farsi, R. M. Elhuni. – Berlin : Springer, 2022. – 356 p.
18. Козлов С.В. Програмування мікроконтролерів ESP з Arduino IDE / С.В. Козлов. – Київ : Арт-Книга, 2021. – 144 с.
19. Степаненко А.Ю. Побудова енергоефективних IoT-систем / А.Ю. Степаненко. – Одеса : ОНУ, 2020. – 152 с.
20. Node.js Documentation [Electronic resource]. – 2022. – Access mode: <https://nodejs.org/>
21. Express.js Guide [Electronic resource]. – 2023. – Access mode: <https://expressjs.com/>
22. Mosquitto MQTT Broker Documentation [Electronic resource]. – 2022. – Access mode: <https://mosquitto.org/>
23. DallasTemperature Arduino Library [Electronic resource]. – 2023. – Access mode: <https://github.com/milesburton>
24. Paul Stoffregen. OneWire Library for Arduino [Electronic resource]. – 2022. – Access mode: <https://github.com/PaulStoffregen>
25. Загурський О.М. Хмарні технології в аграрному секторі / О.М. Загурський, В.В. Луцик. – Тернопіль : ТНТУ, 2021. – 116 с.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення призначене для віддаленого моніторингу стану ґрунту з метою оптимізації використання води та добрив у сільському господарстві. Воно забезпечує збір даних про вологість і температуру ґрунту, їх передачу, обробку, зберігання та візуалізацію для кінцевих користувачів.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для віддаленого моніторингу стану ґрунту», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

А.2 Призначення розробки

Програмний продукт призначений для автоматизації процесу збору та аналізу даних про стан ґрунту, забезпечення доступу до цих даних через мобільний додаток та формування сповіщень про критичні зміни параметрів ґрунту.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для віддаленого моніторингу стану ґрунту забезпечує виконання таких функцій:

- збір даних про вологість та температуру ґрунту з використанням датчиків;
- передачу даних на сервер через бездротові технології (WiFi, LoRaWAN);

- зберігання та обробку даних на сервері;
- візуалізацію даних у мобільному додатку;
- аналіз даних та генерацію сповіщень про критичні стани;
- управління пристроями моніторингу та їх налаштуваннями.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення повинен бути зручним та інтуїтивно зрозумілим для користувачів різних рівнів технічної підготовки. Мобільний додаток має бути адаптивним для різних розмірів екранів та орієнтацій пристрою, підтримувати платформи iOS та Android. Інтерфейс повинен забезпечувати наочну візуалізацію даних у вигляді графіків та діаграм, а також зручну навігацію між різними розділами додатку.

A.3.3 Вимоги до надійності

Програмне забезпечення повинно забезпечувати надійне функціонування з коректною обробкою помилок.

A.3.4 Умови експлуатації

Апаратна частина системи має працювати в умовах сільськогосподарських угідь з температурним діапазоном від -10°C до $+50^{\circ}\text{C}$, вологістю до 95% та можливими опадами. Корпус пристрою повинен мати ступінь захисту не нижче IP65. Мобільний додаток має функціонувати на пристроях з операційними системами iOS 13.0+ та Android 8.0+.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для роботи програмного забезпечення потрібні такі технічні та програмні ресурси:

Апаратна частина:

- мікроконтролер ESP32;
- ємнісний датчик вологості ґрунту Capacitive Soil Moisture v1.2;
- цифровий датчик температури DS18B20;
- модуль WiFi (вбудований в ESP32);
- модуль LoRaWAN RFM95 (опціонально);
- літій-іонний акумулятор ємністю не менше 2000 мАг;
- сонячна панель потужністю не менше 2 Вт.

Серверна частина:

- хостинг з підтримкою Node.js;
- база даних MongoDB Atlas;
- MQTT-брокер Mosquitto;
- 512 МБ оперативної пам'яті;
- 10 ГБ дискового простору.

Клієнтська частина:

- смартфон з операційною системою iOS 13.0+ або Android 8.0+;
- доступ до інтернету (мобільний зв'язок або WiFi).

A.3.6 Вимоги до маркування і пакування

Апаратна частина системи віддаленого моніторингу стану ґрунту має бути упакована в захисний корпус зі ступенем захисту IP65, з маркуванням, що містить назву, серійний номер та контактну інформацію виробника.

Мобільний додаток має бути опублікований у відповідних магазинах додатків (Google Play Store, Apple App Store) з назвою "Soil Monitor" та відповідним описом функціональності.

A.4 Вхідні та вихідні дані

Вхідними даними до програмного забезпечення є:

- дані про вологість ґрунту з ємнісного датчика (у відсотках);
- дані про температуру ґрунту з цифрового датчика (у градусах

Цельсія);

– інформація про стан пристрою (рівень заряду батареї, якість сигналу);

- налаштування та параметри від користувача.

