Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ___другий (магістерський)_____
(рівень вищої освіти)
_____Sensor-Cloud Computing у системі біоресурсів_____
_____Sensor-Cloud Computing in the Bioresource System_____
(тема)

Виконав: студент 2 курсу, групи СКСм-24-1

_____Мовчан К.М._____
(прізвище, ініціали)

Спеціальність __123 Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми ___Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма
__Спеціалізовані комп'ютерні системи
(повна назва освітньої програми)

Керівник _____проф. Чумаченко С.В._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри (підпис) проф. Чумаченко С.В.
(прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123_ Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри __ _____
(підпис)
«_19_»_____12_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Мовчан Ксенії Миколаївні
(прізвище, ім'я, по батькові)

1. Тема роботи Sensor-Cloud Computing у системі біоресурсів
затверджена наказом по університету від 07.11. 2025 р. № 1012 Ст
2. Термін подання студентом роботи до екзаменаційної комісії _19_ ____12____ 2025 р.
3. Вихідні дані до роботи PlatformIO
ESP32
Платформа Android Studio
Датчики
Flutter

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області
Аналіз технологій комунікації та вибір протоколу передачі даних.
Архітектура програмно-апаратного комплексу
Програмна реалізація вбудованої логіки
Програмна реалізація застосунку та тестування

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) __14_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання _07.11.2025_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	07.11.2025	
2	Аналіз предметної області	07.11.2025 – 17.11.2025	
3	Вибір інструментальних засобів та розробка	18.11.2025 – 25.11.2025	
4	Розробка програми	26.11.2025 – 30.11.2025	
5	Програмна реалізація	01.12.2025 – 07.12.2025	
6	Тестування розробленої системи	08.12.2025 – 12.12.2025	
7	Оформлення пояснювальної записки	13.12.2025 – 15.12.2025	
8	Оформлення графічного матеріалу	16.12.2025 – 17.12.2025	
9	Перевірка виконаного проекту керівником	18.12.2025 – 19.12.2025	
10	Захист роботи	23.12.2025	

Студент _____
(підпис)

Керівник роботи _____ ____проф. Чумаченко С.В._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 60 сторінок, 9 рисунків, 1 таблицю, 14 лістингів, 17 джерел за переліком посилань.

ESP32, PlatformIO, MQTT, БАЗА ДАНИХ, ДАТЧИКИ, БІОРЕСУРСИ, КЛІМАТ, ІНТЕРНЕТ РЕЧЕЙ, ОС ANDROID, ЗАСТОСУНОК.

Дана дипломна робота присвячена вирішенню актуальної науково-практичної задачі автоматизації процесів у закритому ґрунті (теплицях) в умовах нестабільного інтернет-з'єднання. Об'єктом дослідження є процеси керування мікрокліматом біотехнічних об'єктів. Предметом дослідження є програмно-апаратний комплекс на базі мікроконтролера ESP32, що реалізує концепцію "Offline-First".

У роботі проведено аналіз існуючих архітектурних рішень IoT, обґрунтовано неефективність класичної хмарної моделі (Cloud-centric) для критичних систем життєзабезпечення рослин у сільській місцевості.

Практична цінність роботи полягає у створенні повністю функціонального прототипу системи, що інтегрує сенсори BME280, ємнісні датчики вологості ґрунту та 4-канальний блок реле. Програмне забезпечення, розроблене у середовищі PlatformIO (VS Code) з використанням FreeRTOS, фреймворк від виробника Espressif – ESP-IDF забезпечує багатозадачність та асинхронну обробку подій. Клієнтська частина реалізована на кросплатформному фреймворку Flutter, що дозволяє керувати системою як через глобальну мережу (MQTT), так і локально.

ABSTRACT

The explanatory note contains 60 pages, 9 figures, 1 table, 14 listings, and 17 sources in the list of references.

ESP32, PlatformIO, MQTT, DATABASE, SENSORS, BIORESOURCES, CLIMATE, INTERNET OF THINGS, ANDROID OS, APP.

This thesis is devoted to solving the urgent scientific and practical problem of automating processes in closed ground (greenhouses) in conditions of unstable Internet connection. The object of research is the processes of microclimate control in biotechnical facilities. The subject of the study is a software and hardware complex based on the ESP32 microcontroller, which implements the “Offline-First” concept.

The work provides an analysis of existing IoT architectural solutions and substantiates the inefficiency of the classic cloud-centric model for critical plant life support systems in rural areas.

The practical value of the work lies in the creation of a fully functional prototype system that integrates BME280 sensors, capacitive soil moisture sensors, and a 4-channel relay unit. The software, developed in the PlatformIO (VS Code) environment using FreeRTOS, a framework from Espressif – ESP-IDF, provides multitasking and asynchronous event processing. The client part is implemented on the cross-platform Flutter framework, which allows you to control the system both via a global network (MQTT) and locally.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення;

API – інтерфейс програмування додатків (англ. Application Programming Interface);

HTTP – протокол передачі гіпертексту (англ. Hypertext Transfer Protocol);

IoT – інтернет речей (англ. Internet of Things);

SDK – комплект засобів розробки програмного забезпечення (англ. Software Development Kit).

ВСТУП

Стрімкий розвиток технологій "Інтернету речей" (Internet of Things, IoT) трансформує аграрний сектор, перетворюючи його на високотехнологічну галузь "Precision Agriculture" (точне землеробство). Забезпечення продовольчої безпеки та підвищення рентабельності тепличних господарств вимагає переходу від інтуїтивного керування до методів, що базуються на об'єктивних даних (Data-Driven). Однак, специфіка українського агросектору, зокрема, географічна віддаленість теплиць та недосконаліть телекомунікаційної інфраструктури, накладає суттєві обмеження на використання стандартних IoT-рішень.

Класичні системи моніторингу, представлені на ринку (наприклад, аматорські рішення на базі Xiaomi або професійні хмарні станції), часто будуються за архітектурою "тонкий клієнт". У такій моделі датчики лише передають "сирі" дані на сервер, де приймається рішення про увімкнення поливу чи вентиляції. Це створює критичну точку відмови: при втраті зв'язку теплиця стає некерованою, що може призвести до втрати врожаю за лічені години (наприклад, через перегрів у сонячний день). Аналіз комерційних рішень, таких як Xiaomi Mi Flora або GreenIQ, демонструє їхню орієнтацію на стабільний домашній Wi-Fi та закриті екосистеми, що унеможливлює їх гнучке використання у професійних задачах.

Метою роботи є суттєве підвищення якості цифрового моніторингу та управління біоресурсами шляхом створення відмовостійкої, автономної архітектури системи керування теплицею на підставі Sensor-Cloud Computing, яка нівелює залежність від стабільності інтернет-каналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Стрімкий розвиток концепції Індустрії 4.0 та її експансія в аграрний сектор сформували новий технологічний уклад – Сільське господарство 4.0 (Agriculture 4.0). Ключовим драйвером цього процесу є перехід від ізольованих автоматизованих систем до глобально підключених інтелектуальних мереж. У цьому контексті Sensor-Cloud Computing виступає як еволюційний наступник класичних бездротових сенсорних мереж (WSN), пропонуючи вирішення проблем обмежених обчислювальних ресурсів та енергозалежності польових вузлів за рахунок інтеграції з хмарними обчисленнями.

1.1 Концепція Інтернету Речей (IoT) у точному землеробстві

Інтернет речей (IoT) – це система взаємопов'язаних обчислювальних пристроїв, сенсорів та механічних вузлів, які можуть збирати та передавати дані через бездротові мережі без прямого втручання людини. У контексті сільського господарства (часто згадується як "точне землеробство" або "Smart Farming"), IoT використовується для оптимізації продуктивності, зменшення відходів та інтенсифікації операцій.

Збираючи дані в реальному часі про якість ґрунту, вологість, температуру повітря, ріст рослин та кліматичні умови, фермери можуть приймати обґрунтовані рішення щодо того, коли сіяти, удобрювати, зрошувати та збирати врожай. Технології IoT дозволяють підвищити продуктивність, оптимізувати використання ресурсів та зменшити вуглецевий слід.

1.2 Sensor-Cloud Computing

Сенсорно-хмарні обчислення розглядаються як одна з перспективних технологій для систем моніторингу в сільському господарстві та охороні здоров'я.

Сенсорно-хмарні обчислення – це нова модель для Cloud Computing, яка використовує фізичні датчики для збору даних і передачі всіх сенсорних даних в інфраструктуру хмарних обчислень. Вона також ефективно контролює дані датчиків, що використовується для багатьох додатків моніторингу.

Хмарні обчислення – це одна з найпопулярніших технологій, що розвиваються, нова і перспективна парадигма, яка надає обчислення як утиліту [1]. Вона забезпечує використання програмного забезпечення, доступ до даних, послуги зберігання даних та інші обчислення через Інтернет і дозволяє клієнтам орендувати ресурси на основі моделі «платити по мірі використання».

Сенсорно-хмарна архітектура (рис. 1.1) концептуально інтегрує хмарну інфраструктуру з сенсорними мережами, що дозволяє здійснювати моніторинг в реальному часі додатків з інтенсивним використанням даних, які зазвичай розподілені в географічно розподілених місцях [1].

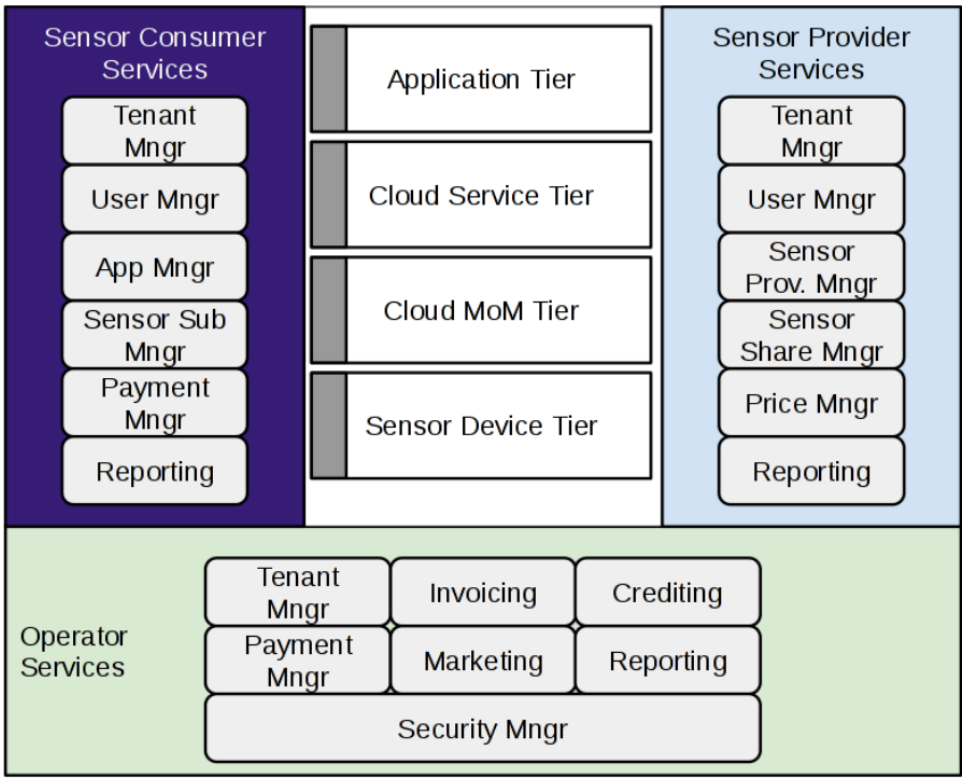


Рисунок 1.1 – SSC Architecture Overview

Переваги SCC:

– обчислювальна потужність: хмарні сервери можуть обробляти величезні масиви даних (Big Data) та виконувати складні завдання, такі як тренування моделей машинного навчання (ML) для прогнозування врожайності або ідентифікації хвороб[2];

– масштабованість та зберігання: хмара надає практично необмежені ресурси для зберігання історичних даних (погоди, стану ґрунту, врожайності за минулі роки), що є критичним для довгострокового стратегічного аналізу;[5]

– доступність: користувачі можуть отримувати доступ до даних та керувати системою з будь-якої точки світу через веб-інтерфейс або мобільний додаток.