Вихідними даними програмного забезпечення є:

- візуалізація даних про стан ґрунту у вигляді графіків та діаграм;
- сповіщення про критичні зміни параметрів ґрунту;
- аналітичні звіти про стан ґрунту за обраний період;
- команди управління пристроями моніторингу.

ДОДАТОК Б
Фрагмент тексту програми

```

// 1. КОД МИКРОКОНТРОЛЕРА ESP32

// --- SensorManager.h ---
#ifndef SENSOR_MANAGER_H
#define SENSOR_MANAGER_H

#include <OneWire.h>
#include <DallasTemperature.h>

class SensorManager {
private:
    const int _moisturePin;
    const int _temperaturePin;
    const int _dryValue;
    const int _wetValue;

    OneWire* _oneWire;
    DallasTemperature* _sensors;

public:
    SensorManager(int moisturePin, int temperaturePin, int dryValue, int
wetValue);
    ~SensorManager();

    void begin();
    float getSoilMoisture();
    float getSoilTemperature();
};

#endif

// --- SensorManager.cpp ---
#include "SensorManager.h"

SensorManager::SensorManager(int moisturePin, int temperaturePin, int
dryValue, int wetValue)
    : _moisturePin(moisturePin), _temperaturePin(temperaturePin),
      _dryValue(dryValue), _wetValue(wetValue) {

```

```

    _oneWire = new OneWire(_temperaturePin);
    _sensors = new DallasTemperature(_oneWire);
}

SensorManager::~SensorManager() {
    delete _sensors;
    delete _oneWire;
}

void SensorManager::begin() {
    pinMode(_moisturePin, INPUT);
    _sensors->begin();
    _sensors->setResolution(12);
}

float SensorManager::getSoilMoisture() {
    int rawSum = 0;
    int samples = 10;

    // Виконуємо кілька вимірювань для зменшення шуму
    for (int i = 0; i < samples; i++) {
        rawSum += analogRead(_moisturePin);
        delay(10);
    }

    int rawAverage = rawSum / samples;

    // Застосовуємо калібрувальні коефіцієнти
    float moisture = map(rawAverage, _dryValue, _wetValue, 0, 100);

    // Обмежуємо значення в діапазоні 0-100%
    moisture = constrain(moisture, 0, 100);

    return moisture;
}

float SensorManager::getSoilTemperature() {
    _sensors->requestTemperatures();
    float temperature = _sensors->getTempCByIndex(0);
}

```

```

    if (temperature == DEVICE_DISCONNECTED_C) {
        return -127.0;  // Помилка - датчик не підключений
    }

    return temperature;
}

// --- CommunicationManager.h ---
#ifndef COMMUNICATION_MANAGER_H
#define COMMUNICATION_MANAGER_H

#include <WiFi.h>
#include <PubSubClient.h>

class CommunicationManager {
private:
    const char* _ssid;
    const char* _password;
    const char* _mqttServer;
    const int _mqttPort;
    const char* _mqttUser;
    const char* _mqttPassword;
    const char* _deviceId;

    WiFiClient _wifiClient;
    PubSubClient _mqttClient;

    String _moistureTopic;
    String _temperatureTopic;
    String _statusTopic;
    String _commandTopic;

public:
    CommunicationManager(const char* ssid, const char* password,
                        const char* mqttServer, int mqttPort,
                        const char* mqttUser, const char* mqttPassword,
                        const char* deviceId);

```

```

    void begin();
    bool connectWiFi();
    bool connectMQTT();
    bool sendMoistureData(float moisture, unsigned long timestamp);
    bool    sendTemperatureData(float    temperature,    unsigned    long
timestamp);
    bool sendStatusData(float batteryLevel, unsigned long uptime);
    void handleMqttMessage(char* topic, byte* payload, unsigned int
length);
    void loop();
};

#endif

// --- main.cpp ---
#include <Arduino.h>
#include "SensorManager.h"
#include "CommunicationManager.h"
#include "PowerManager.h"

// Піни для підключення датчиків
const int MOISTURE_PIN = 34;
const int TEMPERATURE_PIN = 15;

// Калібрувальні значення для датчика вологості
const int DRY_VALUE = 3200;
const int WET_VALUE = 1200;

// Налаштування WiFi та MQTT
const char* WIFI_SSID = "YourWiFiSSID";
const char* WIFI_PASSWORD = "YourWiFiPassword";
const char* MQTT_SERVER = "mqtt.example.com";
const int MQTT_PORT = 1883;
const char* MQTT_USER = "mqttuser";
const char* MQTT_PASSWORD = "mqttpassword";
const char* DEVICE_ID = "soil-monitor-001";

// Інтервал між вимірюваннями (у мілісекундах)
const unsigned long MEASUREMENT_INTERVAL = 300000;    // 5 хвилин