1.3 Edge computing

Незважаючи на переваги SCC, чиста хмарна модель має фундаментальний недолік, особливо критичний для сільського господарства: тотальну залежність від стабільного інтернет-з'єднання [3]. У реальних умовах ферм, розташованих у віддаленій місцевості, зв'язок часто є ненадійним, дорогим або відсутнім[4].

Граничні обчислення – це платформа розподілених обчислень, яка наближає корпоративні додатки до джерел даних, таких як пристрої Інтернету речей або локальні периферійні сервери.

Така близькість до джерел даних може забезпечити значні переваги для бізнесу, включаючи більш швидке отримання інформації, поліпшення часу відгуку і кращу доступність пропускної здатності.

Надсилання всіх даних, згенерованих пристроями, до централізованого центру обробки даних або хмари викликає проблеми з пропускною здатністю та затримками.

Переваги Edge Computing:

– низька затримка (Low Latency): обробка даних на місці дозволяє приймати рішення миттєво. Це критично для автономних систем, таких як трактор, що має негайно реагувати на перешкоди, або система, що має екстрено увімкнути вентиляцію при перегріві [5];

– автономна робота (Offline Functionality): система продовжує повноцінно функціонувати (збирати дані, керувати поливом та вентиляцією) навіть за повної відсутності інтернет-з'єднання[5];

– конфіденційність та безпека: чутливі дані обробляються локально,

зменшуючи ризики перехоплення під час передачі до хмари.[5]

Граничні обчислення пропонують більш ефективну альтернативу – дані обробляються і аналізуються ближче до місця їх створення. Оскільки дані не передаються через мережу до хмари або центру обробки даних для обробки, затримки зменшуються.

1.4 Гібридна модель "Edge-Cloud" як оптимальна архітектура

Ані чистий "Cloud", ані чистий "Edge" не є ідеальними. Сучасні відмовостійкі системи у сільському господарстві використовують гібридну модель, яка поєднує найкраще з обох підходів.

Ця архітектура функціонує наступним чином:

– рівень Edge (Локальний вузол/Шлюз): виконує миттєву обробку даних, фільтрацію шумів, прийняття рішень у реальному часі (наприклад, увімкнення поливу) та локальну буферизацію даних;

– рівень Cloud (Хмарна платформа): використовується для довгострокового зберігання агрегованих даних, виконання складних аналітичних завдань (ML), тренування моделей та надання глобального доступу до дашбордів[2].

1.5 Low-power wide-area network

Для передачі даних від сенсорів, особливо на великих територіях (поля, а не одна теплиця), традиційні технології, як-от Wi-Fi, є неефективними через

високе енергоспоживання та малий радіус дії. Тут вступають у гру мережі Low Power Wide Area Networks (LPWAN).

LPWAN – це клас бездротових технологій, розроблених спеціально для IoT-пристроїв, що живляться від батарей та потребують передачі невеликих обсягів даних на великі відстані (до 10-15 км у сільській місцевості). Найвідомішими протоколами в цьому класі є LoRaWAN та Zigbee (хоча Zigbee має менший радіус дії і більше підходить для локальних mesh-мереж)[6]. Ця технологія є важливою для майбутнього масштабування проєкту за межі однієї теплиці.

Мережа Low Power Wide Area Networks, покладається на стандартну і повсюдну бездротову послугу LTE, збагачену протоколами оптимізації для міжмашинного зв'язку і зниження енергоспоживання.

Сучасні підключені пристрої мають різні вимоги, тому необхідно знайти компроміс між швидкістю передачі даних, кращою енергоефективністю та більшим радіусом дії сигналу. Саме тут у гру вступають мережі LPWA з їхніми сигналами більшого радіусу дії та додатковими протоколами для зменшення енергоспоживання пристроїв.

Датчики, монітори, камери та інші пристрої Інтернету речей зазвичай не підключені до постійного джерела живлення і повинні залишатися на полі до десяти років без втручання людини.

Тому вони потребують зв'язку, який може охопити їх по всьому світу, не витрачаючи при цьому енергію.

1.6 Критичний аналіз існуючих комерційних рішень

Ринок пропонує низку готових рішень для моніторингу, однак аналіз виявляє їхні суттєві недоліки для нашої задачі.

– Xiaomi Mi Flora: цей пристрій є популярним серед аматорів, але не є професійним інструментом. Він використовує Bluetooth Low Energy (BLE), що обмежує його радіус дії 10 метрами[7]. Він не має можливостей автоматизації (лише моніторинг), а для інтеграції з системами типу Home Assistant вимагає постійно увімкненого проміжного шлюзу (наприклад, ESP32, що працює як BLE-сканер)[8];

– Netatmo Weather Station: це високоякісне рішення для моніторингу погоди, але воно не орієнтоване на агроавтоматизацію. Система не має виходів для керування виконавчими пристроями (реле) і є закритою екосистемою;

– GreenIQ Smart Garden Hub: це більш професійна система, орієнтована на управління поливом, з підтримкою Wi-Fi та інтеграцією з хмарними сервісами. Однак вона орієнтована на комерційний сегмент, має високу вартість і, як і більшість хмарних рішень, має обмежену функціональність або повністю виходить з ладу за відсутності стабільного інтернет-з'єднання.

Загальні проблеми комерційних рішень:

– висока вартість: початкові інвестиції в комерційні IoT-системи є значним бар'єром;

– залежність від Інтернету: більшість систем спроектовані за принципом "Cloud-First", що робить їх недієздатними в сільській місцевості;

– закриті екосистеми: дані користувача "замкнені" у пропрієтарній хмарі виробника, що унеможливлює інтеграцію з іншими системами або локальний аналіз.

Сучасні IoT-системи часто інтегруються з мобільними додатками або веб-інтерфейсами, що дозволяє аграріям отримувати своєчасні повідомлення про зміни середовища (сповіщення), переглядати графіки історії параметрів та дистанційно керувати виконавчими пристроями (поливом, вентиляцією, освітленням).

Аналіз показує, що на ринку існує незадоволений попит на доступне, надійне та відмовостійке рішення. Жоден з проаналізованих комерційних продуктів не відповідає вимогам автономної роботи в умовах ненадійного зв'язку. Це обґрунтовує необхідність розробки власного програмно-апаратного комплексу на базі гібридної "Edge-Cloud" архітектури та з пріоритетом "Offline-First".

1.7 Сучасний стан автоматизації біоресурсних систем

Управління біоресурсами, зокрема в тепличних господарствах, історично пройшло шлях від ручного керування до локальної автоматизації (SCADA). Сучасний етап характеризується впровадженням кіберфізичних систем (CPS), де фізичні процеси (ріст рослин, клімат) тісно інтегровані з обчислювальними ресурсами.

Ключовими задачами в цій області є:

– точне землеробство (Precision Agriculture): мінімізація витрат води та добрив за рахунок точного дозування на основі реальних потреб рослини, а не розкладу;

– кліматичний контроль: підтримка параметрів середовища (температура, вологість, CO2, освітлення) у вузьких діапазонах, оптимальних для конкретної культури;

– прогнозування врожайності: використання накопичених історичних даних для побудови предиктивних моделей росту.

Проте більшість існуючих систем все ще функціонують ізольовано, не маючи можливості масштабування та глибокої аналітики, що стримує перехід до повноцінного Smart Farming.

1.8 Проблематика та обмеження класичних WSN у агросекторі

Класичні бездротові сенсорні мережі (Wireless Sensor Networks WSN), які є основою більшості сучасних агро-рішень, мають низку критичних обмежень, виявлених у ході аналізу:

– обмежені енергетичні ресурси: сенсорні вузли часто живляться від батарей. Безперервна передача даних по протоколах типу ZigBee або Wi-Fi швидко виснажує джерело живлення, що вимагає частого обслуговування;

– низька обчислювальна потужність: традиційні мікроконтролери (наприклад, 8-бітні AVR) не здатні виконувати складну обробку даних (Edge AI) або надійне шифрування, передаючи "сирі" дані, що перевантажує канал зв'язку;

– жорстка прив'язка до додатків: у класичній WSN мережа будується під одну конкретну задачу (наприклад, тільки полив). Додавання нових

функцій (наприклад, моніторингу хвороб) вимагає розгортання нової паралельної інфраструктури, що економічно невигідно;

– проблема надійності даних: у агресивному середовищі теплиці (висока вологість, температура) дешеві сенсори часто дрейфують або виходять з ладу.

Без інтелектуальних алгоритмів валідації це призводить до хибних спрацювань виконавчих механізмів.

1.9 Еволюція до Sensor-Cloud та Sensing-as-a-Service (Se-aaS)

Для подолання зазначених вище проблем виникла парадигма Sensor-Cloud Computing. Це нова модель, в якій фізичні сенсори віртуалізуються у хмарі.

Суть трансформації:

– фізичний рівень відділяється від логічного. Користувач взаємодіє не з "залізом", а з "віртуальним сенсором" через API;

– з'являється модель Sensing-as-a-Service (Se-aaS): дані з одного фізичного датчика можуть продаватися або надаватися різним споживачам (фермерам, страховим компаніям, науковцям) як сервіс.

Переваги для біоресурсів:

– масштабованість: хмара дозволяє зберігати та обробляти терабайти даних про ріст рослин за кілька сезонів (Big Data), що неможливо на локальному сервері;

– еластичність: обчислювальні ресурси виділяються динамічно.