```

```

// Створення об'єктів
SensorManager sensorManager(MOISTURE_PIN, TEMPERATURE_PIN, DRY_VALUE,
WET_VALUE);
CommunicationManager commManager(WIFI_SSID, WIFI_PASSWORD,
MQTT_SERVER, MQTT_PORT, MQTT_USER, MQTT_PASSWORD, DEVICE_ID);
PowerManager powerManager;

// Змінні для відстеження часу
unsigned long lastMeasurementTime = 0;
unsigned long startupTime = 0;

void setup() {
    Serial.begin(115200);
    Serial.println("Soil Monitoring System Starting...");

    startupTime = millis();

    sensorManager.begin();
    commManager.begin();

    commManager.connectWiFi();
    commManager.connectMQTT();

    Serial.println("Setup completed");
}

void loop() {
    commManager.loop();

    unsigned long currentTime = millis();
    if (currentTime - lastMeasurementTime >= MEASUREMENT_INTERVAL) {
        Serial.println("Performing measurements...");

        float moisture = sensorManager.getSoilMoisture();
        float temperature = sensorManager.getSoilTemperature();
        float batteryLevel = powerManager.getBatteryLevel();

        Serial.print("Soil moisture: ");
    }
}

```

```

Serial.print(moisture);
Serial.println("%");

Serial.print("Soil temperature: ");
Serial.print(temperature);
Serial.println("°C");

commManager.sendMoistureData(moisture, currentTime);
commManager.sendTemperatureData(temperature, currentTime);
commManager.sendStatusData(batteryLevel, (currentTime -
startupTime) / 1000);

lastMeasurementTime = currentTime;

// Переходимо в режим глибокого сну до наступного вимірювання
if (MEASUREMENT_INTERVAL > 5000) {
    Serial.print("Going to sleep for ");
    Serial.print(MEASUREMENT_INTERVAL / 1000);
    Serial.println(" seconds");

    powerManager.deepSleep(MEASUREMENT_INTERVAL);
}
}
}

// 2. КОД СЕРВЕРНОЇ ЧАСТИНИ (NODE.JS)

// --- models/measurement.js ---
const mongoose = require('mongoose');

const measurementSchema = new mongoose.Schema({
  device_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Device',
    required: true,
    index: true
  },
  sensor_id: {
    type: mongoose.Schema.Types.ObjectId,

```

```

    ref: 'Sensor',
    required: true,
    index: true
  },
  value: {
    type: Number,
    required: true
  },
  unit: {
    type: String,
    required: true
  },
  timestamp: {
    type: Date,
    default: Date.now,
    index: true
  }
}, {
  timestamps: true
});

// Індекси для оптимізації запитів
measurementSchema.index({ device_id: 1, timestamp: -1 });
measurementSchema.index({ sensor_id: 1, timestamp: -1 });

const Measurement = mongoose.model('Measurement', measurementSchema);
module.exports = Measurement;

// --- mqtt/client.js ---
const mqtt = require('mqtt');
const config = require('../config');
const measurementService = require('../services/measurementService');
const logger = require('../utils/logger');

class MqttClient {
  constructor() {
    this.client = null;
    this.topicPattern = 'devices/+/sensors/+/measurements';
  }

```

```

connect() {
  const options = {
    clientId:
`soil_monitor_server_${Math.random().toString(16).substring(2, 8)}`,
    clean: true,
    username: config.mqtt.username,
    password: config.mqtt.password,
    reconnectPeriod: 5000
  };

  this.client = mqtt.connect(config.mqtt.url, options);

  this.client.on('connect', () => {
    logger.info('Connected to MQTT broker');
    this.client.subscribe(this.topicPattern, { qos: 1 });
  });

  this.client.on('message', this.handleMessage.bind(this));
}

async handleMessage(topic, message) {
  try {
    // Парсимо тему для отримання ідентифікаторів пристрою і датчика
    const topicParts = topic.split('/');
    const deviceId = topicParts[1];
    const sensorId = topicParts[3];

    // Парсимо JSON повідомлення
    const payload = JSON.parse(message.toString());

    // Створюємо об'єкт вимірювання
    const measurement = {
      device_id: deviceId,
      sensor_id: sensorId,
      value: payload.value,
      unit: payload.unit,
      timestamp: payload.timestamp ? new Date(payload.timestamp) :
new Date()

```

```

};

// Зберігаємо вимірювання через сервіс
await measurementService.createMeasurement(measurement);

// Перевіряємо умови для створення сповіщень
await measurementService.checkAlertConditions(measurement);

} catch (error) {
  logger.error('Error processing MQTT message:', error);
}
}
}

const mqttClient = new MqttClient();
module.exports = mqttClient;

// --- controllers/measurementController.js ---
const Measurement = require('../models/measurement');
const mongoose = require('mongoose');

// Отримання вимірювань з фільтрацією та пагінацією
exports.getMeasurements = async (req, res) => {
  try {
    const {
      device_id,
      sensor_id,
      from,
      to,
      limit = 100,
      page = 1
    } = req.query;

    // Формування фільтра
    let filter = {};

    if (device_id) {
      filter.device_id = mongoose.Types.ObjectId(device_id);
    }
  }

```

```

if (sensor_id) {
  filter.sensor_id = mongoose.Types.ObjectId(sensor_id);
}

// Фільтрація за часом
if (from || to) {
  filter.timestamp = {};

  if (from) {
    filter.timestamp.$gte = new Date(from);
  }

  if (to) {
    filter.timestamp.$lte = new Date(to);
  }
}

// Розрахунок параметрів пагінації
const skip = (page - 1) * limit;

// Запит без агрегації
const measurements = await Measurement.find(filter)
  .sort({ timestamp: -1 })
  .skip(skip)
  .limit(parseInt(limit))
  .populate('device_id', 'name location')
  .populate('sensor_id', 'name type unit');

// Підрахунок загальної кількості записів для пагінації
const total = await Measurement.countDocuments(filter);

res.status(200).json({
  data: measurements,
  pagination: {
    total,
    page: parseInt(page),
    limit: parseInt(limit),
    pages: Math.ceil(total / limit)
  }
});

```

```

    }
  });
} catch (error) {
  console.error('Error fetching measurements:', error);
  res.status(500).json({ error: 'Internal server error' });
}
};

// 3. КОД МОБІЛЬНОГО ДОДАТКУ (REACT NATIVE)

// --- components/charts/MeasurementChart.js ---
import React, { useState, useEffect } from 'react';
import { View, StyleSheet, Dimensions } from 'react-native';
import { LineChart } from 'react-native-chart-kit';
import { Card, Title, Chip, Text, ActivityIndicator } from 'react-native-paper';
import { useSelector, useDispatch } from 'react-redux';
import { fetchMeasurementStats } from '../store/slices/measurementSlice';

const MeasurementChart = ({ deviceId, sensorId, sensorType, unit }) => {
  {
    const dispatch = useDispatch();
    const [period, setPeriod] = useState('day');
    const { stats, loading, error } = useSelector(state => state.measurements);

    useEffect(() => {
      if (deviceId && sensorId) {
        dispatch(fetchMeasurementStats({ deviceId, sensorId, period }));
      }
    }, [deviceId, sensorId, period, dispatch]);

    const renderChart = () => {
      if (loading) {
        return (
          <View style={styles.loadingContainer}>
            <ActivityIndicator size="large" color="#3498db" />
          </View>

```

```

    );
}

if (!stats || stats.length === 0) {
    return (
        <View style={styles.noDataContainer}>
            <Text>No data available for the selected period</Text>
        </View>
    );
}

// Підготовка даних для графіка
const labels = stats.map(item => item.label);
const data = stats.map(item => item.value);

const chartConfig = {
    backgroundColor: '#ffffff',
    backgroundGradientFrom: '#ffffff',
    backgroundGradientTo: '#ffffff',
    color: (opacity = 1) => `rgba(52, 152, 219, ${opacity})`,
    strokeWidth: 2,
    decimalPlaces: 1,
};

const chartData = {
    labels,
    datasets: [
        {
            data,
            color: (opacity = 1) => `rgba(52, 152, 219, ${opacity})`,
            strokeWidth: 2
        }
    ],
    legend: [`${sensorType.charAt(0).toUpperCase()} +
sensorType.slice(1)} (${unit})`]
};

return (
    <LineChart

```

```

        data={chartData}
        width={Dimensions.get('window').width - 32}
        height={220}
        chartConfig={chartConfig}
        bezier
        style={styles.chart}
        yAxisSuffix={unit}
        fromZero
      />
    );
  };

  return (
    <Card style={styles.container}>
      <Title style={styles.title}>
        {sensorType.charAt(0).toUpperCase()} + {sensorType.slice(1)}
      </Title>

      <View style={styles.periodContainer}>
        <Text style={styles.periodLabel}>Period:</Text>
        <View style={styles.chipContainer}>
          <Chip
            selected={period === 'day'}
            onPress={() => setPeriod('day')}
            style={styles.chip}
          />
          Day
        </Chip>
        <Chip
            selected={period === 'week'}
            onPress={() => setPeriod('week')}
            style={styles.chip}
          />
          Week
        </Chip>
        <Chip
            selected={period === 'month'}
            onPress={() => setPeriod('month')}

```

```

        style={styles.chip}
      >
        Month
      </Chip>
    </View>
  </View>

  <View style={styles.chartContainer}>
    {renderChart()}
  </View>
</Card>
);
};

```

```

const styles = StyleSheet.create({
  container: {
    padding: 16,
    marginVertical: 8,
    borderRadius: 8,
  },
  title: {
    marginBottom: 16,
    textAlign: 'center',
  },
  periodContainer: {
    marginBottom: 16,
  },
  periodLabel: {
    fontSize: 14,
    fontWeight: 'bold',
    marginBottom: 4,
  },
  chipContainer: {
    flexDirection: 'row',
    flexWrap: 'wrap',
  },
  chip: {
    marginRight: 8,
    marginBottom: 8,
  },