Наприклад, під час різкої зміни погоди частота опитування та складність аналізу можуть автоматично зростати.

Впровадження Sensor-Cloud Computing безпосередньо сприяє досягненню ключових глобальних цілей сталого розвитку (ЦСР) ООН:

ЦСР 2: Подолання голоду (Food Security). Завдяки точному землеробству (Precision Farming), SCC дозволяє: прогнозувати врожайність з високою точністю; оптимізувати внесення добрив, що підвищує продуктивність без виснаження земель; зменшувати втрати врожаю через раннє виявлення хвороб та шкідників.

ЦСР 6: Чиста вода та належні санітарні умови. Розумне зрошення: сенсори вологості в поєднанні з хмарними прогнозами погоди дозволяють подавати воду лише тоді і туди, де це необхідно. Це економить до 30-50% прісної води.

ЦСР 12: Відповідальне споживання та виробництво. Мінімізація хімікатів: використання пестицидів стає точковим, а не суцільним; прозорість ланцюга постачань – хмарні дані дозволяють відстежити походження біоресурсів ("від поля до столу"), підтверджуючи їх екологічність.

1.10 Огляд існуючих платформ та рішень

Аналіз ринку показує наявність кількох класів рішень:

– промислові платформи (AWS IoT, Azure IoT, Google Cloud IoT): надають потужні інструменти для бекенду (MQTT брокери, бази даних, Serverless функції), але вимагають розробки власного "заліза" та прошивки;

– спеціалізовані агро-платформи (CropIn, FarmBeats): фокусуються на аналітиці супутникових знімків та макро-даних, часто ігноруючи мікрокліматичні параметри конкретної теплиці;

– IoT-конструктори (ThingsBoard, Blynk): дозволяють швидко створювати дашборди, але часто мають обмеження по складності алгоритмів керування (наприклад, реалізація Fuzzy Logic вимагає зовнішніх скриптів).

Існує потреба у розробці гібридної системи, яка б поєднувала гнучкість промислових хмар (AWS) з енергоефективністю спеціалізованих Edge-пристроїв (ESP32) та алгоритмами, адаптованими під фізіологію рослин (VPD, Fuzzy Logic).

2 АНАЛІЗ ТЕХНОЛОГІЙ КОМУНІКАЦІЇ ТА ВИБІР ПРОТОКОЛУ ПЕРЕДАЧІ ДАНИХ

Вибір комунікаційного протоколу є фундаментальним етапом проектування IoT-системи, оскільки він визначає архітектуру взаємодії компонентів, енергоспоживання вузлів та здатність системи працювати в умовах низької пропускної здатності каналу (Low Bandwidth) та високих затримок (High Latency). Для обґрунтування вибору MQTT проведемо детальне порівняння з основними альтернативами: HTTP та CoAP. Для проекту, що включає мікроконтролер ESP32, доцільно застосувати гібридну архітектуру, де протоколи виконують чітко визначені, взаємодоповнюючі ролі.

2.1 Аналіз протоколу HTTP (Hypertext Transfer Protocol) в контексті IoT

Протокол HTTP, що є основою Web, широко використовується в IoT завдяки своїй поширеності та наявності розвинених бібліотек. Він базується на моделі «Запит-Відповідь» (Request-Response) і працює поверх TCP.

Недоліки HTTP для систем моніторингу теплиць:

– синхронна блокуюча модель: клієнт (сенсор) повинен чекати відповіді від сервера. В умовах поганого зв'язку це призводить до блокування основного циклу програми та підвищеного енергоспоживання через тривалу активність радіомодуля;

– надлишковість заголовків (Overhead): HTTP є текстовим протоколом. Кожен запит, навіть якщо він передає лише одне число

(температуру), містить сотні байт заголовків (User-Agent, Content-Type, Accept тощо). Дослідження показують, що енерговитрати на передачу даних через HTTP можуть бути в 10–100 разів вищими, ніж через MQTT, особливо при передачі малих пакетів даних (telemetry);

– відсутність стану (Stateless): сервер не зберігає інформацію про стан з'єднання з клієнтом. Для отримання команд керування (наприклад, увімкнення реле) пристрій змушений постійно опитувати сервер (Polling), що є вкрай неефективним і створює зайве навантаження на мережу;

– складність реалізації двостороннього зв'язку:для реалізації push-повідомлень (миттєва реакція на команду з застосунку) необхідне використання WebSockets або Long Polling, що значно ускладнює прошивку мікроконтролера.

2.2. Аналіз протоколу CoAP (Constrained Application Protocol)

CoAP розроблений IETF спеціально для пристроїв з обмеженими ресурсами (Constrained Devices). Він використовує архітектуру REST, подібну до HTTP, але працює поверх UDP.

Переваги та недоліки: CoAP має дуже низькі накладні витрати та підтримує роботу в режимі сну. Однак, використання UDP (User Datagram Protocol) не гарантує доставки пакетів та порядку їх отримання. Хоча CoAP має механізм підтверджуваних повідомлень (Confirmable messages), надійність у нестабільних мережах значно поступається TCP-рішенням. Крім того, наявність NAT (Network Address Translation) у мобільних мережах створює проблеми для CoAP, оскільки UDP-сесії часто розриваються

маршрутизаторами, що унеможливлює відправку команд на пристрій ззовні без додаткових механізмів keep-alive або використання проксі.

2.3. Обґрунтування вибору MQTT (Message Queuing Telemetry Transport)

MQTT – це бінарний протокол, що працює поверх TCP/IP і використовує модель «Видавець-Підписник» (Publish-Subscribe). Ця модель забезпечує повну розв'язку (decoupling) компонентів системи у просторі та часі, що є ідеальним для архітектури Offline-First[9].

2.3.1. Переваги архітектури Publish/Subscribe

У розробленій системі ESP32 (Видавець) публікує дані у топік (наприклад, greenhouse/sensor/temp), а мобільний застосунок (Підписник) отримує їх. Вони не знають IP-адрес один одного і не повинні бути онлайн одночасно. Брокер виступає буфером. Якщо застосунок офлайн, брокер збереже повідомлення (при правильному налаштуванні) і віддасть їх при підключенні. Це фундаментально вирішує проблему синхронізації[10].

2.3.2. Механізми забезпечення надійності (QoS)

MQTT надає три рівні якості обслуговування (Quality of Service), які дозволяють гнучко балансувати між надійністю та трафіком :

- QoS 0 (At most once): повідомлення відправляється один раз без підтвердження. Використовується для регулярної телеметрії (температура, вологість). Якщо одне значення втрачено, наступне надійде через хвилину, що не є критичним. Це економить трафік;

– QoS 1 (At least once): гарантує доставку повідомлення мінімум один раз. Використовується для критичних подій (спрацювання датчика протікання) та команд керування реле. Мікроконтролер буде повторювати відправку пакету, доки не отримає PUBACK від брокера;

– QoS 2 (Exactly once): гарантує доставку рівно один раз. Використовується рідко через високі затримки (4-етапне рукостискання), для даної системи QoS 1 є достатнім.

2.3.3. Retained Messages та синхронізація стану

Механізм Retained Messages (Утримувані повідомлення) є критичним для UX мобільного застосунку. При публікації статусу реле (greenhouse/relay/1->ON) з прапором retained=true, брокер зберігає це повідомлення. Коли користувач відкриває застосунок, він миттєво підписується на топік і отримує останній відомий стан, навіть якщо ESP32 в цей момент спить або перезавантажується. Це усуває необхідність запитувати статус при старті[11].

2.3.4. Last Will and Testament (LWT)

Для моніторингу доступності системи використовується механізм LWT. При підключенні ESP32 реєструє у брокера повідомлення "Offline" у топік greenhouse/status. Якщо з'єднання розривається некоректно (зникло живлення, обрив мережі) і брокер не отримує keep-alive пінгів, він автоматично публікує це повідомлення. Це дозволяє застосунку миттєво відобразити користувачеві аварійний стан системи.

Таблиця 2.1 – Порівняння протоколів

Характеристика	MQTT	HTTP	CoAP
Модель	Pub/Sub (Асинхронна)	Req/Res (Синхронна)	Req/Res (Асинхронна)
Транспорт	TCP	TCP	UDP
Overhead (Заголовок)	2 байт	>100 байт (текст)	4 байт
Гарантія доставки	Вбудована (QoS 1/2)	Тільки TCP (Retransmission)	Confirmable (на рівні застосунку)
Енергоефективність	Висока (Keep-Alive)	Низька (нові з'єднання)	Дуже висока
Підтримка Offline	Persistent Sessions	Немає (потрібен polling)	Обмежена (Caching)

Протокол MQTT є безальтернативним вибором для проектованої системи завдяки вбудованим механізмам керування станом з'єднання, гарантованої доставки та мінімізації трафіку, що прямо відповідає вимогам архітектури Offline-First.

3 АРХІТЕКТУРА ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Створення великомасштабного IT або інфраструктурного проєкту вимагає не просто плану, а надійної, інтегрованої конфігурації (P-Config), що слугує нерухомим базисом для всіх подальших рішень. Початкова конфігурація проєкту є критично важливим документом, який формалізує стратегічні наміри, технічні зобов'язання, фінансові обмеження та юридичні захисні механізми. Метою цього звіту є створення архітектурного обґрунтування, яке забезпечить цілісність проєкту протягом усього його життєвого циклу.

Система складається з трьох рівнів:

1. Рівень периферії (Edge): мікроконтролер ESP32, що збирає дані з сенсорів та керує реле.
2. Рівень комунікації (Broker): MQTT-брокер, що забезпечує маршрутизацію повідомлень.
3. Рівень користувача (Client): мобільний застосунок на Flutter з локальною базою даних.

3.1. Обґрунтування вибору апаратних компонентів

Вимога до початкової конфігурації проєкту полягає у забезпеченні нерозривного зв’язку між бізнес-цілями та їхньою технічною реалізацією, фінансуванням та управлінням ризиками. Недостатня деталізація на цьому етапі неминуче призводить до розповзання обсягу, фінансових перевитрат та компрометації якості кінцевого продукту.

3.1.1. Мікроконтролер ESP32 WROOM-32

Плата розробника ESP-WROOM-32 ESP-32 30 Pin (рис.3.1) побудована на новому мініатюрному високопродуктивному модулі ESP-WROOM-32 від популярного бренду, призначеного для широкого спектру застосувань, починаючи від мікропотужних мережевих датчиків до складних програм, наприклад, таких як кодування голосу, потокова передача музики та MP3 кодування. На модулі зібрано всю необхідну мінімальну периферію, достатню для швидкого та комфортного старту роботи з ESP-WROOM-32.