```

```

    },
    chartContainer: {
      alignItems: 'center',
    },
    chart: {
      marginVertical: 8,
      borderRadius: 16,
    },
    loadingContainer: {
      height: 220,
      justifyContent: 'center',
      alignItems: 'center',
    },
    noDataContainer: {
      height: 220,
      justifyContent: 'center',
      alignItems: 'center',
    },
  }));

export default MeasurementChart;

// --- screens/devices/DeviceDetailsScreen.js ---
import React, { useEffect } from 'react';
import { View, ScrollView, StyleSheet, RefreshControl } from 'react-native';
import { useSelector, useDispatch } from 'react-redux';
import { Card, Title, Paragraph } from 'react-native-paper';
import { useRoute } from '@react-navigation/native';
import { fetchDeviceDetails } from '../../../store/slices/deviceSlice';
import { fetchSensorsByDevice } from '../../../store/slices/sensorSlice';
import MeasurementChart from '../../../components/charts/MeasurementChart';
import DeviceStatusCard from '../../../components/devices/DeviceStatusCard';

const DeviceDetailsScreen = () => {
  const route = useRoute();
  const dispatch = useDispatch();

```

```

const { deviceId } = route.params;
const [refreshing, setRefreshing] = React.useState(false);

const { device } = useSelector(state => state.devices);
const { sensors } = useSelector(state => state.sensors);

useEffect(() => {
  if (deviceId) {
    dispatch(fetchDeviceDetails(deviceId));
    dispatch(fetchSensorsByDevice(deviceId));
  }
}, [deviceId, dispatch]);

const onRefresh = async () => {
  setRefreshing(true);
  await Promise.all([
    dispatch(fetchDeviceDetails(deviceId)),
    dispatch(fetchSensorsByDevice(deviceId))
  ]);
  setRefreshing(false);
};

return (
  <ScrollView
    style={styles.container}
    refreshControl={
      <RefreshControl
        refreshing={refreshing}
        onRefresh={onRefresh}
      />
    }
  >
    {device && (
      <Card style={styles.infoCard}>
        <Card.Content>
          <Title>{device.name}</Title>
          <Paragraph>Location: {device.location}</Paragraph>

```

```

        <Paragraph>Last                                updated:
Date(device.lastSeen).toLocaleString()</Paragraph>
    </Card.Content>
</Card>
    )}

    {device && <DeviceStatusCard device={device} />}

    {sensors.map(sensor => (
        <MeasurementChart
            key={sensor._id}
            deviceId={deviceId}
            sensorId={sensor._id}
            sensorType={sensor.type}
            unit={sensor.unit}
        />
    )})}
</ScrollView>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 16,
    backgroundColor: '#f5f5f5',
  },
  infoCard: {
    marginBottom: 16,
    borderRadius: 8,
  },
});

export default DeviceDetailsScreen;

// --- store/slices/measurementSlice.js ---
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import { measurementService } from '../../services';

```

```

export const fetchMeasurements = createAsyncThunk(
  'measurements/fetchMeasurements',
  async (params, { rejectWithValue }) => {
    try {
      return await measurementService.getMeasurements(params);
    } catch (error) {
      return rejectWithValue(error.message);
    }
  }
);

export const fetchMeasurementStats = createAsyncThunk(
  'measurements/fetchMeasurementStats',
  async ({ deviceId, sensorId, period, metric }, { rejectWithValue })
=> {
    try {
      return await measurementService.getMeasurementStats(deviceId,
sensorId, period, metric);
    } catch (error) {
      return rejectWithValue(error.message);
    }
  }
);

const measurementSlice = createSlice({
  name: 'measurements',
  initialState: {
    measurements: [],
    stats: [],
    loading: false,
    error: null,
    pagination: {
      total: 0,
      page: 1,
      limit: 20,
      pages: 0
    }
  },
  reducers: {},

```

```

extraReducers: (builder) => {
  builder
    .addCase(fetchMeasurements.pending, (state) => {
      state.loading = true;
      state.error = null;
    })
    .addCase(fetchMeasurements.fulfilled, (state, action) => {
      state.loading = false;
      state.measurements = action.payload.data;
      state.pagination = action.payload.pagination;
    })
    .addCase(fetchMeasurements.rejected, (state, action) => {
      state.loading = false;
      state.error = action.payload;
    })
    .addCase(fetchMeasurementStats.pending, (state) => {
      state.loading = true;
      state.error = null;
    })
    .addCase(fetchMeasurementStats.fulfilled, (state, action) => {
      state.loading = false;
      state.stats = action.payload;
    })
    .addCase(fetchMeasurementStats.rejected, (state, action) => {
      state.loading = false;
      state.error = action.payload;
    });
}
});

export default measurementSlice.reducer;

// --- services/measurementService.js ---
import api from './api';

export const measurementService = {
  getMeasurements: async (params) => {
    const response = await api.get('/measurements', { params });
    return response.data;
  }
};

```

```

    },

    getMeasurementStats: async (deviceId, sensorId, period, metric) => {
      const params = {
        device_id: deviceId,
        sensor_id: sensorId,
        period,
        metric
      };

      const response = await api.get('/measurements/stats', { params });
      return response.data;
    },

    exportMeasurements: async (deviceId, period) => {
      const params = {
        device_id: deviceId,
        period,
        format: 'csv'
      };

      const response = await api.get('/measurements/export', {
        params,
        responseType: 'blob'
      });

      return response.data;
    }
  };

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для віддаленого моніторингу стану ґрунту.

Студент, групи КНТ-151
Керівник проєкту, к.т.н, доцент

Микита КОНОВАЛОВ
Олександр СТЕПАНЕНКО

Рисунок В.1 – Слайд 1

Об'єкт, предмет, мета та задачі

Об'єкт дослідження — процес розробки програмного забезпечення для систем дистанційного збору та обробки даних про стан ґрунту.

Предмет дослідження — методи та програмні засоби для отримання, передачі, обробки та візуалізації даних про стан ґрунту в системах віддаленого моніторингу.

Мета роботи — підвищення ефективності процесу збору та аналізу даних про стан ґрунту шляхом розробки програмного забезпечення для віддаленого моніторингу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- ▶ провести аналіз існуючих технологій для побудови систем віддаленого моніторингу ґрунту та обґрунтувати вибір оптимальних рішень;
- ▶ розробити архітектуру програмного забезпечення для системи віддаленого моніторингу стану ґрунту;
- ▶ спроектувати та реалізувати програмне забезпечення для мікроконтролера, що забезпечує збір даних з датчиків та їх передачу;
- ▶ створити серверну частину для обробки та зберігання даних;
- ▶ розробити мобільний додаток для візуалізації даних та сповіщення користувачів;
- ▶ провести тестування розробленої системи та оцінити її ефективність.

2

Рисунок В.2 – Слайд 2

Порівняння характеристик комерційних систем моніторингу ґрунту

Назва системи	Виробник	Вимірювані параметри	Технологія передачі даних	Автономність	Вартість
Soil Scout	Soil Scout Ltd (Фінляндія)	Вологість, температура, електропровідність	Радіочастотна передача (433 МГц)	До 20 років	Висока
EnviroSCAN	Sentek Technologies (Австралія)	Багаторівневий профіль вологості	GSM/GPRS	До 5 років	Висока
Плантер	Krau Technologies (Україна)	Вологість, температура	LoRaWAN	До 3 років	Середня
AquaCheck	AquaCheck Ltd (ПАР)	Вологість, температура, солоність	GSM/GPRS	До 5 років	Висока
TEROS	METER Group (США)	Вологість, температура, електропровідність	ZigBee	До 5 років	Висока

Рисунок В.3 – Слайд 3

Порівняльний аналіз датчиків для моніторингу ґрунту

Тип датчика	Модель	Діапазон вимірювань	Точність	Інтерфейс	Енергоспоживання	Вартість
Вологість (резистивний)	YL-69	0-100%	Низька	Аналоговий	Низьке	Низька