Модуль розробника виконаний на базі популярного двоядерного чіпсету ESP32, з тактовою частотою від 80 МГц до 240 МГц, що змінюється, можливістю індивідуального управління і живлення. Модуль розроблений для переносної та автономної електроніки та додатків інтернету речей, виконаний у мініатюрному корпусі 2,5 см х 1,8 см. Мікромодуль має багату периферію, що включає різні інтерфейси, роз'єм для SD карти, інфрачервоний порт, інтерфейс для підключення ємнісної сенсорної панелі.

Однією з переваг модуля є наднизьке споживання електроенергії та гнучкий вибір «сплячих» режимів, що дозволяють отримати цифри до 20 мкА в режимі deep sleep mode. Модуль підтримує три режими роботи: AP+STA, AP та STA[12].

Рисунок 3.1 – Модуль розробника ESP32

На відміну від ESP8266, модуль ESP32 побудований на базі двоядерного процесора Xtensa® LX6. Це дозволяє розділити задачі: одне ядро (Protocol CPU) обслуговує стек Wi-Fi/Bluetooth та MQTT (бібліотека PubSubClient та мережевий стек LwIP), а друге (Application CPU) – виконує логіку опитування датчиків та керування реле. Це критично важливо для стабільності, оскільки блокуючі операції з датчиками не призводять до розриву з'єднання з брокером. Наявність 4 МБ Flash-пам'яті дозволяє використовувати файлову систему для буферизації даних[13].

3.1.2. Сенсор BME280 (Температура, вологість, тиск)

Використання модуля GY-BME280(рис.3.2) є оптимальним рішенням для теплиці. На відміну від дешевих датчиків DHT11/DHT22, BME280 використовує інтерфейс I2C, що забезпечує вищу завадостійкість та швидкість опитування. Він має високу точність (±0.5°C, ±3% RH) та малий час реакції, що важливо для систем автоматичної вентиляції.

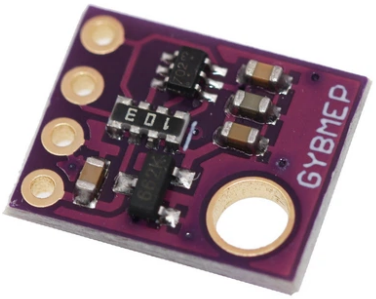


Рисунок 3.2 – Датчик атмосферного тиску, температури та вологості, модуль GY-BME280/BMP280 5B

BME280 – це комбінований цифровий датчик вологості, тиску та температури, що базується на перевірених принципах вимірювання. Модуль датчика розміщений у надзвичайно компактному корпусі LGA з металевою кришкою, розміром лише 2,5 × 2,5 мм² та висотою 0,93 мм. Його невеликі розміри та низьке енергоспоживання дозволяють використовувати його в пристроях, що працюють від батарей, таких як мобільні телефони, GPS-модулі

або годинники. BME280 є сумісним за регістрами та характеристиками з цифровим датчиком тиску Bosch Sensortec BMP280.

Датчик вологості забезпечує надзвичайно швидкий час відгуку для застосувань, що вимагають швидкого реагування на контекст, та високу загальну точність у широкому діапазоні температур.

Датчик тиску є абсолютним барометричним датчиком тиску з надзвичайно високою точністю та роздільною здатністю і значно нижчим рівнем шуму, ніж Bosch Sensortec BMP180. Вбудований датчик температури був оптимізований для найнижчого рівня шуму та найвищої роздільної здатності. Його вихід використовується для температурної компенсації датчиків тиску та вологості, а також може використовуватися для оцінки температури навколишнього середовища [14].

3.1.3 Ємнісний датчик вологості ґрунту (Capacitive Soil Moisture Sensor v1.2)

Ємнісний датчик вологості ґрунту(рис.3.3) відрізняється від більшості резистивних датчиків, представлених на ринку. Він використовує принцип ємнісного зондування для визначення вологості ґрунту. Проблема легкої корозії резистивного датчика усунена, а термін його експлуатації значно подовжений. Датчик має вбудований мікросхему стабілізатора напруги, яка підтримує робоче середовище з широким діапазоном напруги 3,3~5,5 В, що означає, що він може нормально працювати навіть на головній платі Arduino 3,3 В. Знаковий інтерфейс DFRobot-Gravity забезпечує сумісність інтерфейсу і може бути безпосередньо підключений до плати розширення Gavity IO. Мікрокомп'ютер, такий як Raspberry Pi, для роботи потребує лише зовнішнього модуля перетворення ADC (аналогового сигналу в цифровий). За

допомогою зовнішнього екрану та материнської плати ви можете поговорити зі своєю рослиною, щоб дізнатися, чи не хоче вона пити і чи не потребує вона трохи більше води[15].

Рисунок 3.3 – Ємнісний датчик вологості ґрунту 3.3В для Arduino

На відміну від резистивних датчиків, ємнісні не піддаються швидкій корозії через електроліз. Датчик вимагає індивідуального калібрування ("повітря" vs "вода") для перетворення "сирих" значень АЦП (0-4095) у відсотки вологості, оскільки показники залежать від конкретного екземпляра та напруги живлення[16].

Найбільша проблема резистивних датчиків вологості грунту – малий термін експлуатації, обумовлений схильністю корозії контактів вимірювача. Ємнісні датчики вільні від цього недоліку, а корозійностійке покриття

електродів робить їх практично вічними. Ще це позитивно впливає на стабільність показань та точність вимірювання датчика.

3.1.4. 4-канальний модуль реле (5V)

Модуль реле використовується для комутації силового навантаження (насос поливу, вентилятор, фітолампи). Використання 5-вольтового модуля з ESP32 (3.3В логіка) вимагає уваги. Більшість модулів мають опторозв'язку (оптрони PC817). Схема підключення «Active Low» дозволяє керувати реле, подаючи логічний «0» (GND) на вхід, що безпечно для ESP32. Для повної гальванічної ізоляції мікроконтролера від силової частини рекомендується використовувати окреме живлення для котушок реле (знявши перемичку JD-VCC).

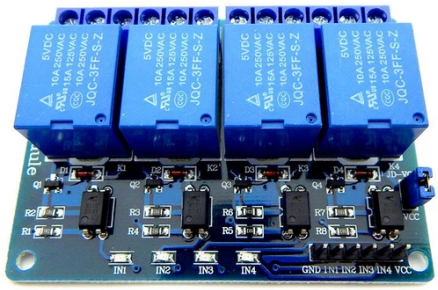


Рисунок 3.4 – Реле 4-х канальний модуль 5V для Arduino

Це низькорівнева 4-канальна релейна плата інтерфейсу 5В, де кожен канал потребує струму драйвера 15-20 мА. Вона може використовуватися для

керування різними приладами та обладнанням з великим струмом. Вона оснащена реле високого струму, які працюють під напругою 250 В змінного струму 10 А або 30 В постійного струму 10 А. Вона має стандартний інтерфейс, який може керуватися безпосередньо мікроконтролером. Цей модуль оптично ізольований від сторони високої напруги для забезпечення безпеки, а також запобігає утворенню контуру заземлення при підключенні до мікроконтролера[17].

3.2 Фізична реалізація та схема підключення апаратних компонентів

Розробка апаратної частини системи виконана на макетній платі (breadboard) для перевірки схемотехнічних рішень та налагодження програмного забезпечення перед інтеграцією в кінцевий корпус (рис. 3.5). Схема комутації розроблена з урахуванням електричних характеристик мікроконтролера ESP32 (логіка 3.3В) та периферійних пристроїв.

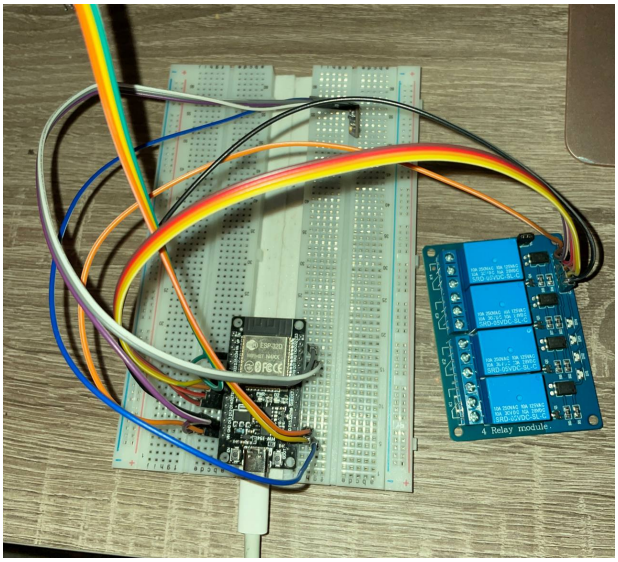


Рисунок 3.5 Лабораторний стенд

Живлення макету здійснюється через інтерфейс USB type-C, що забезпечує стабілізацію напруги до 3.3В внутрішнім LDO-регулятором плати ESP32 для живлення ядра та сенсорів. Релейний модуль отримує пряме живлення 5В з піну VIN, оскільки котушки реле вимагають напруги 5В для спрацьовування.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВБУДОВАНОЇ ЛОГІКИ

Еволюція систем автоматизації в агропромисловому секторі (Smart Agriculture) демонструє фундаментальний зсув від апаратно-орієнтованих рішень до програмно-визначених систем (Software-Defined Systems). Якщо раніше логіка керування мікрокліматом теплиці або системою зрошення жорстко задавалася фізичною конфігурацією реле та аналогових компараторів, то сучасні вбудовані системи (Embedded Systems) базуються на мікроконтролерах високої інтеграції (SoC), таких як ESP32 або STM32. Програмна реалізація вбудованої логіки (Embedded Software Implementation) в цьому контексті стає критичним фактором, що визначає не лише функціональність, але й надійність, енергоефективність та економічну доцільність розгортання IoT-рішень.

Специфіка аграрних застосувань накладає унікальні вимоги на програмне забезпечення: пристрої часто працюють у віддалених локаціях з нестабільним живленням та зв'язком, обслуговують біологічні об'єкти з високою інерційністю та чутливістю до умов середовища, і повинні функціонувати автономно місяцями або роками без втручання оператора. У цьому звіті представлено вичерпний аналіз архітектурних патернів, алгоритмічних стратегій та методів забезпечення надійності вбудованого ПЗ, що базується на сучасному стані досліджень та інженерних практик.

Розробка прошивки виконана в середовищі PlatformIO з використанням C++. Це дозволяє структурувати проект краще, ніж Arduino IDE, та зручно керувати бібліотеками.

4.1. Алгоритм роботи Offline-First на стороні ESP32

Головною вимогою є збереження даних при втраті зв'язку. Для цього реалізовано механізм Store-and-Forward (Збережи та Перешли).