Вологість (ємнісний)	Capacitive Soil Moisture v1.2	0-100%	Середня	Аналоговий	Низьке	Низька
Вологість (професійний)	TEROS 10	0-100%	Висока	SDI-12	Середнє	Висока
Температура	DS18B20	-55 °C до +125 °C	±0.5 °C	1-Wire	Низьке	Низька
pH	SEN0161	pH 0-14	±0.1 pH	Аналоговий	Середнє	Середня
Електропровідність	DFR0300	0-20 мС/см	±10%	Аналоговий	Середнє	Середня
NPK	SEN0463	N, P, K (5 рівнів)	Середня	UART	Високе	Висока

Рисунок В.4 – Слайд 4

Аналіз мікроконтролерів для систем моніторингу ґрунту

Мікроконтролер	Активний режим (мА)	Режим сну (мкА)	Режим глибокого сну (мкА)	Час пробудження (мс)
ESP32	240	20	10	45
Arduino Nano 33 IoT	15	5	1	8
Raspberry Pi Pico W	25	2.4	0.7	15

Інтерфейс	ESP32	Arduino Nano 33 IoT	Raspberry Pi Pico W
GPIO	34	14	26
ADC	12-біт, 18 каналів	12-біт, 8 каналів	12-біт, 4 канали
DAC	8-біт, 2 канали	10-біт, 1 канал	Немає
I2C	2	1	2
SPI	3	1	2
UART	3	1	2
WiFi	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n
Bluetooth	BLE 4.2	BLE 4.2	Немає

5

Рисунок В.5 – Слайд 5

Порівняння серверних технологій для систем моніторингу ґрунту

Технологія	Тип	Переваги	Недоліки	Ліміти безкоштовного плану
Node.js (Express)	Серверний фреймворк	Асинхронність, швидкість, простота	Менше бібліотек для аналізу даних	Безкоштовний
Python (Flask)	Серверний фреймворк	Простий синтаксис, наукові бібліотеки	Нижча продуктивність для асинхронних операцій	Безкоштовний
MongoDB Atlas	База даних	Гнучка схема, простота використання	Обмежені можливості для складних запитів	512 МБ, 100 з'єднань
PostgreSQL	База даних	Потужна SQL-підтримка, надійність	Складніше налаштування	Залежить від хостингу
Mosquitto	MQTT брокер	Легкий, низькі вимоги до ресурсів	Обмежена масштабованість	Безліміт
Firebase Realtime DB	База даних + API	Простота інтеграції, реальний час	Обмежені можливості запитів	1 ГБ сховища, 10 ГБ трафіку
Heroku	Хостинг	Простота розгортання, інтеграція з Git	Сплячий режим для безкоштовних додатків	550-1000 годин роботи на місяць
Netlify	Хостинг	Безперервна інтеграція, CDN	Обмежені обчислювальні ресурси	100 ГБ трафіку, 300 хвилин сборки
Vercel	Хостинг	Оптимізація для React/Next.js, глобальний CDN	Обмежені типи серверних функцій	100 ГБ трафіку, обмеження на виклики функцій

Рисунок В.6 – Слайд 6

Порівняння технологій розробки мобільних додатків

Технологія	Мова програмування	Продуктивність	Інтерфейс користувача	Швидкість розробки	Спільнота та підтримка	Розмір додатку
React Native	JavaScript	Висока	Нативний вигляд	Висока	Велика спільнота, багато бібліотек	Середній
Flutter	Dart	Дуже висока	Кастомні віджети	Висока	Зростаюча спільнота, офіційна підтримка Google	Середній
Expo (на базі React Native)	JavaScript	Висока	Нативний вигляд	Дуже висока	Підтримка Expo, доступ до спільноти React Native	Великий
Kotlin (Android)	Kotlin	Найвища	Повний доступ до нативних компонентів	Середня	Офіційна підтримка Google	Малий
Swift (iOS)	Swift	Найвища	Повний доступ до нативних компонентів	Середня	Офіційна підтримка Apple	Малий