При роботі в офлайн-режимі пристрій повинен накопичувати дані, використовувати файлові системи (SPIFFS, LittleFS) або вбудовані бази даних (SQLite для потужніших систем) для буферизації телеметрії, при відновленні зв'язку має відбуватись передача накопичених даних. Важливо реалізувати механізми вирішення конфліктів (Conflict Resolution), наприклад, стратегію "Last Write Wins" (перемагає останній запис) або злиття на основі векторних годинників, якщо параметри змінювалися і локально, і віддалено.

Для забезпечення доступності інтерфейсу при втраті з'єднання з роутером, система повинна автоматично переходити в режим Точки Доступу (AP Mode). Це створює локальну Wi-Fi мережу, до якої користувач може підключитися напряму зі смартфону для діагностики або ручного керування.

Лістинг 4.1 – Реалізація надійної підписки та LWT (фрагмент логіки):

```
if (client.connect(clientId.c_str(), mqttUser, mqttPass,
                   "greenhouse/status", 1, true, "offline")) {
    // Публікація статусу Online (Retained)
    client.publish("greenhouse/status", "online", true);
    // Підписка на топіки керування (QoS 1)
    client.subscribe("greenhouse/control/relay/#", 1);
}
```

4.3. Керування реле та зворотний зв'язок

При отриманні команди через MQTT, ESP32 змінює стан піна, а потім обов'язково публікує новий стан у топік статусу (greenhouse/state/relay/1). Це забезпечує зворотний зв'язок: користувач бачить

у застосунку не те, що він "натиснув кнопку", а те, що "реле реально перемкнулось".

Програмна реалізація вбудованої логіки для агропромислових IoT-систем є комплексною інженерною дисципліною, що вимагає інтеграції знань з теорії автоматичного керування, архітектури обчислювальних систем та мережевих технологій. Перехід від простих циклічних алгоритмів до подієво-орієнтованих RTOS-архітектур, впровадження глибоких механізмів самодіагностики (Watchdogs) та прийняття філософії Local-First є ключовими факторами створення надійних, масштабованих та автономних рішень. Успіх розробки залежить не лише від вибору апаратної платформи, але й від глибини розуміння та коректного застосування розглянутих програмних патернів, що дозволяє мінімізувати ризики та забезпечити безперебійну роботу в реальних умовах експлуатації.

5 ФУНКЦІОНУВАННЯ СИСТЕМИ ТА ВЗАЄМОДІЯ КОМПОНЕНТІВ

Розроблена система являє собою розподілений кіберфізичний комплекс, що функціонує за принципом Edge Computing (обчислення на периферії). На відміну від класичних хмарних систем, де рішення приймаються на сервері, у даній розробці центр прийняття рішень перенесено безпосередньо на мікроконтролер ESP32. Це забезпечує критично важливу для агросектору властивість – детермінованість керування незалежно від стану мережі.

5.1. Інформаційний обмін та синхронізація (Offline-First)

Взаємодія між теплицею та користувачем побудована на асинхронній моделі Publish/Subscribe протоколу MQTT. Це дозволяє розв'язати проблему синхронізації в умовах нестабільного зв'язку.

5.1.1 Штатний режим (Online)

ESP32 раз на 10 секунд формує JSON-пакет (publishData()) з поточними показниками та публікує його у топік greenhouse/sensors, мобільний застосунок на Flutter, підписаний на цей топік, миттєво оновлює UI та кешує дані у локальну SQLite базу даних, команда користувача (наприклад, "Увімкнути світло") відправляється у топік greenhouse/relay/control. ESP32 отримує повідомлення через callback-функцію, змінює стан GPIO та, що критично важливо, публікує підтвердження нового стану назад у топік статусу.

5.1.2 Втрата зв'язку (Offline Mode)

ESP32 виявляє втрату з'єднання з MQTT-брокером (!mqttClient.connected()), функція applyAutomation() продовжує виконуватися. Якщо температура впаде, обігрівач увімкнеться, навіть якщо "інтернету немає", дані телеметрії не відкидаються, а записуються у внутрішню Flash-пам'ять (через LittleFS) або зберігаються у кільцевому буфері в RAM (якщо втрата зв'язку короткочасна), при відновленні зв'язку (reconnectMQTT()), система автоматично публікує накопичені дані. Застосунок, використовуючи механізм offline_records, завантажує "пропущену історію" для побудови безперервних графіків.

5.2 Забезпечення відмовостійкості (Resilience)

Для підвищення надійності в системі реалізовано кілька рівнів захисту:

– сторожовий таймер (Watchdog Timer): ESP32 має вбудований Task WDT. У разі "зависання" головного циклу (наприклад, через помилку в I2C драйвері BME280), WDT перезавантажить контролер, відновивши роботу системи автоматично;

– Keep-Alive та LWT: використання механізму MQTT Keep-Alive (налаштовано на 60 с у main.dart та PubSubClient) дозволяє брокеру детестувати "тиху смерть" пристрою (втрату живлення). У такому випадку брокер автоматично відправить повідомлення "Offline" у топік greenhouse/system/status (Last Will and Testament), і інтерфейс користувача змінить колір індикатора на червоний;

– Fallback AP Mode: у функції setupWiFi() реалізовано логіку перемикання режимів. Якщо ESP32 не може підключитися до основного

роутера (STA), вона автоматично піднімає власну точку доступу (AP) ESP32_Greenhouse. Це дозволяє користувачу підійти до теплиці і зчитати дані напряму, навіть якщо глобальна мережа відсутня.

Розроблена архітектура поєднує надійність промислових ПЛК (завдяки локальній автоматиці на ESP32) із зручністю сучасних IoT-рішень (завдяки мобільному застосунку та хмарній синхронізації). Використання MQTT як транспортного шару є оптимальним рішенням, що забезпечує низьку латентність, економію трафіку та вбудовану підтримку нестабільних мереж.

5.3 Реалізація Firmware Edge-вузла

Програмне забезпечення для ESP32 розроблено в середовищі Arduino IDE з використанням C++. Код структуровано для роботи в асинхронному режимі з використанням переривань та таймерів.

Нижче наведено фрагмент коду, що відповідає за безпечне підключення до AWS IoT Core з використанням сертифікатів.

Лістинг 5.1. – Ініціалізація захищеного з'єднання MQTT

```
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include "secrets.h" // Містить сертифікати: AWS_CERT_CA,
AWS_CERT_CRT, AWS_CERT_PRIVATE

#define AWS_IOT_PUBLISH_TOPIC   "greenhouse/esp32/telemetry"
#define AWS_IOT_SUBSCRIBE_TOPIC "greenhouse/esp32/control"

WiFiClientSecure net = WiFiClientSecure();
PubSubClient client(net);

void connectAWS() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

```
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status()!= WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  client.setServer(AWS_IOT_ENDPOINT, 8883);
  client.setCallback(messageHandler); // Callback для вхідних
повідомлень

  Serial.print("Connecting to AWS IoT");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }

  if (!client.connected()) {
    Serial.println("AWS IoT Timeout!");
    return;
  }

  client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
  Serial.println("\nAWS IoT Connected!");
}
```

Для забезпечення інтероперабельності системи (Sensing-as-a-Service) та зручності обробки великих масивів даних, розроблено уніфіковану структуру корисного навантаження. Дані передаються у форматі JSON (JavaScript Object Notation), що дозволяє легко інтегрувати нові поля без порушення зворотної сумісності.

Лістинг 5.2. – Приклад JSON Payload

```
{
  "temperature": 22.5,
  "humidity": 65.3,
  "pressure": 1013.25,
  "soil_moisture": 45,
```

```
  "timestamp": 1703001234
}
```

Така висока частота дискретизації (0.1 Гц) дозволяє реалізувати алгоритми швидкого реагування на критичні зміни мікроклімату, наприклад, при раптовому відключенні вентиляції або прориві системи поливу.

6 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ТЕСТУВАННЯ

6.1 Локальне сховище даних

Для реалізації локального зберігання даних (Fog/Edge layer) обрано реляційну базу даних SQLite версії 3.51. SQLite є вбудованою бібліотекою, що не потребує окремого процесу сервера, що критично для ресурсів мікроконтролера або одноплатного комп'ютера (Raspberry Pi/ESP32). Забезпечує цілісність даних при раптовому відключенні живлення, що є типовим ризиком для тепличних господарств. Дозволяє використовувати складні аналітичні запити (агрегація, фільтрація) безпосередньо на пристрої перед відправкою у хмару.

6.1.1 Таблиця телеметрії

Дана таблиця є основною для зберігання часових рядів (Time Series Data) параметрів мікроклімату.

Лістинг 6.1. Створення таблиці sensor_readings

```
CREATE TABLE sensor_readings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    temperature REAL,
    humidity REAL,
    pressure REAL,
    soil_moisture INTEGER
);
```

6.1.2 Таблиця подій виконавчих механізмів (relay_events)

Фіксація дій автоматики (вмикання поливу, вентиляції) необхідна для аналізу ефективності алгоритмів керування.

Лістинг 6.2. Створення таблиці подій

```
CREATE TABLE relay_events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    relay_number INTEGER,
    relay_name TEXT,
    state INTEGER,
    triggered_by TEXT,
    reason TEXT
);

CREATE INDEX idx_relay_events_timestamp
    ON relay_events(timestamp DESC);
```

Текстові поля дозволяють реалізувати детальний аудит ("Хто увімкнув?"). Наприклад, triggered_by = 'FuzzyLogicController', reason = 'VPD > 1.2 kPa'. Це дозволяє розрізняти автоматичні спрацювання та ручне втручання оператора. state INTEGER зберігає стан реле (0 або 1), що є стандартом для цифрових виходів мікроконтролерів.

6.1.3 Системні логи та налаштування

Для діагностики роботи пристрою та збереження конфігурації використовуються допоміжні таблиці.

Лістинг 6.3 Таблиці логів та налаштувань

```
CREATE TABLE system_logs (
```

```
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    level TEXT,
    message TEXT,
    details TEXT
);

CREATE TABLE system_settings (
    key TEXT PRIMARY KEY,
    value TEXT,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Використання key TEXT PRIMARY KEY дозволяє гнучко додавати нові параметри (наприклад, mqtt_broker_ip, irrigation_threshold) без зміни схеми бази даних (без ALTER TABLE). Це спрощує оновлення прошивки.

6.1.4 Вставка та вибірка даних

Лістинг 6.4 Вставка даних

```
INSERT INTO sensor_readings (temperature, humidity, pressure,
soil_moisture)
VALUES (22.5, 65.3, 1013.25, 45);
```

Використовується стандартний SQL Syntax. Поле timestamp заповнюється автоматично (DEFAULT CURRENT_TIMESTAMP), що гарантує точний час запису моменту потрапляння даних у базу, незалежно від затримок сенсора.

Лістинг 6.5 Аналітичні запити (Агрегація)

```
SELECT
    DATE(timestamp) as date,
    AVG(temperature) as avg_temp,
    AVG(humidity) as avg_humidity,
    AVG(soil_moisture) as avg_soil
```

```
FROM sensor_readings
WHERE timestamp >= datetime('now', '-7 days')
GROUP BY DATE(timestamp)
ORDER BY date DESC;
```

Використання вбудованих функцій AVG() та GROUP BY перекладає навантаження з обчислення статистики з коду програми (Python/C++) на рушій бази даних, який оптимізований для таких операцій. Це зменшує обсяг даних, що передаються між диском та пам'яттю.

6.2 Реалізація застосунку

Застосунок реалізовано з використанням архітектурного патерну MVVM (Model-View-ViewModel), адаптованого під реактивну природу Flutter через механізм ChangeNotifier. Це дозволяє чітко розділити логіку обробки даних (Business Logic), стан інтерфейсу (UI State) та відображення (Widgets).

6.2.1 Рівень управління станом (State Management)

Центральним елементом архітектури є клас GreenhouseController, який виступає в ролі ViewModel. Він інкапсулює:

1. MQTT-клієнт: відповідає за підтримку постійного з'єднання з брокером, підписку на топіки та автоматичне перепідключення.
2. API-клієнт: виконує HTTP-запити до сервера для отримання історичних даних, які занадто великі для передачі через MQTT.
3. Локальний кеш: використовує SharedPreferences для збереження останніх відомих значень сенсорів. Це дозволяє миттєво відображати дані при запуску застосунку ще до встановлення з'єднання з мережею.

6.2.2 Реалізація гібридної передачі даних

Застосунок використовує гібридну модель отримання даних:

– гарячі дані (Hot Data): поточні показники температури, вологості та стани реле надходять через MQTT в реальному часі. Це забезпечує мінімальну затримку (Latency < 200ms) для оперативного реагування;

– холодні дані (Cold Data): історичні графіки та статистика за тиждень/місяць завантажуються через REST API. Це розвантажує MQTT-канал від передачі великих масивів даних (Bulk Data Transfer).

6.3 Аналіз критично важливих фрагментів коду

Нижче наведено пояснення ключових алгоритмічних рішень, використаних у проекті, з обґрунтуванням їх необхідності для стабільності системи.

6.3.1. Неблокуюча багатозадачність на ESP32

Це фундаментальний патерн для embedded-систем, відомий як Cooperative Multitasking (Кооперативна багатозадачність) на базі системного таймера millis().

Лістинг 6.6 Читання датчиків кожні 10 сек

```
if (millis() - lastSensorRead >= SENSOR_READ_INTERVAL) {
  lastSensorRead = millis();
  readSensors();
```

```
  applyAutomation();
  publishData();
}
```

Використання функції delay(10000) для паузи між вимірюваннями повністю зупинило б процесор. У цей час мікроконтролер не міг би обробляти вхідні MQTT-повідомлення (mqttClient.loop()) або підтримувати Wi-Fi з'єднання, що призвело б до розриву зв'язку (Connection Timeout). Використання різниці часу дозволяє головному циклу loop() прокручуватися тисячі разів на секунду. Це гарантує, що функція mqttClient.loop() викликається часто, підтримуючи "серцебиття" (keep-alive) протоколу MQTT та миттєво реагуючи на команди керування реле, навіть у проміжках між зчитуванням сенсорів.

Використання StaticJsonDocument замість DynamicJsonDocument є критичним для стабільності мікроконтролера.

Лістинг 6.7 Ефективна серіалізація даних

```
void publishData() {
  if (!mqttConnected) return;
  StaticJsonDocument doc; // Фіксований буфер у стеку
  doc["temperature"] = currentSensorData.temperature;
  //... додавання інших даних
  String output;
  serializeJson(doc, output);
  mqttClient.publish("greenhouse/sensors", output.c_str());
}
```

StaticJsonDocument виділяє пам'ять у стеку (Stack), а не в купі (Heap). У вбудованих системах, які працюють 24/7, динамічне виділення пам'яті (malloc/free) часто призводить до фрагментації купи (Heap Fragmentation). З часом це викликає непередбачувані перезавантаження пристрою. Розмір 384

байти розраховано заздалегідь. Це запобігає переповненню пам'яті (Stack Overflow) та гарантує, що JSON-пакет завжди сформується коректно.

6.3.2. Реактивна обробка вхідних даних у Flutter

Тут реалізовано асинхронний прослуховувач (Stream Listener).

Лістинг 6.8 Stream Listener

```
client!.updates!.listen((List<MqttReceivedMessage<MqttMessage>>
messages) {
  for (var message in messages) {
    final payload = MqttPublishPayload.bytesToStringAsString(
        (message.payload as
MqttPublishMessage).payload.message);
    handleMessage(topic, payload);
  }
});
```

Потокова обробка: Flutter отримує дані через потік (Stream). Це означає, що інтерфейс не "заморожується" в очікуванні даних. Як тільки пакет приходить від брокера, спрацьовує callback-функція. Отриманий бінарний пейлоад (bytes) конвертується у рядок, а потім розбирається як JSON. Це дозволяє мобільному застосунку розуміти структуровані дані від ESP32.

6.3.3. Стратегія кешування "Offline-First"

Цей блок коду реалізує принцип Persistence (Персистентність).

Лістинг 6.9 Persistence

```
// Збереження
Future<void> saveDataToCache() async {
```

```
  final prefs = await SharedPreferences.getInstance();
  await prefs.setDouble('temperature', temperature);
  //...
}

// Відновлення при старті
Future<void> loadDataFromCache() async {
  final prefs = await SharedPreferences.getInstance();
  temperature = prefs.getDouble('temperature')?? 0.0;
  notifyListeners(); // Оновлення UI
}
```

При холодному старті (перезапуску) застосунку без інтернету користувач побачив би нулі або порожній екран, що створює враження несправності системи. Метод loadDataFromCache() викликається до спроби підключення до мережі. Він миттєво заповнює UI останніми відомими даними. Виклик notifyListeners() повідомляє віджетам Flutter, що дані змінилися, ініціюючи перемальовування екрану. Це ключова вимога UX для систем, що працюють у полях з поганим покриттям.

6.4 Програмна реалізація інтерфейсу користувача

Клієнтська частина системи розроблена на фреймворку Flutter, що дозволило створити кросплатформний застосунок з реактивним інтерфейсом. Архітектура застосунку базується на патерні Provider, який забезпечує відділення бізнес-логіки (GreenhouseController) від шару відображення (UI).

6.4.1 Інформаційна панель та відображення телеметрії (Dashboard)

Головний екран застосунку (Рис.6.1) реалізує концепцію «єдиного вікна» для моніторингу стану теплиці. Інтерфейс поділено на логічні блоки: статус

підключення, показники мікроклімату та статус виконавчих механізмів. Для відображення даних використовуються віджети SensorCard.

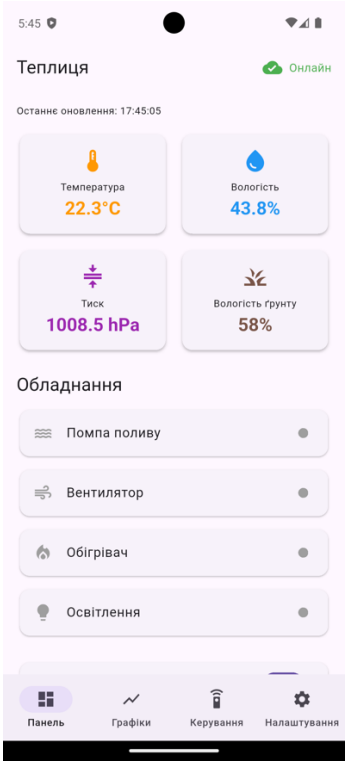


Рисунок 6.1 Головний екран

Кожна картка містить піктограму, назву параметра та його поточне значення. Колірне кодування (помаранчевий для температури, синій для вологості) покращує ергономіку сприйняття. Дані на екрані оновлюються реактивно. Клас GreenhouseController розширює ChangeNotifier. Коли через MQTT надходить нове повідомлення, метод handleMessage оновлює змінні стану та викликає notifyListeners(), що змушує Flutter перемалювати віджети.

6.4.2 Візуалізація історичних даних (Charts)

Екран графіків (Рис. 6.2) дозволяє користувачеві оцінити динаміку змін параметрів. Використовується бібліотека fl_chart для побудови кривих.

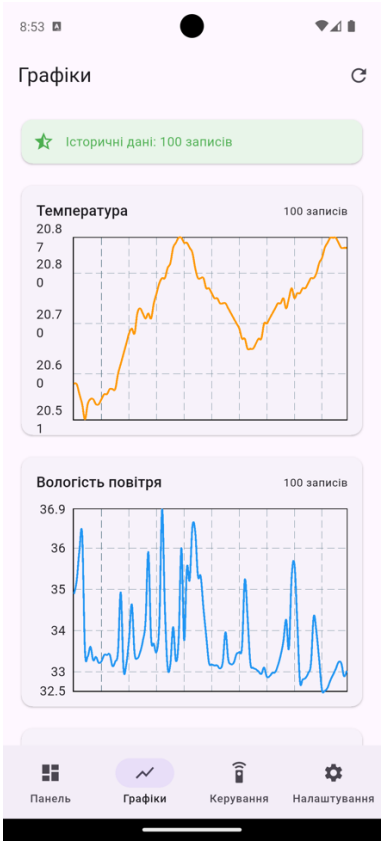


Рисунок 6.2 Екран графіків

Лістинг 6.10 Накопичення даних для графіків

```
// main.dart: Додавання нових точок у список історії
temperatureHistory.add(SensorRecord(now, temperature));

// Обмеження розміру буфера (FIFO) для економії пам'яті
смартфону
if (temperatureHistory.length > maxHistoryLength) {
  temperatureHistory.removeAt(0);
}
```

Система використовує гібридний підхід до даних: при завантаженні екрану виконується HTTP-запит до API для отримання архіву (GreenhouseAPI.getLatestReadings). Нові точки, що надходять через MQTT, динамічно додаються до графіка без необхідності перезавантаження всієї історії.

6.4.3 Дистанційне керування та зворотний зв'язок (Control)

Екран керування (рис. 6.3) надає прямий доступ до реле. Реалізовано перемикачі (SwitchListTile) для помпи, вентилятора, обігрівача та освітлення. У коді реалізовано захист від розсинхронізації інтерфейсу та реального стану пристрою. Коли користувач натискає перемикач, застосунок відправляє команду в топік greenhouse/relay/control.