Рисунок В.7 – Слайд 7

Обрані технології для розробки системи віддаленого моніторингу стану ґрунту

Компонент системи	Обрана технологія	Обґрунтування вибору
Датчик вологості	Capactive Soil Moisture v1.2	Достатня точність, стійкість до корозії, низька вартість
Датчик температури	DS18B20	Висока точність, водонепроникний корпус, низьке енергоспоживання
Мікроконтролер	ESP32	Вбудована підтримка WiFi та BLE, ефективні режими енергозбереження, велика спільнота
Технологія передачі даних	WiFi / LoRaWAN	Комбінований підхід для різних умов експлуатації
Серверна платформа	Node.js + Express	Висока продуктивність для асинхронних запитів, простота розробки
База даних	MongoDB Atlas	Гнучка схема, безкоштовний план 512 МБ
MQTT брокер	Mosquitto	Відкритий код, низькі вимоги до ресурсів
Хостинг	Netlify	Безкоштовний план 100 ГБ трафіку, проста інтеграція з Git
Фреймворк для мобільного додатку	React Native + Expo	Кросплатформність, швидкість розробки, велика спільнота
Візуалізація даних	Chart.js	Простий API, підтримка різних типів графіків
Аутентифікація та сповіщення	Firebase	Простота інтеграції, безкоштовний план, надійна інфраструктура

Рисунок В.8 – Слайд 8

Блок-схема алгоритму збору даних з датчиків



Рисунок В.9 – Слайд 9

Діаграма класів програмного забезпечення мікроконтролера

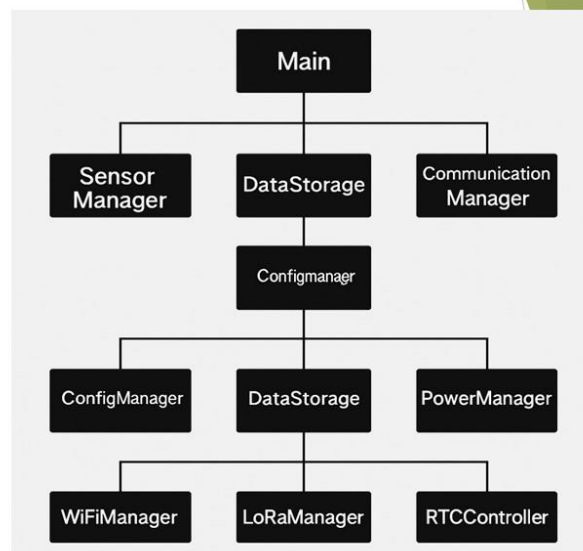
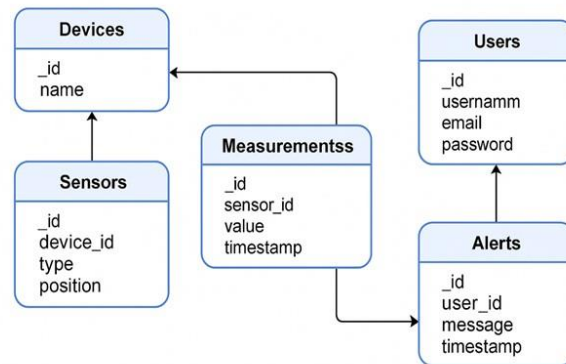


Рисунок В.10 – Слайд 10

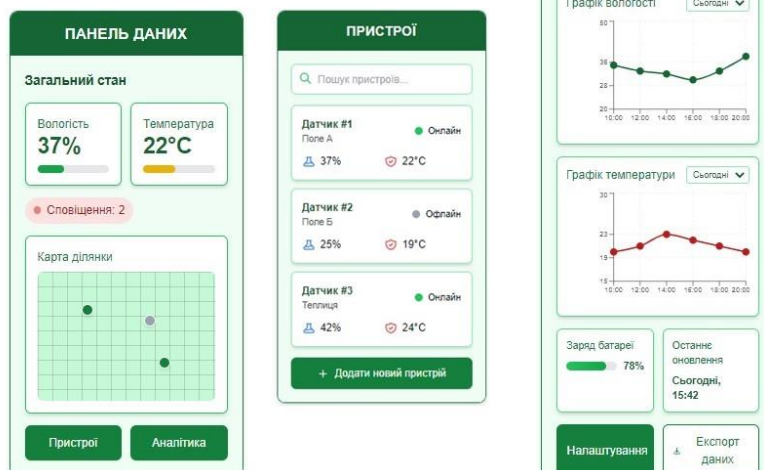
Схема бази даних системи віддаленого моніторингу стану ґрунту



11

Рисунок В.11 – Слайд 11

Інтерфейс



12

Рисунок В.12 – Слайд 12

Інтерфейс

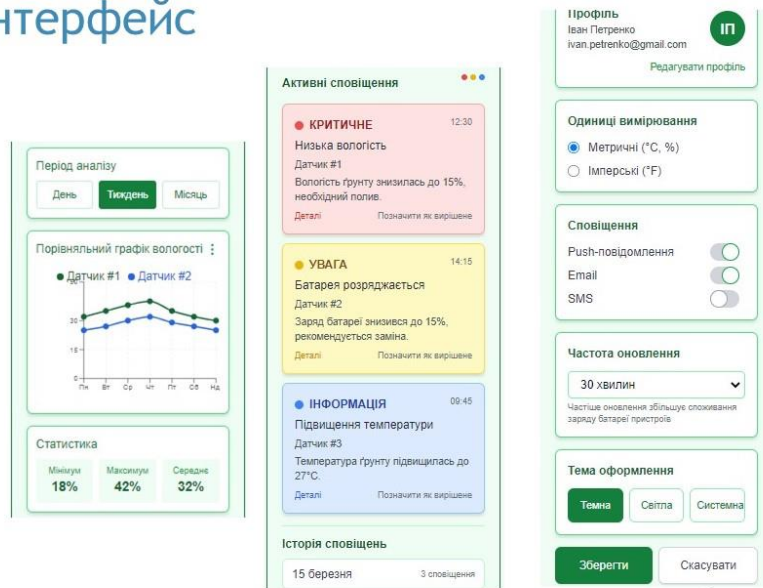


Рисунок В.13 – Слайд 13

Оцінка ефективності системи моніторингу ґрунту

Критерій	Метрика	Результат	Цільове значення
Надійність передачі даних	Втрата пакетів даних	2.3%	<5%
Надійність передачі даних	Успішність відновлення даних	98.7%	>95%
Зручність інтерфейсу	SUS (System Usability Scale)	82/100	>75/100
Зручність інтерфейсу	Час виконання типових завдань	1.7 хв	<2 хв
Економічна ефективність	Вартість однієї станції	1250 грн	<1500 грн

Рисунок В.14 – Слайд 14

ВИСНОВКИ

- ▶ Розроблено систему віддаленого моніторингу стану ґрунту з використанням технологій IoT
- ▶ Система включає апаратну частину з датчиками вологості та температури, серверну частину на базі Node.js і MongoDB, та кросплатформний мобільний додаток на React Native
- ▶ Досягнуто високу надійність передачі даних (втрата пакетів лише 2.3%) та зручність інтерфейсу (оцінка SUS 82/100) Забезпечено енергоефективність системи через оптимізацію роботи мікроконтролера та використання сонячної панелі
- ▶ Досягнуто економічну ефективність – вартість станції моніторингу 1250 грн, що на 17% нижче планової
- ▶ Система дозволяє оптимізувати використання води та добрив у сільському господарстві, підвищуючи врожайність та знижуючи вплив на довкілля

15

Рисунок В.15 – Слайд 15