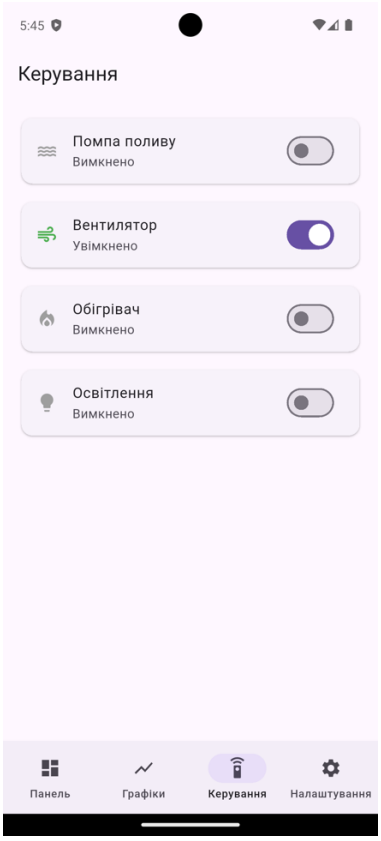


Рисунок 6.3 Екран керування

Коли користувач натискає перемикач, застосунок відправляє команду в топік greenhouse/relay/control.

Лістинг 6.11 Відправка команд

```
// main.dart: Формування та відправка команди
void controlRelay(int relayNumber, bool state) {
  if (!isConnected) return; // Захист від дій офлайн
(опціонально)

  // Формування JSON пейлоаду
  final message = json.encode({
    'relay$relayNumber': state,
  });

  // Публікація в MQTT топік
  client!.publishMessage(
    'greenhouse/relay/control',
```

```
    MqttQos.atLeastOnce, // QoS 1 гарантує доставку
    builder.payload!,
  );

  // Оптимістичне оновлення UI для миттєвої реакції
  switch (relayNumber) {
    case 1: relay1 = state; break;
    //...
  }
  notifyListeners();
}
```

Таким чином, розроблений мобільний застосунок є повнофункціональним інструментом, що повністю відповідає вимогам до сучасних IoT-систем агропромислового призначення: він є автономним, реактивним та ергономічним.

ВИСНОВКИ

У кваліфікаційній роботі запропоновано відмовостійку, автономну архітектуру комплексної системи моніторингу теплиці, яка здатна функціонувати в умовах нестабільного покриття глобальної мережі, що дозволяє суттєво підвищити якість цифрового моніторингу та управління біоресурсами на підставі Sensor-Cloud Computing, яка нівелює залежність від стабільності інтернет-каналу.

Проведений порівняльний аналіз протоколів прикладного рівня показав, що для систем агромоніторингу з обмеженим каналом зв'язку протокол MQTT є безальтернативним вибором. Він забезпечує мінімізацію накладних витрат трафіку (overhead заголовка – 2 байти) та гарантовану доставку повідомлень завдяки механізмам Quality of Service (QoS). Доведено, що використання архітектури Publish/Subscribe, на відміну від Request/Response (HTTP/CoAP), дозволяє ефективно розв'язати проблему синхронізації пристроїв, які не знаходяться в мережі одночасно.

Розроблено та впроваджено концепцію повної автономності на двох рівнях системи. Використання файлової системи LittleFS на мікроконтролері ESP32 дозволило реалізувати буферизацію телеметрії під час втрати зв'язку. Алгоритм «Store-and-Forward» забезпечує збереження цілісності історичних даних без втрат пакетів. Мобільний застосунок на Flutter з локальною базою даних SQLite забезпечує миттєвий доступ до інтерфейсу та історії показників незалежно від наявності інтернету, що значно покращує користувацький досвід (UX) порівняно з хмарно-залежними аналогами.

Впровадження механізму Retained Messages вирішило проблему розсинхронізації стану інтерфейсу користувача та фізичних виконавчих механізмів. При підключенні до мережі застосунок автоматично отримує останній актуальний статус реле, що виключає необхідність надлишкового опитування пристрою. Використання механізму Last Will and Testament (LWT) дозволило реалізувати автоматичний моніторинг "здоров'я" системи та миттєве сповіщення користувача про аварійне відключення живлення або зв'язку.

Підтверджено ефективність використання двоядерної архітектури SoC ESP32 для задач реального часу. Розроблена прошивка на C++ з використанням неблокуючих алгоритмів (cooperative multitasking) забезпечує стабільну роботу мережевого стеку паралельно з опитуванням датчиків та керуванням реле. Реалізація гістерезису в алгоритмах автоматичного керування дозволила уникнути часових осциляцій («джиттеру») виконавчих механізмів, подовжуючи термін їх експлуатації.

Практична цінність роботи полягає у створенні завершеного прототипу системи, готового до масштабування та впровадження у фермерських господарствах, де проблема стабільності інтернет-з'єднання є критичною. Система забезпечує автономне підтримання мікроклімату, мінімізуючи ризики втрати врожаю через технічні збої.

На основі аналізу функціоналу розробленого прототипу визначено такі перспективні напрямки модернізації:

1. Розширення інтерфейсів доступу: розробка Web-панелі для керування системою з десктопних браузерів без необхідності встановлення ПЗ.

2. Хмарна аналітика: інтеграція з базами даних часових рядів (наприклад, InfluxDB) або хмарними платформами (Firebase/AWS IoT) для

довгострокового зберігання великих масивів даних та доступу до них з будь-якої точки світу.

3. Інтелектуальне керування: впровадження алгоритмів машинного навчання (Machine Learning) для прогнозування потреб рослин у поливі на основі накопиченої історії та прогнозу погоди.

4. Візуальний моніторинг: інтеграція модуля камери (на базі ESP32-CAM) для візуального контролю стану рослин та виявлення шкідників.

5. Масштабування сенсорної мережі: підключення додаткових датчиків для вимірювання рівня CO_2, освітленості (Lux) та кислотності ґрунту (pH) для більш точного контролю агротехнічних процесів.

СПИСОК ДЖЕРЕЛ ПОСИЛАНЬ

1. Saiz-Rubio V., Rovira-Más F. From Smart Farming towards Agriculture 5.0: A Review on Crop Data Management. Agronomy. 2020. T. 10, № 2. C. 207.

2. Ayoub Shaikh T., Rasool T., Rasheed Lone F. Towards leveraging the role of machine learning and artificial intelligence in precision agriculture and smart farming. Computers and Electronics in Agriculture. 2022. Vol. 198. P. 107119.

3. Irwin D., Shenoy P., Cecchet E., Zink M. Resource management in data-intensive clouds: Opportunities and challenges. Proceedings of IEEE Workshop on Local and Metropolitan Area Networks (LANMAN), Long Branch, NJ, (2010), pp. 1–6.

4. Misra S., Tiwari V., Obaidat M. S. LACAS: Learning Automata-Based Congestion Avoidance Scheme for Healthcare Wireless Sensor Networks. IEEE Journal on Selected Areas in Communications, vol. 27, no. 4, (2009), pp. 466–479.

5. Atzori L., Iera A., Morabito G. The Internet of Things: A survey. *Comput. Netw.*, vol. 54, no. 15, (2010), pp. 2787-2805.

6. LoRaWAN vs Zigbee: Differences, Applications, and Pros & Cons - Minew, accessed on November 6, 2025, https://www.minew.com/lorawan-vs-zigbee/

7. Conclusion - Xiaomi MI Flora Sensor, https://www.thesmarthomeblog.com/post/review-xiaomi-mi-flora-sensor/conclusion/

8. After 6 weeks my Mi Flora sensors finally arrived from China! Can hardly wait to automate killing my plants using this ratty data. : r/homeassistant - Reddit, https://www.reddit.com/r/homeassistant/comments/mxqqrw/after_6_weeks_my_mi_flora_sensors_finally_arrived/

9. MQTT [Електронний ресурс] // Wikipedia. – URL: https://en.wikipedia.org/wiki/MQTT

10. Goodrich R. S. Message Queuing Telemetry Transport Protocol – How it Works [Electronic resource] // InSight: Rivier Academic Journal. – 2022. – Vol. 17, no. 1. – URL: https://www2.rivier.edu/journal/ROAJ-Fall-2022/J1204_Goodrich_2022.pdf

11. MQTT 5.0 Feature: Retained Message [Електронний ресурс] // EMQ. – URL: https://www.emqx.com/en/blog/mqtt5-features-retain-message

12. Плата розробника ESP-WROOM-32 ESP32 30 Pin (Wi-Fi + Bluetooth) [Електронний ресурс] // ArduShop. – URL: https://ardushop.in.ua/arduino/developer-board-esp-wroom-32-esp-32-wi-fi-bluetooth

13. Bosch Sensortec. BME280 : Environmental sensor : datasheet [Electronic resource]. – Document number: BST-BME280-DS001-10. – Revision 1.1. – 2015. – URL: https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf

14. Fasate A. Optimizing Storage on Embedded Systems: A Guide to SPIFFS, LittleFS, and FAT [Electronic resource] // Engineering IoT : [Medium blog]. – 2023. – 14 Sept. – URL: https://medium.com/engineering-iot/explore-about-the-file-system-n-controller-spiffs-fat-little-fs-62c2b5924591

15. HW-101 HW Moisture Sensor V1.0 : datasheet [Electronic resource]. – 2022. – 9 Sept. – URL: https://www.datocms-assets.com/28969/1662716326-hw-101-hw-moisture-sensor-v1-0.pdf

16. Soil Moisture and pH Measurement System with Temperature Compensation [Electronic resource] : Circuit Note CN-0398 // Analog Devices. – 2016. – URL: https://www.analog.com/media/en/reference-design-documentation/reference-designs/cn0398.pdf

17. 4 Channel 5V Optical Isolated Relay Module [Electronic resource] : User Guide // HandsOn Technology. – URL: https://www.handsontec.com/dataspecs/4Ch-relay.pdf

18. Gabdola A., Jatayev S., Khassanova G. Drought tolerance in chickpea: physiological–biochemical aspects and laboratory assessment methods. Proceedings of the XVI International Scientific and Practical Conference. Seville, Spain. 2025. Pp. 11-13 URL: https://isg-konf.com/trends-in-the-development-of-science-through-the-creation-of-new-technologies/

ДОДАТОК А

Графічний матеріал до кваліфікаційної роботи

Sensor-Cloud Computing у системі біоресурсів

Харківський національний університет радіоелектроніки

Кваліфікаційна магістерська робота за спеціальністю "Комп'ютерна інженерія"

Кафедра АПОТ

Магістрант: Мовчан К. М.

Керівник: професор Чумаченко С.В.

2025 рік

1

Актуальність

Виклики агроіндустрії

Комп'ютерна інженерія надає інструменти для подолання викликів пов'язаних із зміною клімату, обмеженими ресурсами.

Зміни клімату

Непередбачувані погодні умови та екстремальні явища ставлять під загрозу традиційне землеробство, вимагаючи нових підходів.

Цілі сталого розвитку ООН

Проєкт безпосередньо підтримує 6 Цілей сталого розвитку ООН: від подолання голоду до боротьби зі зміною клімату. Економія 50% води та підвищення врожайності на 20%.

Вартість рішень

Існуючі високотехнологічні системи часто є занадто дорогими або мають обмежену автономність без постійного Інтернет-з'єднання.

2

Мета та задачі дослідження

Мета:

Метою роботи є суттєве підвищення якості цифрового моніторингу та управління біоресурсами шляхом створення відмовостійкої, автономної архітектури системи керування теплицею на підставі Sensor-Cloud Computing, яка нівелює залежність від стабільності інтернет-каналу.

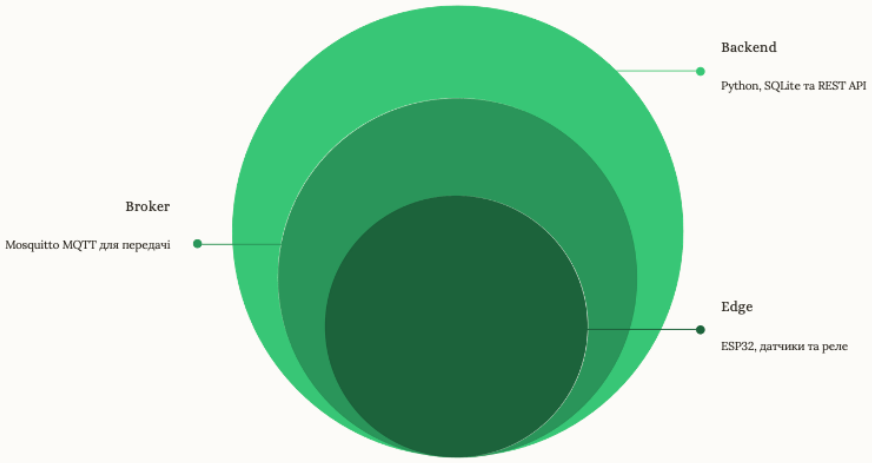
Задачі:

- Аналіз існуючих комерційних та DIY рішень на ринку.
- Проектування гібридної архітектури системи для оптимальної роботи.
- Розробка апаратної платформи на базі **ESP32**.
- Реалізація firmware з використанням протоколу **MQTT**.
- Створення серверної частини (backend) на базі **Mosquitto та SQLite**.
- Розробка мобільного застосунку **Flutter** для взаємодії з системою.
- Комплексне тестування та валідація функціональності системи.

3

Архітектура гібридної системи

Розроблена система складається з чотирьох інтегрованих шарів, що забезпечують комплексне управління мікрокліматом теплиці.



4

Обґрунтування вибору MQTT

Розроблена система складається з чотирьох інтегрованих шарів, що забезпечують комплексне управління мікрокліматом теплиці.

- HTTP: Великий overhead, синхронний (блокує роботу), немає збереження стану.
- CoAP: UDP ненадійний без додаткових механізмів.
- MQTT: Легкий (заголовок 2 байти), асинхронний, QoS (гарантія доставки).

Висновок: MQTT ідеальний для слабких мереж (Low Bandwidth, High Latency).

Архітектура MQTT та Інформаційний Обмін

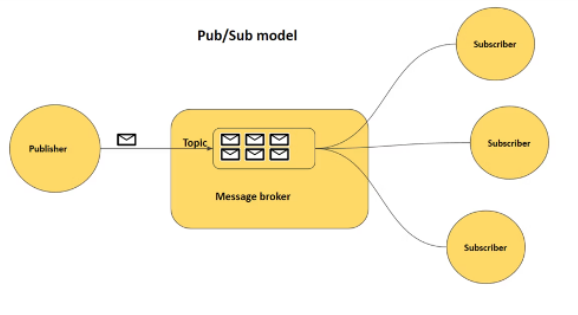


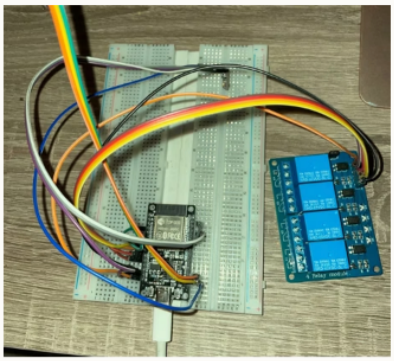
Схема взаємодії (Візуальна частина):

- **Видавець (Publisher):** ESP32 (Сенсори, Статус реле).
- **Брокер (Broker):** Центральний вузол маршрутизації (Local/Cloud).
- **Підписник (Subscriber):** Мобільний застосунок Flutter (Панель керування).

Технологічний стек: Апаратна частина (Hardware)

Апаратна частина (Hardware):

- **ESP32:** Високопродуктивний мікроконтролер (240 MHz Dual-Core) для обробки даних.
- **BME280:** Датчик для точного вимірювання температури, вологості та атмосферного тиску.
- **Soil sensor:** Датчик вологості грунту для оптимізації поливу.
- **4-CH Relay Module:** Модуль реле для керування зовнішнім обладнанням (освітлення, вентиляція, полив).



7

Технологічний стек: Програмна частина (Software)

Програмна частина (Software):

- **Протокол:** MQTT v3.1.1 для ефективної передачі повідомлень.
- **Backend:** Python з фреймворком Flask та базою даних SQLite.
- **Frontend:** Мобільний застосунок на Flutter з використанням мови Dart.
- **Notifications:** Локальні push-сповіщення для оперативного інформування.

8

Ключові алгоритми системи

Офлайн-буферизація даних

Система зберігає до 100 останніх записів вимірювань, забезпечуючи безперервність збору даних навіть за відсутності Інтернету.

Автоматичне управління реле

Запрограмовані правила дозволяють автономно керувати освітленням, вентиляцією та поливом на основі даних датчиків.

Синхронізація при підключенні

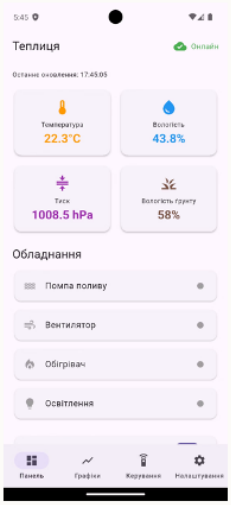
При відновленні зв'язку, всі буферизовані дані автоматично синхронізуються з центральним сервером.

Запобігання спаму сповіщень

Реалізовано механізм cooldown (15 хв) для запобігання надмірним push-сповіщенням при стабільних відхиленнях.

9

Демонстрація функціоналу

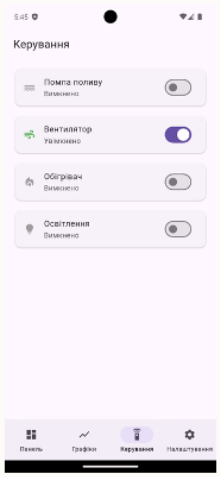


Ключові можливості:

- Моніторинг у реальному часі: Доступ до актуальних даних з датчиків у будь-який момент.
- Історичні графіки: Візуалізація змін мікроклімату за певний період.
- Керування 4 реле: Дистанційне управління обладнанням теплиці.
- Push-сповіщення: Миттєві повідомлення про критичні
- події або відхилення.
- Офлайн-режим: Безперебійна робота системи та збереження даних без доступу до Інтернету.

10

Демонстрація функціоналу(керування реле)



Реле 1 (Помпа):
- Вмикається: вологість ґрунту < 30%
- Вимикається: вологість > 70%
- Cooldown: 1 година між циклами

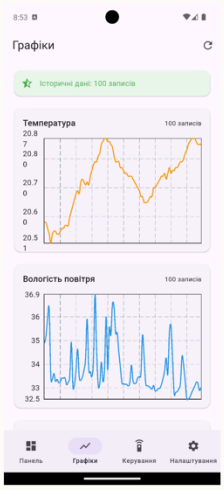
Реле 2 (Вентилятор):
- Вмикається: температура > 28°C
- Вимикається: температура < 24°C

Реле 3 (Обігрівач):
- Вмикається: температура < 15°C
- Вимикається: температура > 22°C
- Взаємоблокування з вентилятором!

Реле 4 (Освітлення):
- За часом: 06:00-20:00

11

Демонстрація функціоналу(графіки)



Гібридна архітектура даних:

- *Історія:* Завантаження архівних даних через REST API (для великих періодів).
- *Real-time:* Миттєве додавання нових точок через MQTT потік без перезавантаження сторінки.

Технічна реалізація:

- Використання бібліотеки `fl_chart` для високопродуктивного рендерингу у Flutter.
- Адаптивні осі координат та автоматичне масштабування.

Offline-доступ:

- Графіки будуються на основі даних з локальної БД **SQLite** при відсутності мережі.
- Синхронізація «прогалин» у даних відбувається у фоновому режимі при відновленні з'єднання.

12

Наукова новизна та перспективи

Вдосконалена архітектура системи

система використовує сучасні досягнення комп'ютерної інженерії – вбудовані системи, IoT-протоколи, розподілені бази даних – для створення доступного рішення автоматизації.

Офлайн-буферизація

Унікальний механізм офлайн-буферизації даних з автоматичною синхронізацією при відновленні зв'язку, мінімізуючи втрати даних.

13

Результати роботи

- Запропоновано відмовостійку, автономну архітектуру комплексної системи моніторингу теплиці, яка здатна функціонувати в умовах нестабільного покриття глобальної мережі.
- Створено апаратно-програмний комплекс на базі **ESP32** та **Flutter**, що працює автономно.
- Забезпечено повну автономність: дані не втрачаються при збоях мережі.
- Використання протоколу **MQTT** знизило накладні витрати трафіку (overhead) у **~8 разів** порівняно з HTTP.
- Механізм **Retained Messages** усунув затримки синхронізації UI при запуску застосунку.

14

ДОДАТОК Б


INTERNATIONAL
SCIENCE GROUP
Certificate
OF PARTICIPATION
is awarded to
Мовчан Ксенія Миколаївна
for active involvement in
XVI International Scientific and Practical Conference
«TRENDS IN THE DEVELOPMENT OF SCIENCE THROUGH
THE CREATION OF NEW TECHNOLOGIES»
December 16-19, 2025, Seville, Spain
24 Hours of Participation
(0,8 ECTS credits)
Organizing committee
SIGNATURE
Google Scholar
Bowker
OUCI
Open Ukrainian Citation Index
doi
Crossref
Content Registration

«Sensor-Cloud Computing у системі біоресурсів»

	Прізвище та ініціали відповідальної особи	Підпис	Дата
Роботу виконав студент групи СКСм-24-1. Структура кваліфікаційної роботи: – пояснювальна записка 60 с.; – графічний матеріал 14 арк.	Мовчан К.М.		19.12.2025
Керівник роботи	Чумаченко С.В.		19.12.2025
Перевірка на антиплагіат здійснено, відповідальна особа	Литвинова Є.І.		20.12.2025
Нормоконтроль проведено :	Чумаченко С.В.		19.12.2025